Engineering with Computers (2023) 39:3691–3720
https://doi.org/10.1007/s00366-022-01779-3

ORIGINAL ARTICLE

Scalable computational kernels for mortar finite element methods

Matthias Mayr[1,2] · **Alexander Popp**[1]

Received: 7 February 2022 / Accepted: 22 December 2022 / Published online: 25 January 2023

Abstract
Targeting simulations on parallel hardware architectures, this paper presents computational kernels for efficient computations in mortar finite element methods. Mortar methods enable a variationally consistent imposition of coupling conditions at high accuracy, but come with considerable numerical effort and cost for the evaluation of the mortar integrals to compute the coupling operators. In this paper, we identify bottlenecks in parallel data layout and domain decomposition that hinder an efficient evaluation of the mortar integrals. We then propose a set of computational strategies to restore optimal parallel communication and scalability for the core kernels devoted to the evaluation of mortar terms. We exemplarily study the proposed algorithmic components in the context of three-dimensional large-deformation contact mechanics, both for cases with fixed and dynamically varying interface topology, yet these concepts can naturally and easily be transferred to other mortar applications, e.g. classical meshtying problems. To restore parallel scalability, we employ overlapping domain decompositions of the interface discretization independent from the underlying volumes and then tackle parallel communication for the mortar evaluation by a geometrically motivated reduction of ghosting data. Using three-dimensional contact examples, we demonstrate strong and weak scalability of the proposed algorithms up to 480 parallel processes as well as study and discuss improvements in parallel communication related to mortar finite element methods. For the first time, dynamic load balancing is applied to mortar contact problems with evolving contact zones, such that the computational work is well balanced among all parallel processors independent of the current state of the simulation.

Keywords Mortar methods · Contact mechanics · Interface problems · Parallel algorithms · Finite elements · Domain decomposition

1 Introduction

Mortar finite element methods (FEM) are nowadays well established in a variety of application areas in computational science and engineering as discretization technique for the coupling of non-matching meshes. Their general applicability in a vast range of problems as well as their mathematical properties, e.g. variational consistency, make them one of the most popular choices among interface discretization techniques. They are undoubtedly the most preferred choice for robust finite element discretization in computational contact mechanics undergoing large deformations [16, 61, 62, 80, 82]. However, the numerical effort and computational cost is high and can be considered a bottleneck in many scenarios. This paper discusses several performance challenges of mortar methods in the context of parallel computing and proposes remedies to reduce the overall runtime, obtain optimal scalability as well as reduce parallel communication and memory consumption. As a demanding prototype application, several test cases from computational contact mechanics showcase the proposed algorithms and their impact on runtime and parallel scalability.

Originally being developed in the context of domain decomposition for the weak imposition of interfacial constraints [5, 10], mortar methods soon became popular in meshtying [59, 60] and contact mechanics problems [6, 34, 53, 56–58, 62, 63, 85, 86]. Recently, mortar methods for

✉ Matthias Mayr
matthias.mayr@unibw.de

Alexander Popp
alexander.popp@unibw.de

[1] Institute for Mathematics and Computer-Based Simulation, Universität der Bundeswehr München, Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany

[2] Data Science & Computing Lab, Universität der Bundeswehr München, Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany

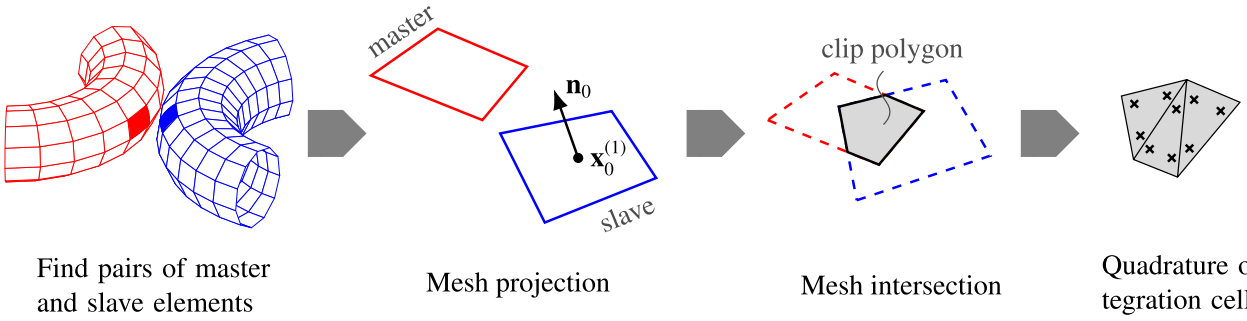


Fig. 1 Main steps of 3D mortar coupling (from left to right): Pairs of master and slave elements, that (i) are potentially in contact, need to be (ii) projected onto each other along the normal vector $\mathbf{n}_0$ to (iii) compute the mesh intersection and to (iv) perform numerical quadrature of mortar contributions on integration cells

meshtying problems have regained attention due to the rise of isogeometric analysis and the need for isogeometric patch coupling [20, 21, 23, 24, 41, 83, 89]. A variety of papers discusses mortar methods in the context of isogeometric analysis for contact problems, among them [16, 17, 19, 25, 66]. Moreover, mortar methods have spread to other single-field problems, e.g. contact mechanics including wear [30] or fluid dynamics [26], as well as a variety of surface-coupled multi-physics problems, among them fluid-structure interaction [42, 47, 52] or the simulation of lithium-ion cells in electrochemistry [27]. Lately, also volume-coupled problems have been addressed by mortar methods [29]. Despite their significant computational cost, the popularity of mortar methods over classical node-to-segment, Gauss-point-to-segment, and other collocation-based approaches is based on their mathematical properties such as their variational consistency and stability. Compared to two-dimensional problems, an efficient mortar evaluation is much more critical in three-dimensional problems, which are at the same time of great practical relevance in real-world applications.

When using a Lagrange multiplier field $\underline{\lambda}$ to impose constraints on the subdomain interfaces, mortar methods discretize $\underline{\lambda}$ on the so-called slave *side* of the interface. The numerical effort of mortar methods is usually related to the search for nearest neighbors, local projection of meshes and subsequent clipping and triangulation of intersected meshes, as well as the resulting segment-based numerical integration, cf. Fig. 1. While these operations themselves are already expensive, implicit contact solvers need to perform them in *every nonlinear iteration*, rendering this a possible feasibility bottleneck or at least a performance impediment, which becomes even more demanding through the necessity of consistent linearizations of all mortar terms. The parallelization of contact search algorithms has been addressed in [36] for example, where standard domain-decomposition-based spatial search is enhanced with thread-level parallelism. To speed-up the subsequent evaluation of contact terms, various integration strategies are available, among them *element-based* and *segment-based* integration, cf. [12, 28, 51, 74]. Segment-based integration subdivides each slave element into segments having no discontinuities of the integrands within their domain. This yields a highly accurate quadrature, though is computationally expensive. Element-based integration on the other hand reduces the effort of clipping and triangulation the intersected meshes by employing higher-order integration schemes to deal with weak discontinuities at element edges, though brings along a less accurate evaluation of the mortar integrals. While the segment-based integration strategy is unequivocally preferable due to its accuracy, it comes at significantly higher computational cost. Furthermore, systems of linear equations arising from mortar-based interface discretizations require tailored preconditioning techniques for an efficient iterative solution procedure. Depending on the specific details of the discretization, the resulting linear system might exhibit saddle-point structure. Efficient preconditioners to be used in conjunction with Krylov solvers are available in literature [1, 14, 68, 70–73, 78] and, thus, are not in the scope of this paper. We rather focus on the cost of evaluating all mortar-related terms.

As outlined previously, many theoretical aspects of mortar methods have already been discussed and solved in the literature, e.g. the choice of discrete basis functions [31, 49, 50, 57, 58, 77, 81], numerical quadrature [12, 28, 51, 74], conservation laws [39, 40, 86], or contact search algorithms [8, 75, 76, 84, 85, 87, 88]. However, computational aspects of mortar methods for contact problems—especially in the context of parallel computing—have largely been neglected by the scientific community so far. To fill this gap, this work is motivated and guided by the quest for parallel scalability of *all* algorithmic components of mortar methods for arbitrarily evolving contact zones in three-dimensional problems. Therefore, we analyze the computational kernels of mortar finite element methods and design their interplay to assure parallel scalability. To the best of our knowledge, most contributions in literature have focused on the serial

case (i.e. one processor) only or have embedded mortar methods into existing parallel finite element codes without specific provisions. An exception to this observation is the work of Krause and Zulian [48], where a parallel approach to the variational transfer of discrete fields between unstructured finite element meshes as well as the associated proximity and intersection detections are described in detail and examples for the evaluation of grid projection operators are given for various surface and volume projection problems. Yet, Krause and Zulian [48] spare dynamic contact problems with evolving contact zones, which are of particular importance in engineering applications. In the present contribution, we analyze several schemes to subdivide mortar interface discretizations into subdomains suitable for parallel computing and discuss their interplay with distributed memory architectures of computing clusters to achieve parallel scalability. Thereby, we follow a message-passing parallel programming model that utilizes the message passing interface (MPI) for communication between address spaces of different processes [54]. Finally, we develop and showcase a dynamic load balancing strategy to address the particular needs of contact problems with evolving contact configurations and interface topologies for three-dimensional problems.

By starting from an analysis of the computational cost of the evaluation of mortar terms, which is most commonly related to the slave side of the contact interface, we identify three main tasks, which will directly lead to the postulation of two essential requirements for parallel and scalable computational kernels for mortar finite element methods:

- For the geometrical task of identifying close master and slave nodes within the contact search, each slave node needs access to the position of every node of the master side of the interface discretization. While the distribution of the master interface discretization to several compute nodes enables larger problem sizes, it requires advanced ghosting (i.e. sending data between different processors) of interface quantities to reduce the overall communication and memory footprint. We will propose ghosting strategies that take a measure of geometric proximity between master and slave nodes into account to precompute and reduce the list of master nodes/elements to be communicated.
- To efficiently parallelize the evaluation of mortar terms, we will start from a baseline approach where interfacial subdomains are aligned with the subdomains of the underlying bulk domain. This method is straightforward to implement, preserves data locality, and reduces communication between parallel processes. However, it does not include all processes in the evaluation of the mortar terms and, thus, is not scalable. We will then devise strategies for redistributing the interface domain decomposition to increase parallel efficiency and scalability of the mortar evaluation.
- As the contact configuration and area often changes over the course of a simulation, we will propose a dynamic load balancing scheme. Therefore, we will monitor characteristic quantities of the parallel evaluation of all mortar terms and will trigger an adaptation of the interface domain decomposition if the current state and computational behavior of the simulation indicate a deterioration of parallel performance.

We will discuss these approaches in detail and demonstrate their scaling behavior and applicability to large three-dimensional problems. Although our current work studies scalable computational kernels for mortar methods in the context of classical finite element analysis, all findings are equally valid for isogeometric mortar methods (i.e. NURBS-based interface discretizations).

The remainder of this paper is organized as follows: After a brief description of the contact problem, its discretization, and suitable solution techniques in Sect. 2, the implications of storing mortar discretizations on distributed memory machines will be discussed in Sect. 3. Domain decomposition approaches for an efficient evaluation of the mortar integrals will then be developed in Sect. 4. Section 5 presents several numerical studies to assess communication patterns and demonstrate the parallel scalability of the proposed methods in the context of computational contact mechanics, before we conclude with some final remarks in Sect. 6.

2 Problem formulation and finite element discretization

While mortar methods are applicable to a broad spectrum of problems and partial differential equations (PDEs), finite deformation contact problems are nowadays certainly one of the most appealing and challenging application areas for mortar methods in computational mechanics. Hence, we focus on contact problems now, but keep the generality of mortar evaluations in mind.

2.1 Governing equations

In general, mortar methods allow for the coupling of several physical domains governed by PDEs through enforcing coupling conditions at various coupling surfaces or interfaces. Without loss of generality, we focus our presentation on the two-body contact problem with bodies $\Omega_0^{(1)}$ and $\Omega_0^{(2)}$ which potentially come into frictionless contact along their contact boundaries $\Gamma_*^{(1)}$ and $\Gamma_*^{(2)}$, respectively. Each subdomain $\Omega_0^{(i)}, i \in \{1, 2\}$ is governed by the initial boundary value problem of finite deformation elasto-dynamics and is subject

to the Hertz–Signorini–Moreau conditions for frictionless contact, reading

$$\begin{aligned}
\mathrm{Div}\underline{\mathbf{P}}^{(i)} + \underline{\bar{\mathbf{b}}}_0^{(i)} &= \rho_0^{(i)}\underline{\ddot{\mathbf{u}}}^{(i)} && \text{in } \Omega_0^{(i)} \times [0,T],\\
\underline{\mathbf{u}}^{(i)} &= \underline{\bar{\mathbf{u}}}^{(i)} && \text{on } \Gamma_{\mathrm{D},0}^{(i)} \times [0,T],\\
\underline{\mathbf{P}}^{(i)}\underline{\mathbf{n}}_0^{(i)} &= \underline{\bar{\mathbf{h}}}_0^{(i)} && \text{on } \Gamma_{\mathrm{N},0}^{(i)} \times [0,T],\\
\underline{\mathbf{u}}^{(i)}(\underline{\mathbf{X}}^{(i)},0) &= \underline{\bar{\mathbf{u}}}^{(i)}(\underline{\mathbf{X}}^{(i)}) && \text{in } \Omega_0^{(i)},\\
\underline{\dot{\mathbf{u}}}^{(i)}(\underline{\mathbf{X}}^{(i)},0) &= \underline{\bar{\dot{\mathbf{u}}}}^{(i)}(\underline{\mathbf{X}}^{(i)}) && \text{in } \Omega_0^{(i)},\\
g_n \geq 0, p_n \leq 0, p_n g_n &= 0 && \text{on } \Gamma_* \times [0,T]
\end{aligned}$$

with the unknown displacement field $\underline{\mathbf{u}}$, the first Piola–Kirchhoff stress tensor $\underline{\mathbf{P}}$, the body force vector $\underline{\bar{\mathbf{b}}}_0$, density ρ_0, normal vector $\underline{\mathbf{n}}_0$, and traction vector $\underline{\bar{\mathbf{h}}}_0$. Prescribed boundary values on the Dirichlet boundaries $\Gamma_{\mathrm{D},0}^{(i)}$ and Neumann boundaries $\Gamma_{\mathrm{N},0}^{(i)}$ as well as any initial values are marked with $\bar{(\bullet)}$. First and second time derivatives are given as $\dot{(\bullet)}$ and $\ddot{(\bullet)}$, respectively. The reference configuration is distinguished from the current configuration by the subscript $(\bullet)_0$. Furthermore, p_n refers to the contact pressure acting in the normal direction of the contact interface Γ_* in the current configuration, while the gap function g_n denotes the normal distance between the two bodies in the current configuration. To later distinguish between the two sides of the contact interface, we follow the traditional naming scheme and refer to $\Gamma_*^{(1)}$ carrying the Lagrange multiplier as so-called "slave" side Γ_*^{sl}, while $\Gamma_*^{(2)}$ denotes the "master" side Γ_*^{ma}.

Since this paper is concerned with the efficient evaluation of the mortar terms on parallel computing clusters, we will detail the discretization of all mortar-related terms in Sect. 2.2. However, to keep the focus tight and concise, we refer to the extensive literature for any further details on the finite element formulation and discretization [31, 49, 50, 57, 58, 77, 81], the solution of the nonlinear problem via active set strategies [37, 43, 45, 46, 56], as well as for details on the structure of the arising linear systems of equations and efficient solvers thereof [1, 14, 68, 70–73, 78].

2.2 Discretization

To perform the spatial discretization with FEM, we assume the existence of a weak form of the contact mechanics problem summarized in Sect. 2.1. For the additional terms arising in contact mechanics, a Lagrange multiplier field $\underline{\boldsymbol{\lambda}}$ is introduced into the weak form to enforce the contact constraints, leading to a mixed method with a variational inequality, where both the primal field $\underline{\mathbf{u}}$ as well as the dual variable $\underline{\boldsymbol{\lambda}}$ need to be discretized in space.

For the sake of a concise presentation, we skip the details of the FEM applied to the three-dimensional solid bodies $\Omega_0^{(i)}, i \in \{1,2\}$. Considering the contact interface, we adopt from the volume discretization the isoparametric concept with the parameter coordinate $\boldsymbol{\xi} = [\xi_1, \xi_2]$ and the shape functions $N_k(\boldsymbol{\xi})$ defined at node k of all $n^{(1)}$ nodes on the discrete slave surface $\Gamma_{*,\mathrm{h}}^{\mathrm{sl}}$ and $N_\ell(\boldsymbol{\xi})$ defined at node ℓ of all $n^{(2)}$ nodes on the discrete master surface $\Gamma_{*,\mathrm{h}}^{\mathrm{ma}}$, respectively. The interpolation of the displacement field on element level is then given as

$$\mathbf{u}^{(1)}(\boldsymbol{\xi},t) = \sum_{k=1}^{n^{(1)}} N_k(\boldsymbol{\xi})\mathbf{u}_k(t), \quad \mathbf{u}^{(2)}(\boldsymbol{\xi},t) = \sum_{\ell=1}^{n^{(2)}} N_\ell(\boldsymbol{\xi})\mathbf{u}_\ell(t). \tag{1}$$

As usual in mortar methods, the Lagrange multiplier field $\underline{\boldsymbol{\lambda}}$ is discretized on $m^{(1)}$ nodes of the discrete slave surface $\Gamma_{*,\mathrm{h}}^{\mathrm{sl}}$, reading

$$\boldsymbol{\lambda}(\boldsymbol{\xi},t) = \sum_{j=1}^{m^{(1)}} \boldsymbol{\Phi}_j(\boldsymbol{\xi})\boldsymbol{\lambda}_j(t), \tag{2}$$

where $\boldsymbol{\Phi}_j(\boldsymbol{\xi})$ denotes the Lagrange multiplier shape function at node j. Thereby, either standard or dual shape functions can be used.

Inserting (1) and (2) into the contact virtual work $\delta\mathcal{W}_\lambda = \int_{\Gamma_*} \underline{\boldsymbol{\lambda}} \left(\delta\underline{\mathbf{u}}^{\mathrm{sl}} - \delta\underline{\mathbf{u}}^{\mathrm{ma}}\right) \mathrm{d}\Gamma$ yields

$$\begin{aligned}
\delta\mathcal{W}_\lambda \approx \delta\mathcal{W}_{\lambda,\mathrm{h}} = &\sum_{j=1}^{m^{(1)}}\sum_{k=1}^{n^{(1)}} \boldsymbol{\lambda}_j^{\mathrm{T}} \overbrace{\left[\int_{\Gamma_{*,\mathrm{h}}^{\mathrm{sl}}} \boldsymbol{\Phi}_j N_k^{(1)} \,\mathrm{d}\Gamma\right]}^{\mathcal{D}[j,k]} \delta\mathbf{u}_k^{(1)}\\
&- \sum_{j=1}^{m^{(1)}}\sum_{\ell=1}^{n^{(2)}} \boldsymbol{\lambda}_j^{\mathrm{T}} \underbrace{\left[\int_{\Gamma_{*,\mathrm{h}}^{\mathrm{sl}}} \boldsymbol{\Phi}_j \left(N_\ell^{(2)} \circ \chi_{\mathrm{h}}\right) \mathrm{d}\Gamma\right]}_{\mathcal{M}[j,\ell]} \delta\mathbf{u}_\ell^{(2)}.
\end{aligned} \tag{3}$$

The mortar matrices $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{\mathcal{M}}$ associated with the slave and master side of the coupling interface are then assembled from the nodal blocks $\boldsymbol{\mathcal{D}}[j,k]$ and $\boldsymbol{\mathcal{M}}[j,\ell]$ defined in (3), respectively. In general, both $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{\mathcal{M}}$ are rectangular matrices. If $m^{(1)} = n^{(1)}$ (which is common practice except for a few cases, e.g. higher-order FEM [49, 50, 58]), $\boldsymbol{\mathcal{D}}$ becomes square. Furthermore, if $\boldsymbol{\Phi}_j$ are chosen as so-called dual shape functions that satisfy a biorthogonality relationship with the standard shape functions N_j, then $\boldsymbol{\mathcal{D}}$ becomes a diagonal matrix and, thus, easy and computationally cheap to invert [31, 49, 50, 58, 65, 77, 79, 81].

We stress that both summands in (3) contain integrals over the slave side $\Gamma_{*,\mathrm{h}}^{(1)}$ of the discrete coupling surface, where the discretization is indicated by the additional subscript $(\bullet)_{\mathrm{h}}$. A suitable discrete mapping $\chi_{\mathrm{h}} : \Gamma_{*,\mathrm{h}}^{\mathrm{ma}} \to \Gamma_{*,\mathrm{h}}^{\mathrm{sl}}$ from the master side to the slave side of the coupling interface is required, because the discrete coupling surfaces $\Gamma_{*,\mathrm{h}}^{\mathrm{ma}}$ and $\Gamma_{*,\mathrm{h}}^{\mathrm{sl}}$ do not coincide anymore in general, especially when considering non-matching meshes on

curved interfaces. These projections are usually based on a continuous field of normal vectors defined on the slave side $\Gamma^{\mathrm{sl}}_{*,\mathrm{h}}$, cf. [56, 86].

We note that the mortar matrices $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{\mathcal{M}}$ also occur in the discrete representation of the Hertz–Signorini–Moreau conditions, cf. [57] for example.

2.3 Evaluation of mortar integrals

In general, the evaluation of both $\boldsymbol{\mathcal{D}}\left[j,k\right]$ and $\boldsymbol{\mathcal{M}}\left[j,\ell\right]$ in (3) requires information from both the discrete slave interface $\Gamma^{\mathrm{sl}}_{*,\mathrm{h}}$ and the discrete master interface $\Gamma^{\mathrm{ma}}_{*,\mathrm{h}}$. Firstly, this inevitably involves the discrete mapping χ_{h} to project finite element nodes and quadrature points between slave and master sides. In practice, mortar integration is often performed on a piecewise flat geometrical approximation of the slave surface $\Gamma^{\mathrm{sl}}_{*,\mathrm{h}}$ as proposed in [59]. For further details and an in-depth mathematical analysis, see [18, 61, 62]. Secondly, the slave-sided integration domain $\Gamma^{\mathrm{sl}}_{*,\mathrm{h}}$ has to be split into so-called mortar segments, such that both $\boldsymbol{\Phi}^{(1)}_{j}$ and $N^{(2)}_{\ell}$ are C^1-continuous on these segments, as kinks in the function to be integrated would deteriorate the achievable accuracy of the numerical quadrature. These mortar segments are arbitrarily shaped polygons, which will then be decomposed into triangles to perform quadrature. While the evaluation of $\boldsymbol{\mathcal{D}}\left[j,k\right]$ involves quantities solely defined on the slave interface $\Gamma^{\mathrm{sl}}_{*,\mathrm{h}}$, the evaluation of $\boldsymbol{\mathcal{M}}\left[j,\ell\right]$ requires to integrate the product of master side shape functions $N^{(2)}_{\ell}$ and slave side shape functions $\boldsymbol{\Phi}^{(1)}_{j}$ over the discrete slave interface $\Gamma^{\mathrm{sl}}_{*,\mathrm{h}}$.

Algorithm 1: Segment-based mortar integration for three-dimensional problems (cf. Figure 1)

for *each slave element* $e^{(\iota),\mathrm{sl}}$ **do**
  Construct normal vector $\mathbf{n}_0$ based on slave element center $\mathbf{x}^{(1)}_0$
  for *each master element* $e^{(\tau),\mathrm{ma}}$ *in the vicinity of the current slave element* $e^{(\iota),\mathrm{sl}}$ **do**
    Project $e^{(\tau),\mathrm{ma}}$ and $e^{(\iota),\mathrm{sl}}$ onto each other along $\mathbf{n}_0$
    Find clip polygon $\mathcal{P}$ as intersection $e^{(\tau),\mathrm{ma}} \cap e^{(\iota),\mathrm{sl}}$ via a clipping algorithm, see e.g. [32]
    Divide $\mathcal{P}$ into integration cells $\{\mathcal{T}\}$ with $n^{\mathrm{gp,sl}}$ quadrature points per integration cell $\mathcal{T}$
    `/* Perform Gauss quadrature with` $n^{\mathrm{gp,sl}}$ `quadrature points per integration cell` $\mathcal{T}$ `*/`
    for *each integration cell* $\mathcal{T} \in \{\mathcal{T}\}$ **do**
      for *each quadrature point* $\mathrm{gp} \in \{0,\ldots,n^{\mathrm{gp,sl}}-1\}$ **do**
        Get parameter coordinate η_{gp} within current integration cell $\mathcal{T}$
        Project into slave element's parameter space to obtain $\xi^{(1)}\left(\eta_{\mathrm{gp}}\right)$
        Project into master element's parameter space to obtain $\xi^{(2)}\left(\eta_{\mathrm{gp}}\right)$
        Evaluate and sum integrands in (3) to obtain $\mathcal{D}\left[j,k\right]$ and $\mathcal{M}\left[j,\ell\right]$
      end
    end
  end
end

Algorithm 1 outlines the necessary steps to perform segmentation and numerical quadrature for the interaction of slave and master elements. While we summarize the most important steps of the integration procedure here to highlight its tremendous numerical effort, we refer to [59] for a detailed description of all steps outlined in Algorithm 1. Although segment-based quadrature as described in Algorithm 1 undoubtedly delivers the highest achievable accuracy for the numerical integration of $\boldsymbol{\mathcal{D}}\left[j,k\right]$ and $\boldsymbol{\mathcal{M}}\left[j,\ell\right]$ in three dimensions, it comes at high computational expenses related to mesh projection and intersection, subsequent triangulation as well as numerical quadrature. In practice and also in the present work, both mortar operators $\boldsymbol{\mathcal{D}}\left[j,k\right]$ and $\boldsymbol{\mathcal{M}}\left[j,\ell\right]$ are usually evaluated using segment-based integration to guarantee conservation of linear momentum [59]. More efficient but possibly less accurate integration algorithms have been discussed in [12, 28, 51, 74].

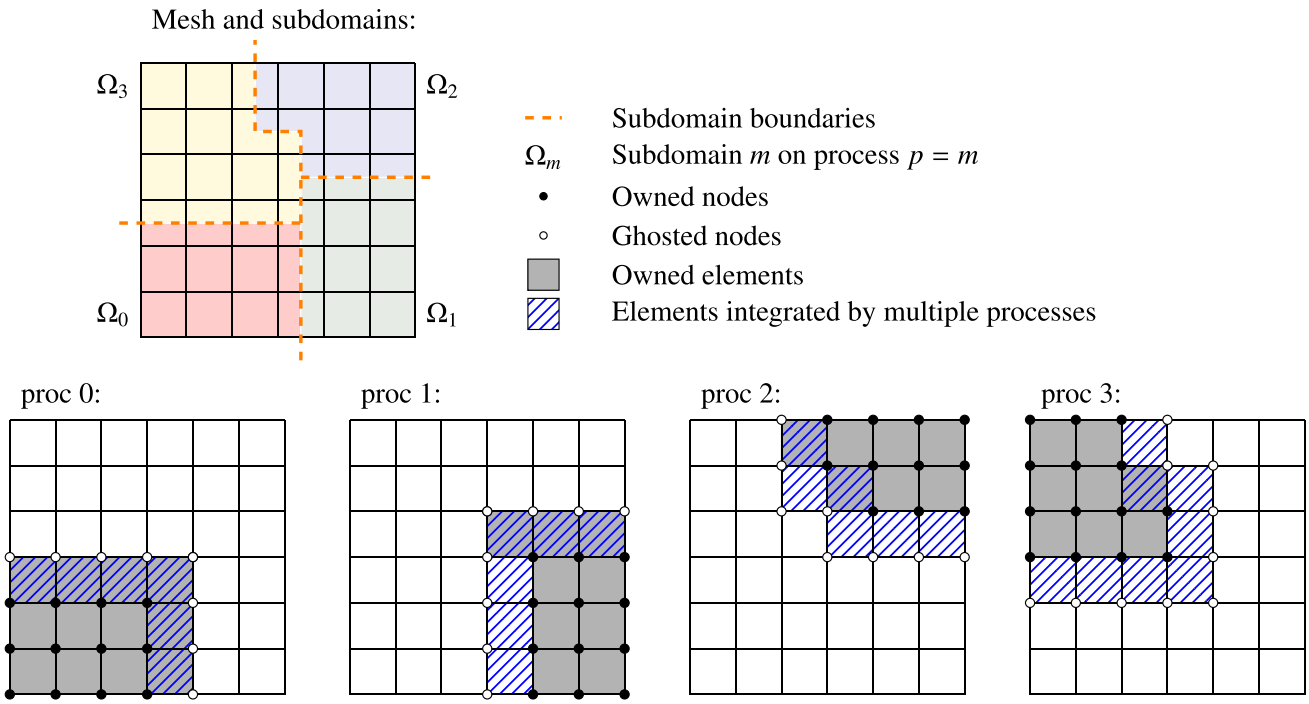


Fig. 2 Exemplary overlapping domain decomposition and parallel assembly involving four subdomains $\Omega_m, m \in \{0, 1, 2, 3\}$ assigned to four parallel processes $p \in \{0, 1, 2, 3\}$. Since each process can only assemble into unknowns of owned nodes, elements spanning across the subdomain boundaries need to be evaluated by multiple processes. This requires ghosting of nodes and elements, which entails parallel communication among multiple processes

Having today's parallel computing architectures with distributed memory in mind, the evaluation of (3) brings along two major implications on the software and algorithm design:

1. The evaluation of the integrands in (3) requires information from both the slave and master side. Slave data are readily available locally on each parallel process. The implementation has to enable access also to master side data, that might be owned by another process or is stored on a different compute node.
2. The computational cost and time is mostly associated with numerical integration over the slave side of the interface. Parallelization can reduce the computational time by distributing the integration domain, i.e. the slave interface, over multiple parallel processes.

Therefore, we deduce the following requirements:

R1: Enable access to all required slave and master data during evaluation of mortar integrals while keeping the memory demand and parallel communication low.

R2: Use parallel resources efficiently for numerical integration over the slave side of the mortar interface, also targeting parallel scalability.

We will elaborate on these implications in Sects. 3 and 4 and outline various approaches to satisfy both requirements **R1** and **R2** in the context of parallel computing.

3 Storing data of the contact interface on a parallel machine

When executing the FEM solver on a parallel machine, data need to be distributed among the different MPI ranks or compute nodes. Now, we first summarize the basics of overlapping domain decomposition to distribute chunks of the discretization to individual processes. Then, we discuss the implications on access to the relevant interface data during contact evaluation, before we present and discuss several strategies to ensure access to the necessary data without excessive data redundancy. Overall, this section is devoted to strategies to satisfy our basic requirement **R1**.

3.1 Overlapping domain decomposition

We base our considerations on the existence of an FEM solver that can be executed on parallel computers with a multitude of CPUs and/or compute nodes using a distributed memory architecture. In our case, this FEM solver is our in-house code Baci [3]. For optimal parallel treatment, the

code base utilizes *overlapping domain decomposition (DD)* techniques [22, 64, 67, 69]. Using n^{proc} to denote the number of available parallel processes, the computational domain Ω is divided into n^{proc} subdomains Ω_m, $m \in \{0, 1, \dots, M-1\}$. A one-to-one mapping of subdomains to processes is employed, such that $n^{\text{proc}} = M$.

An exemplary overlapping DD into four subdomains distributed to processes $p \in \{0, 1, 2, 3\}$ is shown in Fig. 2. While each node in the finite element discretization is uniquely assigned to a subdomain Ω_m, elements might span subdomain boundaries. We stress that processes can only access data of nodes that they own themselves. This has implications on finite element evaluation and assembly: A process p can only assemble into those entries of the global residual vector and those rows of the global Jacobian matrix that are associated with nodes in Ω_p. Hence, elements that span across subdomain boundaries will be evaluated by all processes that own at least one of this element's nodes such that each process can assemble quantities associated with its own nodes.[1] This requires communication of data prior to the evaluation, i.e. data of off-process nodes need to be communicated. This is often referred to as *ghosting*. Ideally, subdomains exhibit a small surface-to-volume ratio to minimize the amount of data subject to ghosting.

In our code base Baci, we employ the hypergraph partitioning package Zoltan [11] with the ParMETIS backend to decompose the computational domain Ω into n^{proc} subdomains Ω_m. Parallel data structures and parallel linear algebra are enabled through the Trilinos[2] packages Epetra, Tpetra, and Xpetra. Iterative solvers for sparse systems of linear equations are taken form the Trilinos packages AztecOO [38] and Belos [4] with scalable multi-level preconditioners from ML [33] and MueLu [9].

3.2 Implications of distributed memory on the contact search and evaluation

Without loss of generality and for ease of presentation, we assume that the entire discretization of a two-body contact problem has undergone an overlapping DD and that each subdomain $m \in \{0, \dots, M-1\}$ has been assigned to a process $p \in \{0, \dots, n^{\text{proc}} - 1\}$. For the purpose of illustration, we will discuss the case of $n^{\text{proc}} = 3$ subdomains and further assume that every process owns a part of the master and of the slave interface as illustrated in Fig. 3. Please note that our considerations also hold, if some processes only own a part of either the slave or the master side of the interface

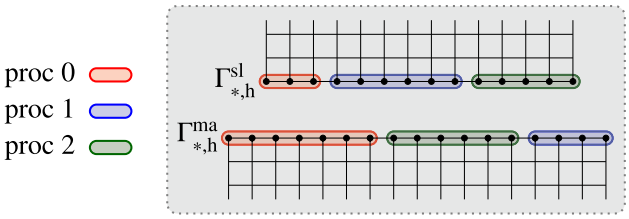


Fig. 3 Without particular measures, DDs of master and slave side of the interface distribute each interface side to some processes. Geometrically close portions of the master and slave interface are not guaranteed to reside on the same process. Without further measures, each process p can only identify possibly contacting pairs of slave/master elements from the subset of master elements owned by process p, i.e. master elements that reside in the set $\Omega_p \cap \Gamma^{\text{ma}}_{*}$. This can and needs to be alleviated by extending the ghosting of the master side of the interface. (For simplicity of visualization, coloring of ownership omits ghosted elements stemming from the overlapping interface DD)

discretization or even if some processes do not own any portion of the contact interface at all.

When process p is performing contact search and evaluation on its share of the slave interface, it needs access to data from the geometrically close master side of the interface. In a parallel computing environment, the required data from the master side of the interface does not necessarily reside on that same process p. Still, access has to be enabled to

- identify pairs of slave/master elements, that potentially are in active contact. This step is usually referred to as "contact search".
- evaluate the second integrand in (3), where the shape functions $N^{(2)}_{\ell}$ defined on the master side need to be evaluated and projected onto the slave side.

If the required data of the master side resides on a different parallel process q than the current slave-sided process p, these data have to be communicated or "ghosted" (cf. Fig. 2) from process q to process p to be known by process p. Therefore, the ghosting of the master interface discretization has to be extended. Since such an extension will impact the inter-processes communication demand as well as the on-process memory demand, we will introduce models for communication and memory demands in Sect. 3.3. More importantly, we will discuss various approaches for extending the ghosting of the master interface discretization in Sects. 3.4 and 3.5, where we will also discuss the impact of these ghosting extension strategies on the memory demand.

[1] As an alternative, linear algebra data structures, that are specialized for FEM computations, are available. They allow to assemble into off-process rows. Naturally, communication among parallel processes is required.

[2] https://trilinos.github.io.

3.3 Models for communication and memory demand

Starting from an overlapping DD and distributed storage of both interface discretizations, data need to be communicated among processes to facilitate the mortar evaluation. We will use σ to denote the amount of data to be sent over the interconnect of all compute nodes and processes. Since data related to the slave side of the interface discretization just remain on its process p, $\sigma_p^{\mathrm{sl}} = 0$. As has already been indicated in Fig. 3, process p owning the portion Γ_m^{sl} of the slave side of the mortar interface requires the master side's data from those processes owning the geometrically close master elements. Hence, usually $\sigma_p^{\mathrm{ma}} > 0$, especially if a situation as depicted in Fig. 3 occurs. Although an explicit expression to compute σ_p^{ma} cannot be given, as it highly depends on the software implementation at hand, it for sure is related to the number of nodes n^{nd} and elements n^{el} to be communicated. We denote this relation by

$$\sigma_p^{\mathrm{ma}} \propto \zeta(n^{\mathrm{nd}}, n^{\mathrm{el}}) \tag{4}$$

with $\zeta(n^{\mathrm{nd}}, n^{\mathrm{el}})$ referring to an implementation-specific measure describing the cost of parallel communication. The total amount of data to be communicated to process p sums up to

$$\sigma = \sum_{p}^{n^{\mathrm{proc}}-1} \sigma_p^{\mathrm{ma}}. \tag{5}$$

Obviously, σ increases with an increasing number of subdomains. More importantly, however, it is impacted by the individual contributions σ_p^{ma}. Especially when the number of subdomains, that are required to solve a given problem, is fixed, reducing σ_p^{ma} is key to reduce the overall cost of communication. Naturally, $\sigma = 0$ if $n^{\mathrm{proc}} = 1$.

From the domain decomposition of the underlying bulk field, the memory demand s_p^{Ω} per process p is given. For the mortar interface discretizations, we use s_p^{sl} to denote the memory demand of the slave interface portion Γ_m^{sl} on process p. Furthermore, s_p^{ma} refers to the memory demand of the master interface portion Γ_m^{ma} on process p. Then, the total memory demand s_p on process p is given as

$$s_p = s_p^{\Omega} + s_p^{\mathrm{sl}} + s_p^{\mathrm{ma}} \quad \forall p \in \{0, \dots, n^{\mathrm{proc}} - 1\}. \tag{6}$$

Note that s_p includes the amount of memory required for owned nodes/elements as well as for ghost nodes/elements originating from the overlapping DD with an element overlap of 1. We stress that s_p^{Ω} is fully determined by the overlapping DD of the underlying bulk fields and that s_p^{sl} is only governed by the overlapping DD of the slave interface discretization, that might arise from any of the schemes proposed in Sect. 4 later. At this point, only the master interface's contribution s_p^{ma} can be controlled by choosing a specific ghosting extension strategy.

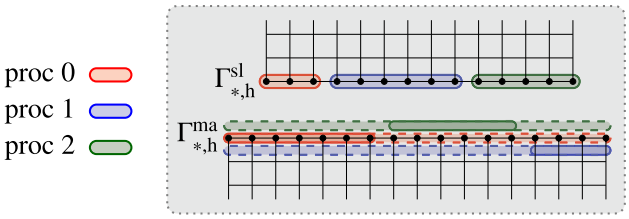


Fig. 4 Fully redundant storage of the master interface discretization: Solid lines indicate data that are owned by a particular process due to the initial DD. Dashed lines indicate data that are available through the extended ghosting. With fully redundant storage of the master discretization on each process, each process p can immediately identify all pairs of slave/master elements, that are possibly in active contact

3.4 Redundant storage: the straightforward case

The probably most straightforward remedy for the issue of undetected master/slave pairs described in Fig. 3 is to fully extend the master side's ghosting to all processes, i.e. to store the entire master side of the interface redundantly on every process p. This scenario of distributed storage of the slave interface discretization, but redundant storage of the master interface discretization is illustrated in Fig. 4 for an exemplary number of three processes. The slave interface Γ_*^{sl} is decomposed into three subdomains and distributed to the processes 'proc 0', 'proc 1', and 'proc 2', indicated by coloring. The master interface Γ_*^{ma} starts out from its initial DD (colored boxes with solid lines) as already seen in Fig. 3. Then, its ghosting is extended over the entire master interface Γ_*^{ma} (colored boxes with dashed lines), such that Γ_*^{ma} is now stored redundantly on all three processes.

The redundant storage of the master side of the interface just requires a one-time setup and communication cost at the beginning of the simulation to extend the ghosting of master data to the entire master interface, but then enables access to every bit of master interface data from every process $p \in \{0, 1, \dots, n^{\mathrm{proc}} - 1\}$ without further communication among parallel processes. After the ghosting has been extended following the idea of fully redundant storage, all algorithmic steps, e.g. the contact search or the evaluation of (3), can be performed immediately without further communication.

In terms of the communication cost σ, however, this approach is rather expensive: since the entire master discretization needs to be communicated to every slave processor p, the total communication cost can be estimated via (4) and (5) as

$$\sigma \approx n^{\text{proc}} \zeta(n^{(2)}, n^{\text{el,ma}}), \tag{7}$$

where *all* nodes and elements of the master side of the interface discretization enter the cost estimate. The model (7) suffers only from a slight over-estimation, since a part of the master surface might already be located on the target process and, thus, does not need to be communicated. Yet, this over-estimation becomes smaller for an increasing number of subdomains.

Since the entire master discretization has to be stored on each process along with a portion of the slave discretization, the memory demand of this approach can grow quite excessively when going to large master interface discretizations. The maximum problem size, for which this strategy still works, cannot be given theoretically. It strongly depends on several key factors, for example the exact specifications of the computing hardware or intricate details of the software implementation. Considering the memory model (6), the per-process master contribution s_p^{ma} has to be replaced by the memory consumption s^{ma} of the entire master interface since each process stores the entire master discretization. Since s^{ma} grows with mesh refinement, the total storage demand s_p on process p is not bounded. This limits the applicability of redundant storage to small and medium sized interface discretizations, depending on the hardware at hand.

Besides the possibly unbounded memory demand, fully redundant ghosting of the master side also comes with a run-time cost: when process p loops over all of its nodes/elements of the master discretization, then it actually loops over *all* nodes/elements of the entire master discretization, although most of the nodes/elements are irrelevant on process p as they are not located in the geometric vicinity of process p's slave nodes/elements. Naturally, the code is not aware of any concept of vicinity prior to the contact search, so this cost cannot be avoided with this approach.

3.5 Distributed storage: going to large problems

As soon as the memory demand s_p exceeds the available memory on a computing node, redundant storage as described in Sect. 3.4 should not be applied anymore to avoid performance degradation due to memory swapping. Following (6), the total storage demand s_p per process can be reduced by reducing the storage demand of the master interface. In particular, when storing also the master interface discretization in a distributed fashion, its storage demand per process can be reduced to $s_p^{\text{ma}} < s^{\text{ma}}$ for $p \in \{0, \dots, n^{\text{proc}} - 1\}, n^{\text{proc}} \geq 2$. Similarly, when the growth in run-time for loops over master nodes/elements becomes prohibitive, reducing the portion of the master interface stored on each process p is expected to speed up simulations. Still, each portion of the slave interface needs to have access to those parts of the master interface that reside in its geometric proximity (cf. Fig. 3). In turn, measuring geometric proximity requires access to all pairs of slave and master nodes.

This situation can be remedied by different algorithmic modifications: Within a token-based evaluation strategy, e.g. inspired by Round-Robin (RR) scheduling [13], the parallel decomposition and distribution of the slave interface is fixed. On the master side, just the decomposition into subdomains is fixed, while the subdomain-to-process mapping is shifted by one process per RR iteration until every process has owned each master interface subdomain once. Since an RR loop requires n^{proc} iterations for a complete evaluation of all slave elements, its run-time cost is high and has even proven to be prohibitive in large-scale applications, which we have also observed in our own experiments.

As an alternative, the incorporation of the notion of *proximity* already into the extension of the master side's ghosting offers a promising solution. Hence, we resort to pre-computing ghosting data based on a *geometrically motivated binning approach*, where we exploit the fact that the contact search needs to identify all master elements in the proximity of a given slave element. This idea is inspired by [55], where a similar parallel algorithm is used for the spatial decomposition of atoms in short-range molecular dynamics simulations.

In the context of mortar methods, we will first construct an axis aligned bounding box around the mortar interface, i.e. a cuboid box that is oriented along the Cartesian axes and encloses all nodes of the mortar interface. Then, this bounding box will be covered with a set of Cartesian bins $\{\mathfrak{B}\}$ that are independent of the finite element meshes of the contacting bodies (cf. Fig. 5). Since the contacting bodies are moving relative to the background bins, slave nodes or elements can migrate between individual bins over time. To not loose track of individual nodes or elements due to this motion, the minimal bin size β_{min} is chosen as

$$\beta_{\text{min}} = \max_{n^{\text{el,sl}}} h^{\text{sl}} + 2 \cdot \Delta t \cdot \overline{\dot{\mathbf{u}}}_*$$

with $\max_{n^{\text{el,sl}}} h^{\text{sl}}$ being the largest element edge of the slave discretization, Δt representing the time step size, $\dot{\mathbf{u}}_*$ denoting the vector of nodal interface velocities and $\overline{(\bullet)}$ referring to the mean value of $(\bullet)$, respectively. If the interface velocity is not available in static problems, it can be replaced via a finite difference approximation w.r.t. to the previous load step. Analogously, the axis aligned bounding box embracing all mortar nodes is expanded by β_{min} in each direction. Then, the actual bin size β and number of bins per direction is computed based on the dimensions of the expanded axis aligned bounding box and the minimal bin size β_{min}. We then

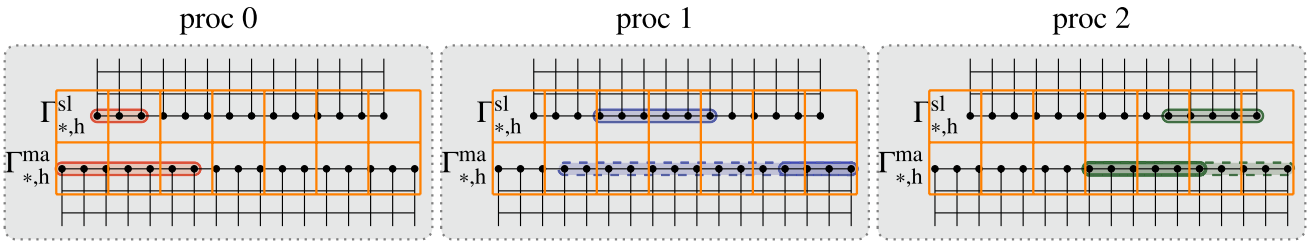


Fig. 5 Extended ghosting of the master interface using a binning scheme—We exemplarily show three parallel processes and depict each one in its own sketch for the sake of presentation. Bins $\{\mathfrak{B}\}$ are sketched in solid orange lines. On Γ^{sl}_{*}, mesh entities (such as nodes and elements) are owned by the respective process anyway. On Γ^{ma}_{*}, solid lines indicate data that are owned by this process, while dashed lines indicate data that has been ghosted via binning

apply Algorithm 2 to compute process-specific lists $\{e_{gh}\}^{ma}_{p}$ of master elements to be ghosted for each process p. Please note that potentially a subset of $\{e_{gh}\}^{ma}_{p}$ already resides on processor p and, thus, does not need to be communicated.

Algorithm 2: Geometrically motivated binning to pre-compute ghosting of the master interface

```
// Setup and preliminary steps
Construct an axis-aligned bounding box covering both interface discretizations Γ^sl_{*,h} and Γ^ma_{*,h}
Subdivide the bounding box into cartesian bins {𝔅}
Sort all n^{el,sl} slave elements {e^sl} into bins {𝔅} to obtain {𝔅^sl}
Sort all n^{el,ma} master elements {e^ma} into bins {𝔅} to obtain {𝔅^ma}

// Compute ghosting informations
for each process p do
    // Find list of bins {𝔅^sl_m} enclosing p's slave subdomain Γ^sl_m
    for each slave element e^{(ι),sl} in proc p's subdomain Γ^sl_m do
        𝔅_{e^{(ι),sl}} ← {𝔅^sl} ∩ e^{(ι),sl}        // get bin 𝔅_{e^{(ι),sl}} ∈ {𝔅^sl} containing current slave element e^{(ι),sl}
        {𝔅^sl_m} ← {𝔅^sl_m} ∪ 𝔅_{e^{(ι),sl}}        // update list of bins {𝔅^sl_m} enclosing p's slave subdomain Γ^sl_m
    end

    // Find set of bins {𝔅^nb_b} neighboring the current bin 𝔅_b
    for each bin 𝔅_b in {𝔅^sl_m} do
        {𝔅^nb_b} = 𝔅_b                              // initialize list of neighboring bins
        for each bin 𝔅_d in {𝔅} do
            if 𝔅_b ∩ 𝔅_d ≠ ∅                        // Are both bins direct neighbors?
             then
                {𝔅^nb_b} ← {𝔅^nb_b} ∪ 𝔅_d
            end
        end
    end

    // Compute list {e_gh}^ma_p containing all master elements in {𝔅^nb_b}
    for each master element e^{(τ),ma} do
        if e^{(τ),ma} ∩ {𝔅^nb_b} ≠ ∅                // Is master element e^{(τ),ma} inside set of bins {𝔅^nb_b}
         then
            {e_gh}^ma_p ← {e_gh}^ma_p ∪ e^{(τ),ma}
        end
    end
end
```

Figure 5 illustrates the binning approach detailed in Algorithm 2 for three processes. For 'proc 0', no further ghosting is required in this example, since all required master elements $\{e_{\text{gh}}\}_0^{\text{ma}}$ already reside in the neighboring bins of the set of bins $\{\mathfrak{B}_0^{\text{sl}}\}$ enclosing all slave elements of $\Gamma_{*,0}^{\text{sl}}$. In contrast, the master elements owned by 'proc 1' are not contained in $\{\mathfrak{B}_1^{\text{sl}}\}$ and do not participate to the evaluation of mortar terms in $\Gamma_{*,1}^{\text{sl}}$. The master elements of interest, i.e. $\{e_{\text{gh}}\}_1^{\text{ma}}$ in the neighboring bins of $\{\mathfrak{B}_1^{\text{sl}}\}$, need to be ghosted, which leaves out the master elements in the left most bin covering Γ_*^{ma}. Finally, 'proc 2' requires $\{e_{\text{gh}}\}_2^{\text{ma}}$, while it already owns some of the required elements and only needs to ghost some additional elements.

The communication cost σ_p^{ma} for each processor p now depends on the number of nodes/elements in the current bin b and its neighboring bins. Due to the Cartesian character of bins, each bin has 8 or 26 neighbors in 2D or 3D, respectively. Based on a constant bin size and assuming uniform mesh sizes, the cost measure ζ per subdomain introduced in (4) is now evaluated with $8 \times n_p^{(2)}$ or $26 \times n_p^{(2)}$ nodes and $8 \times n_p^{\text{el,ma}}$ or $26 \times n_p^{\text{el,ma}}$ elements for 2D and 3D problems, respectively. With an increasing number of subdomains and under the assumption of uniform meshes, the total cost for communication is then bounded by

$$\sigma \leq \begin{cases} 8 \cdot \sigma_p^{\text{ma}} & \text{for 2D} \\ 26 \cdot \sigma_p^{\text{ma}} & \text{for 3D} \end{cases} \tag{8}$$

which is a significant reduction for large core counts compared to (7). The scalar factors in (8) originate from the number of neighboring bins in 2D and 3D, respectively.

Regarding memory demand as estimated via (6), the master side's demand s_p^{ma} now comprises of all master elements stored on process p plus all master elements in neighboring bins. Assuming bin sizes similar to the size of subdomains Ω_m as well as evenly sized master elements, the master side's storage demand is bounded by $5 \times s_p^{\text{ma}}$ or $9 \times s_p^{\text{ma}}$ for 2D and 3D problems, respectively. While the number of bins and, thus, the effort to sort master elements into bins increases with a smaller characteristic bin size β, the storage demands for each process p diminishes even more.

3.6 Intermediate discussion of ghosting strategies

So far, we have concerned ourselves with strategies to satisfy the requirement **R1**. Before addressing **R2** in Sect. 4, we briefly discuss some properties of the presented strategies for the ghosting of the master interface.

While the fully redundant ghosting presented in Sect. 3.4 appears as straightforward, easy to implement, and only needs to be done once at the beginning of the simulation, its runtime cost for communication as well as its memory demand can become prohibitive when going to large problems. The RR approach, in turn, alleviates the issue of excessive growth of memory demand. Yet, the number of necessary RR iterations equals the number of processes n^{proc}, rendering this approach impractical for $n^{\text{proc}} \gg 1$ (especially as it has to be applied in every time/load step). Although the binning approach proposed in Sect. 3.5 needs to be applied in every time/load step, it appears as the only approach without impractical restrictions when going to large problem sizes: Through the choice of the number and size of the bins, the amount of data to be ghosted can be controlled, such that only those master elements will be ghosted, that are likely to be required during contact search and evaluation. In sum, the applicability of the binning approach is neither affected by the number of parallel processes nor greatly impacts the parallel communication or total memory demand.

We will later supplement our assessment with detailed numerical experiments in Sect. 5.1.1, but want to anticipate the main finding here: For the largest problems with $25M$ mesh nodes and 25k interface nodes, the process with the largest ghosting demand asks for the redundant ghosting of 25,921 nodes, while binning reduces this number to 1212 nodes, which amounts to a reduction of more than 20×. On average across all MPI ranks, these numbers can be improved through load balancing which will be introduced in Sect. 4.

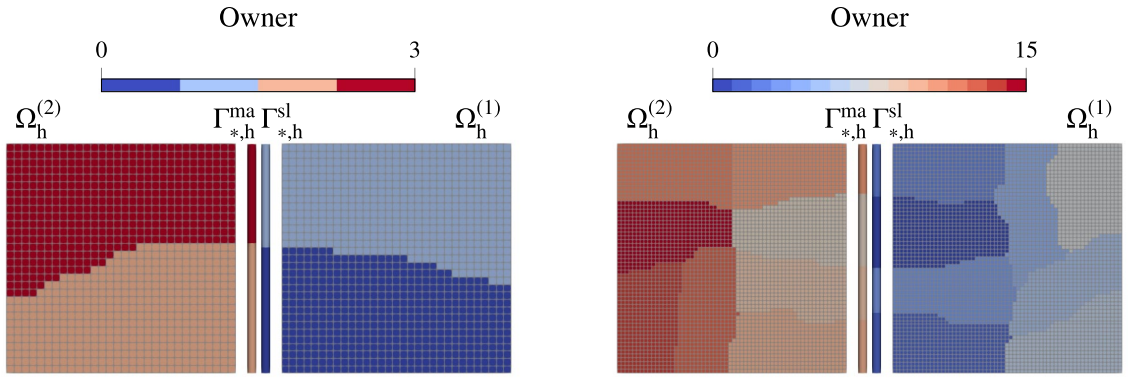
4 Balancing the work load among multiple parallel processes

Now, we discuss strategies for an optimal distribution of the work load to multiple parallel processes. These strategies are intended to satisfy the requirement **R2** from Sect. 2.3. We assume that requirement **R1** has already been satisfied by any of the methods described in Sect. 3 and, thus, all data are accessible whenever needed.

In Sects. 4.1–4.3, we first present some general considerations applicable to all type of mortar interface problems, before we move to the specific scenario of dynamically evolving contact problems in Sect. 4.4.

4.1 The concepts of strong and weak scalability

When assessing the performance of a parallel code and/or algorithm, an important question is whether adding more computational resources will actually speed-up the algorithm's performance at the proper rate. Two concepts are commonly followed and investigated:



(a) $10/11 \times 10/11$ mesh using 4 processes

(b) $20/21 \times 20/21$ mesh using 16 processes

Fig. 6 Perfect alignment of bulk and interface domain decomposition: subdomains in the master side of the interface (left vertical strip) coincide with the master side's bulk discretization (left square), while the slave side's interface subdomains (right vertical strip) are aligned with the slave side's bulk discretization (right square)

- For a fixed problem size, *strong scalability* is given, if the computational time diminishes at the same rate as the used hardware resources grow. The *strong scaling limit* is reached, when increasing the hardware resources does not lead to a further reduction of computational time. See [2].
- *Weak scalability* expects a constant computational time when increasing the problem size and the parallel resources at the same rate, i.e. when the work load per process is kept constant. See [35].

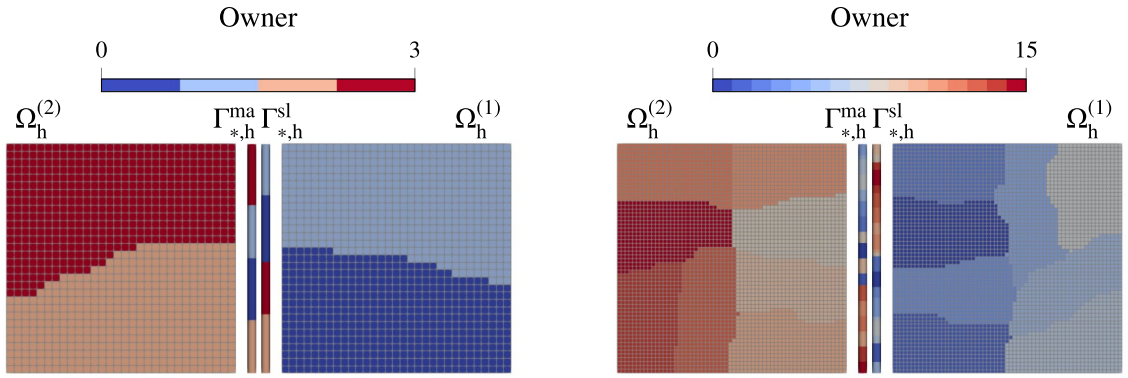
As it is well established in many research and application codes (and also in our code base Baci [3]), weak scalability of the finite element evaluation of the pure bulk field (i.e. volume element evaluation) without the presence of any mortar interface can be achieved under uniform mesh refinement.

4.2 Curse of dimensionality

In surface-coupled problems with d spatial dimensions, the coupling surface is always a $d-1$ dimensional geometric entity. Originally described in [7], this *curse of dimensionality* between the bulk and the interface discretization becomes problematic under uniform mesh refinement. Denoting the characteristic mesh size with h, the number of unknowns in the bulk discretization grows at $\mathcal{O}\left(h^d\right)$ while the surface discretization of the coupling interface exhibits a growth rate of $\mathcal{O}\left(h^{d-1}\right)$ only.

This becomes evident in practice when a first and simple DD of the interface discretizations is now obtained by aligning the interface subdomains of the slave and master side with the subdomains of the underlying bulk discretizations. Although this approach is straightforward to implement and also avoids off-process assembly, thus reducing parallel communication, it does not result in an optimal parallel distribution for the evaluation of the mortar coupling terms. Since computing the interface contributions, i.e. the mortar segmentation process, integration and assembly of the mortar matrices $\mathcal{D}$ and $\mathcal{M}$ to only name the most important tasks, is all done on the slave interface discretization, all numerical tasks might be performed by very few parallel processes only, while others idle.

For simplicity of visualization, this is illustrated using a two-dimensional meshtying problem in Fig. 6, where the domain decomposition of the mortar interface's slave and master side is fully aligned with the underlying bulk discretizations. Considering a coarse discretization distributed to four parallel processes as shown in Fig. 6a, the slave interface is divided into two subdomains and the master interface is owned by two processes only as well. Consequently, there are two processes, that do not own a share of the slave interface, and two other processes not owning any node of the master interface. In Fig. 6b, the mesh has been refined by a factor of two in each direction, and 16 processes have been used such that the load per process remains constant in the bulk discretization. While the bulk discretization is now split into 16 subdomains, the slave and master interface are shared only among four processes each. In sum, only 4 processes tackle the expensive evaluation of mortar terms on the slave side of the interface, while 12 processes are completely left out. Even in these small and only two-dimensional problem, it becomes evident that the alignment of interface subdomains with bulk subdomains potentially leaves a huge fraction of all processes idle during interface evaluation. While it is true that using more processes improves the parallelization of the bulk discretization, it does not necessarily contribute to a good and scalable parallelization of the interface computations. We stress that this



(a) $10/11 \times 10/11$ mesh on 4 processes

(b) $20/21 \times 20/21$ mesh on 16 processes

Fig. 7 Independent interface domain decomposition using all n^{proc} parallel processes

issue is even more pronounced for the three-dimensional case and larger numbers of parallel processes.

4.3 Improving the domain decomposition of interface discretizations

To overcome the curse of dimensionality and to satisfy **R2**, we allow the slave and master side of the interface to be decomposed into subdomains independently from the underlying bulk discretizations to achieve optimal parallel scalability of the computational tasks associated with both the integration and assembly in the bulk domains Ω_1 and Ω_2 as well as integration and assembly on the interfaces Γ^{sl} and Γ^{ma}. In a first and straightforward approach, one can divide both interfaces Γ^{sl} and Γ^{ma} into n^{proc} subdomains, such that each parallel process handles a portion of the interface as illustrated in Fig. 7. This is particularly important for the slave side which needs to perform all computations related to the integration of the mortar terms in (3).

For the coarse and the fine mesh, both Γ^{sl} and Γ^{ma} are distributed to 4 and 16 parallel processes, respectively. A clear advantage of this strategy is that all parallel processes participate in the interface treatment, so idling is mostly avoided. However, the fine mesh already indicates that the interface subdomains may become very small, i.e. they consist only of a few elements. Recalling the curse of dimensionality outlined in Sect. 4.2, this will become an issue at large scale where the bulk field is divided into n^{proc} subdomains of reasonable size, while the subdomain size of the interface decreases when refining the mesh and adding parallel processes at the same rate. Having many but very small interface subdomains does not leave any process idle, but also yields interface subdomains with a large surface-to-volume ratio which indicates an increasing communication overhead. In sum, this strategy distributes the computational work of the interface evaluation more evenly to all processes than just adopting the interface subdomains from the underlying bulk discretization. The numerical experiments in Sect. 5 confirm this statement.

Conceptually, there is still room for further optimizations, in particular related to parallel communication among processes, e.g. by setting a lower bound $n_{\text{min}}^{\text{el}}$ on the number of elements per interface subdomain to reduce the communication overhead. Such an approach needs to compromise between the amount of parallel communication among processes and the number of idling processes. In this work, we have refrained from exploring this research direction, since the distribution of the interface to all parallel processes already delivers satisfying scaling behavior for many practical applications.

4.4 Interface domain decompositions for dynamically evolving interfaces

In many applications, the interface configuration evolves over time, e.g. as in contact problems with large sliding or contact of rolling bodies. In such cases, the interface DD can come out of balance, resulting in some processes to do significantly more work than others, which possibly idle. Then, a rebalancing can become necessary to distribute the computational work evenly to all participating processes.

Algorithm 3: Dynamic load balancing of the interface discretization

```
// Initialize imbalance measures
η_t ← 0.0
η_e ← max_p(n_p^{el,sl}) / min_p(n_p^{el,sl})
for all time steps do
    if η_t ≥ η̂_t ∨ η_e ≥ η̂_e then
        Compute more well-balanced DDs of interfaces Γ^{sl}_{*,h} and Γ^{ma}_{*,h}
        Redistribute interface data to new interface DDs
    end
    while nonlinear solver not converged do
        Evaluate weak form of bulk field Ω^{(i)}, i ∈ {1, 2} and assemble residual and its linearization
        Start time measurement on each process p ∈ {0, ..., n^proc − 1}
        Evaluate weak form of interface Γ_* and assemble residual and its linearization
        Stop time measurement to obtain t_{eval,p}, p ∈ {0, ..., n^proc − 1}
        // Solver linear system and update solution
        ⋮
    end
    // Update imbalance measures using (9)
    η_t ← max_p(t_{eval,p}) / min_p(t_{eval,p})
    η_e ← max_p(n_p^{el,sl}) / min_p(n_p^{el,sl})
end
```

Algorithm 3 details the integration of imbalance monitoring and potential rebalancing steps into the time integration and nonlinear solver routines. In each time step, each processor p tracks the time $t_{\text{eval},p}$ spent in the evaluation of all mortar terms as well as the number of its slave elements $n_p^{\text{el,sl}}$. We then estimate the imbalance among all processes by

$$\eta_t = \frac{\max_p \left(t_{\text{eval},p}\right)}{\min_p \left(t_{\text{eval},p}\right)}, \quad \eta_e = \frac{\max_p \left(n_p^{\text{el,sl}}\right)}{\min_p \left(n_p^{\text{el,sl}}\right)} \tag{9}$$

with η_t and η_e denoting the imbalance in contact evaluation time and number of slave elements per processor, respectively. The theoretical optimum of a perfect balancing of the mortar-related workload is given for $\eta_t = 1$ and $\eta_e = 1$, respectively, i.e. when all processes spend exactly the same time in mortar evaluation and when all processes own the exact same number of slave elements. If in any time step these imbalance estimates exceed user-given thresholds $\hat{\eta}_t \geq 1$ and $\eta_e \geq 1$ for contact evaluation time and number of slave elements per processor, respectively, i.e. if

$$\eta_t \geq \hat{\eta}_t \quad \vee \quad \eta_e \geq \hat{\eta}_e, \tag{10}$$

then we re-compute the interface DD to obtain a DD with better load balancing. As this load balancing procedure is triggered dynamically by the current state of the simulation, we refer to it as *dynamic load balancing*.

Naturally, $\hat{\eta}_t = 1$ will trigger rebalancing in every time step, such that each time step can rely on the best possible interface DD. In practice, the cost for rebalancing needs to be taken into account, such that practical computations require $\hat{\eta}_t > 1$. We will study the impact of the actual choice of $\hat{\eta}_t$ on the run time in Sect. 5.2.

The main difference between the two imbalance measures η_t and η_e is that η_e does not account for the time to evaluate a given master/slave pair, while η_t relies on actual wall clock timings. Thus, situations with $\eta_e \gg 1$, but η_t fairly close to 1 can occur, if the contact search identifies a huge number of pairs of master and slave elements as close to each other, but the subsequent mortar evaluation cannot find a valid projection and, thus, most of the computational work to evaluate (3) is skipped for such pairs of elements. In sum, the time-based trigger η_t is expected to be more effective to avoid idling processes in practical simulations.

4.5 Implication on finite element assembly and communication patterns

Although the slave side's interface discretization might exhibit its independent DD to improve scalability, all system quantities, e.g. the Jacobian matrix $\mathbf{J}$ and the residual vector $\mathbf{f}$, are distributed among parallel processes following the DD of the underlying bulk discretizations. After evaluation of the mortar element matrices defined in (3) within a mortar element in interface subdomain Γ_n, $n \in \{0, \dots, M-1\}$, on process $q \in \{0, \dots, n^{\text{proc}} - 1\}$, a contribution to $\mathbf{J}$ and $\mathbf{f}$ associated with node j in Ω_m, $m \in \{0, \dots, M-1\}$, owned by process $p \in \{0, \dots, n^{\text{proc}} - 1\}$ can only be assembled by process p. Hence, if $p \neq q$, communication is required to send data from process q to process p to assemble into global system quantities. From the perspective of the evaluating process, this is referred to as *off-process assembly*. Communication can only be avoided if and only if $p = q$.

It is true that off-process assembly increases the amount of communication and, thus, puts a cost burden onto the entire algorithm. Although this is not desirable, it is usually the much cheaper price to pay than to just stick to the one-to-one matching of interface and underlying bulk DDs. The speed-up of the cost-intensive evaluation of mortar terms through an independent DD of the interface discretizations easily amortizes the additional cost of communication related to off-process assembly. We will study timings of the mortar evaluation and off-process assembly in detail in the numerical experiments in Sect. 5.

5 Numerical experiments

We first study parallel redistribution and scalability in a simple two-block contact example in Sect. 5.1 before moving on to dynamic contact problems in Sect. 5.2.

All computations are done with our in-house multi-physics research code Baci [3]. All scaling studies have been run on our in-house cluster (20 nodes with 2x Intel Xeon Gold 5118 (Skylake-SP) 12 core CPUs, 196 GB RAM per node, Mellanox Infiniband Interconnect).

5.1 Contact of two cubes

For a first assessment of the scalability of the contact evaluation, we consider a simple two-block contact problem with a small block (dimensions $0.8 \times 0.8 \times 0.8$) and a slightly bigger block (dimensions $1.0 \times 1.0 \times 1.0$), where contact will occur between two flat surfaces of the blocks. To reduce the complexity of the contact problem and to exclude nonlinearities due to changes in the contact active set, the faces opposite to the contact interface are fixed with Dirichlet boundary conditions, while the blocks initially penetrate each other at the contact interface by 0.001. The smaller block acts as the slave side and its entire contact area is already initialized as "active". Application of the contact algorithms will then result in a slight compression of both blocks, such that the initial penetration vanishes. This problem setup allows to distill the computational effort spent on the redistribution, ghosting, and contact evaluation. In fact, for the parallel scaling studies, we only evaluate all contact terms, but then do not even solve the contact problem to allow for an even more concise focus on the scaling behavior of the contact evaluation.

Both blocks use a Neo–Hooke material with Young's modulus $E = 10$ and Poisson's ratio $\nu = 0.3$. Denoting the mesh refinement factor with κ, both blocks are discretized with 5κ linear hexahedral elements along their edges.

As an exemplary visualization, Fig. 8 illustrates the assignment of subdomains to MPI ranks for a simulation with 24 MPI ranks. Since the discretization of both blocks uses the same number of elements per block, the volume DD exhibits 12 subdomains for each block. Without load balancing, the interface DD evidently matches the underlying volume DD (cf. the top right picture in Fig. 8). In particular, the slave side of the interface is shared by only 6 (out of 24) processes, such that the remaining 18 processes idle during the expensive mortar evaluation. While the DD of the solid volume is not affected by the interface load balancing, the interface DD now yields 24 subdomains for both sides of the interface (cf. the bottom right picture in Fig. 8). This allows to share the computational work for the mortar evaluation among *all* processes.

5.1.1 Weak scaling

We perform a weak scaling study. The smallest problem using 1 MPI rank consists of 55,566 displacement unknowns, while 441/400 nodes/elements reside on the slave side of the contact interface. The largest problem using 480 MPI ranks contains 25,039,686 displacement unknowns, with 25,921/25,600 nodes/elements located on the slave side of the contact interface. We target a load of $\approx$ 50k displacement unknowns per MPI rank under weak scaling conditions. Timing results are shown in Fig. 9. With load balancing, the pure contact evaluation time remains constant under weak scaling conditions as shown in Fig. 9a and as expected for finite element evaluations. Manifesting the curse of dimensionality described in Sect. 4.2 though, the case without load balancing does not equally benefit from adding hardware resources since most of the additional processes do not participate in the mortar evaluation. While the choice of load balancing does not impact the serial case ($n^{\text{proc}} = 1$) of course, the contact evaluation without load

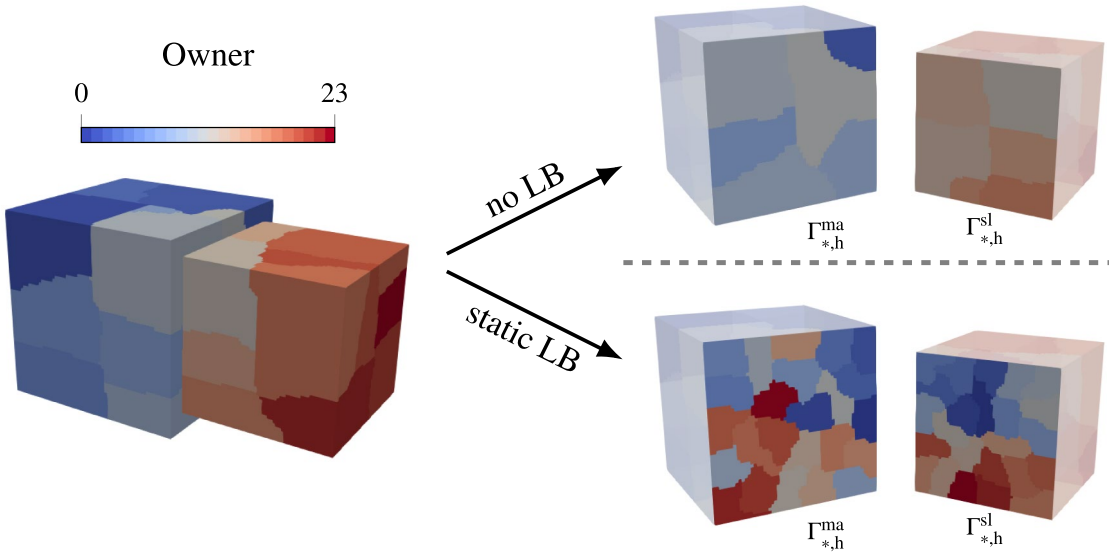


Fig. 8 Two cubes in contact: colors represent the owning MPI rank of a volume/interface subdomain (Color figure online)

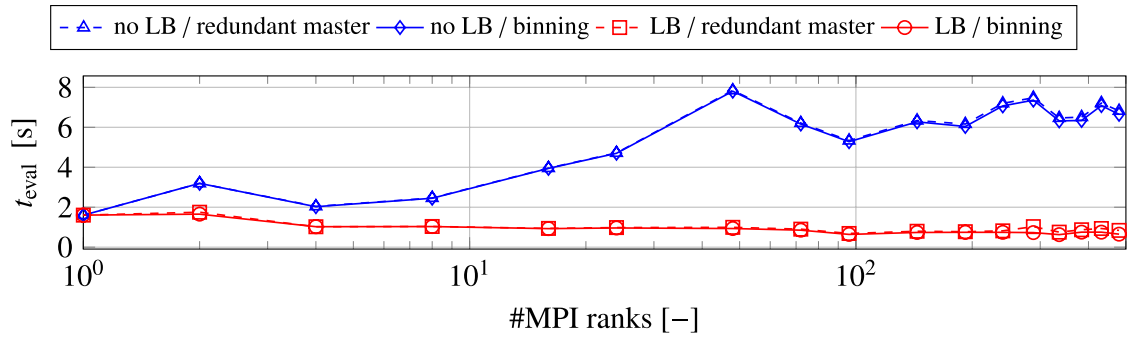
balancing requires twice as much time on 2, 4, and 8 MPI ranks than with load balancing, since the processes owning a piece of the master side of the interface do not contribute to the contact evaluation. For an increasing number of MPI ranks, this gap increases.

Regarding the time spent in redistribution and extending the interface ghosting, $t_{\text{LB}} + t_{\text{gh}}$, an increase with an increasing number of MPI ranks is expected, as the size of the MPI communicator grows and, thus, mandates increased communication. Obviously, this time component is rather independent of the parallel distribution, but is largely impacted by the ghosting strategy: Since the redundant ghosting of the master side requires to communicate all interface nodes and elements of the master side to all MPI ranks, the timings for redundant ghosting exceed the time for ghosting via the geometrically motivated binning approach, where the amount of data to be communicated among processes is reduced based on geometric information.

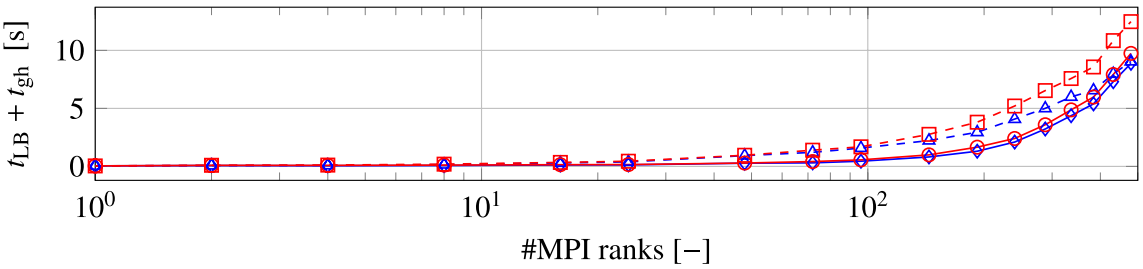
It becomes evident from Fig. 9c, that the time for assembling of all contact terms into the global linear system is only slightly impacted by load balancing, while the impact of the ghosting strategy appears to be negligible.

Finally, we assess the total cost of contact evaluation which is the most relevant target quantity for practical applications. It is given by the total time $t_{\text{total}} = t_{\text{LB}} + t_{\text{gh}} + t_{\text{eval}} + t_{\text{ass}}$ for (possibly) redistributing, ghosting, evaluation, and assembly of the contact interface and is shown in Fig. 9d. Again, ghosting via binning ("binning") results in a lower total time t_{total} than the redundant ghosting of the master side ("redundant master"). Moreover, load balancing ("LB") allows all MPI ranks to participate in the evaluation of the contact terms, yielding a faster total contact time than without load balancing ("no LB"). Dominated by the contact evaluation time t_{eval}, the case without load balancing does not scale beyond 8 MPI ranks, while load balancing shows good weak scalability up to 200 MPI ranks. Overall, our proposed strategy of load balancing in combination with binning delivers the fastest contact evaluation for all mesh sizes and also features the smallest increase in total contact time when increasing the problem size.

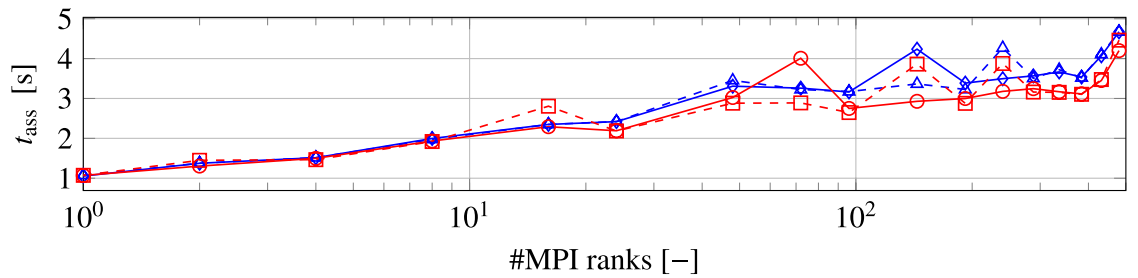
Figure 10 illustrates the impact of both the load balancing and the ghosting strategy on the number of owned and ghosted master side elements by reporting the maximum number of elements per MPI rank among all processes. Naturally, load balancing, where all processes hold a portion of the contact interface, leads to a lower number of owned entities per processor than no load balancing, where the interface is stored only by a subset of all processes: Depicted by the dotted lines, the number of owned elements per MPI rank is smaller in case of load balancing than without load balancing, in particular by a factor of 100 for more than 48 MPI ranks in this example. The influence of the ghosting strategy is shown with dashed and solid lines: While binning (solid lines) just adds a small number of nodes or elements to be ghosted from other processes, fully redundant ghosting (dashed lines) drastically increases the number of ghosted elements. For large examples, this increase can exceed two orders of magnitude. We observe that the number of owned elements is consistently smaller than the number of ghosted elements when using load balancing, while this is not the case without load balancing. This peculiarity is just an artifact of the visualization, since the MPI rank with the maximum number of owned entities is not necessarily the



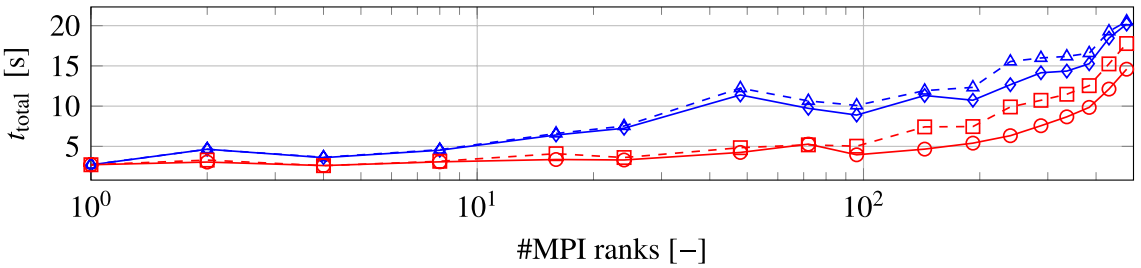
(a) Contact evaluation time



(b) Time for redistribution and ghosting



(c) Time for assembly of contact terms in linear system



(d) Total time for contact evaluation

Fig. 9 Weak scaling of contact time for two cubes

same as the one with the maximum number of ghosted entities. Using the ratio of ghosted elements to owned elements to indicate the additional overhead in memory and parallel communication due to the distributed memory paradigm, we make the following key observation: ghosting via binning as proposed in Sect. 3.5 is much more efficient in terms of memory and parallel communication than fully redundant ghosting. Please note that the respective diagram for owned and ghosted nodes of the interface's master side essentially looks the same and, thus, is not shown for the conciseness of the presentation.

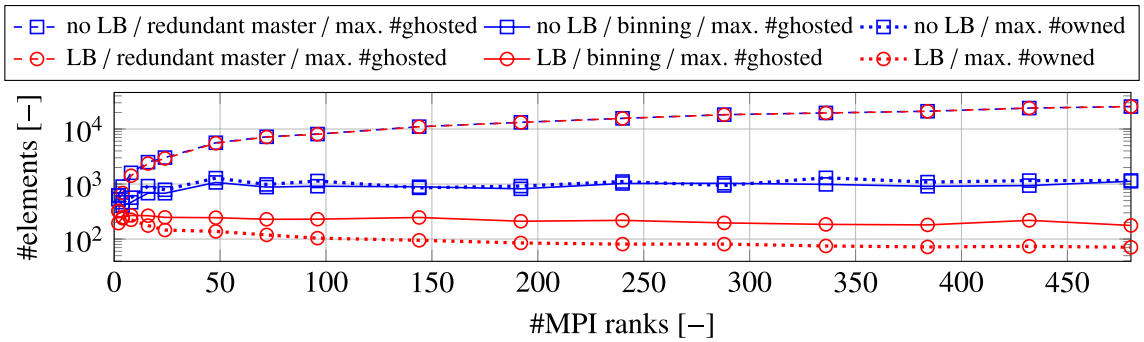


Fig. 10 Number of owned and ghosted elements of the interface's master side under weak scaling conditions

Table 1 Three meshes and problem sizes for strong scaling of a two-block contact example

Mesh ID	Bulk domain		Slave interface	
	Number of nodes	Number of DoFs	Number of nodes	Number of elements
2M	715,822	2,147,466	5041	4900
5M	1,769,472	5,308,416	9216	9025
10M	3,543,122	10,629,366	14,641	14,400

5.1.2 Strong scaling

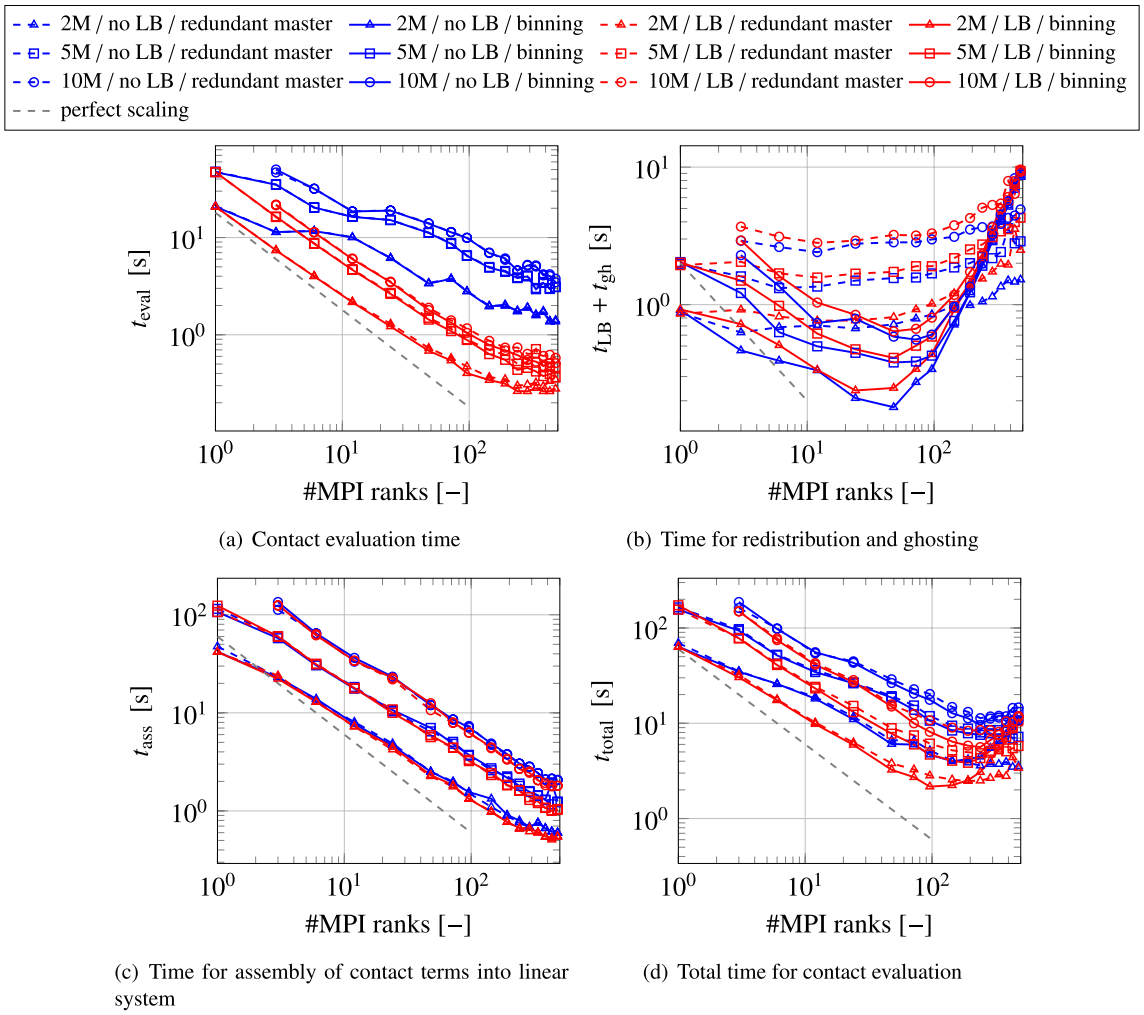
To assess the strong scaling behavior under different load balancing and ghosting strategies, we study three different meshes and problem sizes detailed in Table 1. The strong scaling behavior is reported in Fig. 11. While meshes 2M and 5M could be run in serial, mesh 10M did not fit into the memory of a single core. Hence, the graphs for the mesh 10M start at 3 MPI ranks, while 2M and 5M start at 1 MPI rank.

Regarding the pure contact evaluation time t_{eval} depicted in Fig. 11a, the curse of dimensionality as described in Sect. 4.2 leads to insufficient scaling behavior for the case without load balancing. For some cases, the contact evaluation time does not decrease (or even slightly increase) when adding more processes, (cf. the mesh '2M' without load balancing executed on 3, 6, and 12 MPI ranks for example). In contrast, the proposed load balancing scheme delivers the expected strong scaling behavior across a wide range of MPI ranks, since *all* MPI ranks participate in the contact evaluation. Moreover, load balancing results in faster contact evaluation independent of the mesh size and ghosting strategy than no load balancing. Naturally, strong scaling behavior of the contact evaluation time t_{eval} is not affected by the choice of ghosting strategy.

For the combined time $t_{\text{LB}} + t_{\text{gh}}$ for redistribution and ghosting as shown in Fig. 11b, the timings are now dominated by the choice of ghosting strategy. In particular, fully redundant ghosting of the master interface (dashed lines) requires a consistently larger time across a wide range of MPI ranks. Ghosting via binning (solid lines) can benefit from additional hardware resources, until the strong scaling limit is reached and timings are increasing with an increasing number of MPI ranks. The effect of the load balancing strategy is negligible, but we note that the extra cost of performing a redistribution leads to slightly higher times with load balancing than without load balancing.

Considering the assembly of all contact terms into the global linear system, Fig. 11c shows just a small difference with and without load balancing. Similar to the weak scaling study from Sect. 5.1.1, the ghosting strategy does not impact these timings. We observe good strong scalability for all studied cases.

Having in mind the overall goal of a fast time-to-solution, the total time $t_{\text{total}} = t_{\text{LB}} + t_{\text{gh}} + t_{\text{eval}} + t_{\text{ass}}$ for (possibly) redistributing, ghosting, evaluation, and assembly of the contact interface is depicted in Fig. 11d. Again, the proposed load balancing strategy results in the best timings and in good, but not perfect scaling behavior. Stemming from the pure contact evaluation time t_{eval} (depicted in Fig. 11a), the total time t_{total} without load balancing does not strictly follow the expected strong scaling behavior. Per definition of t_{total}, this diagram combines all characteristics from Fig. 11b–d, namely the better scaling of t_{eval} due to load balancing and the increase in the timing component $t_{\text{LB}} + t_{\text{gh}}$ for large numbers of MPI ranks due to increased communication and redistribution effort.



(a) Contact evaluation time

(b) Time for redistribution and ghosting

(c) Time for assembly of contact terms into linear system

(d) Total time for contact evaluation

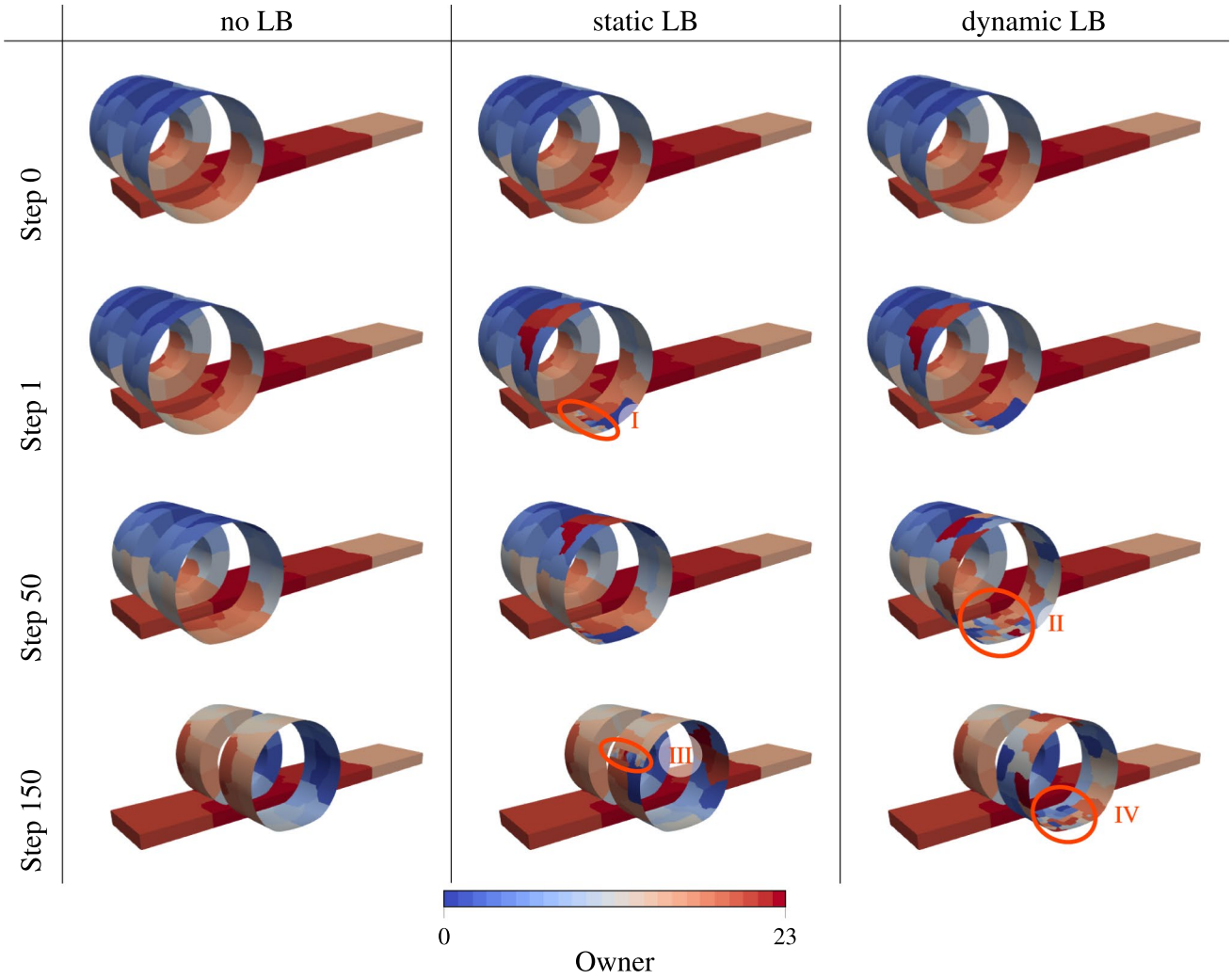
Fig. 11 Two-block contact: strong scaling of contact time

In sum, the best scaling behavior is achieved with the proposed approach of load balancing in combination with ghosting via binning. While both components affect the overall efficiency, the fastest evaluation times and the best weak and strong scaling behavior can only be achieved through the *combination* of load balancing with ghosting via binning. So far, we have limited our analysis to static contact problems without any changes in the contact zone, where the proposed algorithms demonstrate their beneficial effect on the run time and the weak and strong scaling behavior, but could not unfold their full potential. Therefore, we now move to dynamic contact problems, where the contact zone changes over time and, thus, the load balancing is expected to show an even better effect on the scalability and performance.

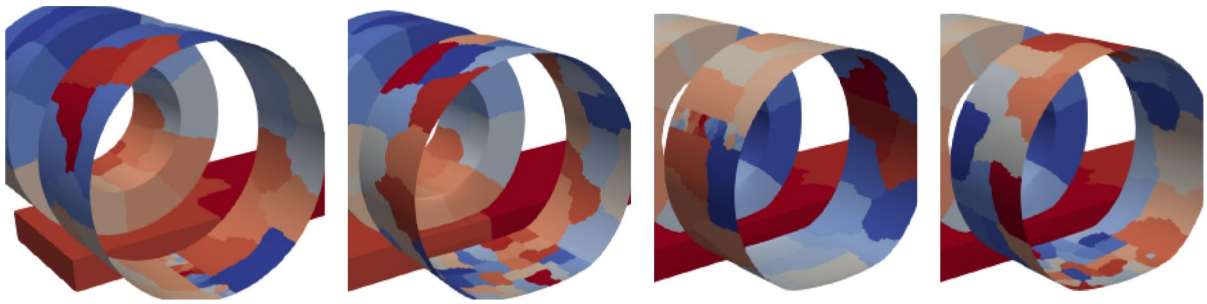
5.2 Rolling cylinder with dynamic contact

This example studies the behavior of parallel algorithms for dynamic contact problems, i.e. for uni-lateral contact problems where the contact zone is changing over time. This will exercise the parallel redistribution of the contact interface discretization to its full extent.

The problem is configured as follows: An elastic hollow cylinder is pushed onto a deformable block with initially flat surfaces. After contact has been established, a rotating motion is imposed on the inner surface of the hollow cylinder, somewhat mimicking a rolling tire. Both bodies are modeled with a compressible Neo-Hooke material with Young's modulus $E = 1$, Poisson's ratio $\nu = 0.3$, and density $\rho = 10^{-6}$.



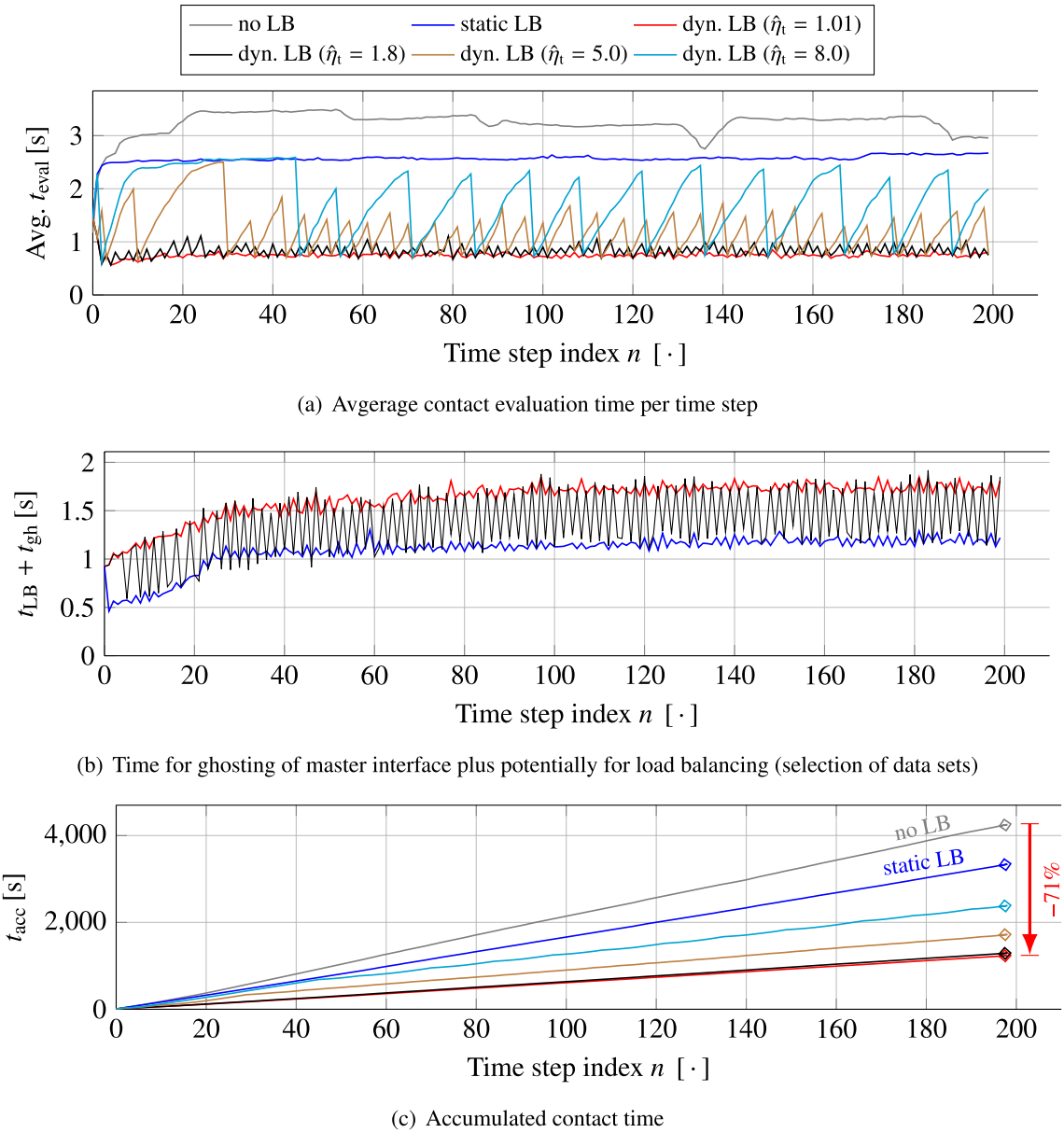
(a) Evolution of interface DD for different load balancing strategies



(b) Close-up of feature I (c) Close-up of feature II (d) Close-up of feature III (e) Close-up of feature IV

Fig. 12 Visualization of volume and interface subdomains for different load balancing strategies in a dynamic contact example. Interesting features are highlighted with roman numbers I–IV and discussed in the text

Both bodies are discretized with first-order hexahedral finite elements. The top surface of the block is chosen as the master side of the contact interface, while the outer surface of the hollow cylinder takes the role of the slave surface. For constraint enforcement, a node-based penalty regularization of the mortar approach with a penalty parameter of 5 is chosen. Time integration employs the generalized-α method [15] with spectral radius $\rho_\infty = 1.0$.



(a) Avgerage contact evaluation time per time step

(b) Time for ghosting of master interface plus potentially for load balancing (selection of data sets)

(c) Accumulated contact time

Fig. 13 Effect of different load balancing strategies on the time spent in the contact evaluation

Figure 12 exemplarily compares the volume and interface subdomains for the different load balancing strategies for the case of 24 MPI ranks. The initial subdomain layout in step 0 is the same for all load balancing strategies. While the DD of the underlying bodies will not be altered, we apply the interface load balancing scheme proposed in Sect. 4.3, which results in different interface DDs for the slave side. To unclutter the presentation, we only show the evolution of the slave side's interface DDs, since this is the key ingredient for a scalable mortar evaluation. Interesting features due to load balancing are highlighted with roman numbers I–IV (see also Fig. 12b–e) and will be discussed below. In the case of no load balancing (column "no LB"), the interface subdomains match the subdomains of the underlying volume DD throughout the entire simulation. For static LB (column "static LB"), an initial interface DD is performed at the beginning of step 1, but it is not updated during the simulation. Hence, a small strip of slave subdomains is generated during the initial load balancing phase (cf. highlight I or Fig. 12b) and then rotates with the rolling motion of the cylinder as marked by highlight III (see also Fig. 12d), such that it quickly leaves the contact area and, thus, does not contribute to an optimal contact evaluation throughout the entire simulation. Based on the threshold criterion (10), the dynamic load balancing (column "dyn. LB") updates the interface DD close to the contact area (cf. highlight II or

Fig. 12c) such that the interface DD is nearly optimal in the vicinity of the contact zone and all processes participate in the evaluation of the contact terms independent of the rolling motion of the cylinder (cf. highlight IV or Fig. 12e).

5.2.1 Effect of load balancing on wall clock time and memory consumption

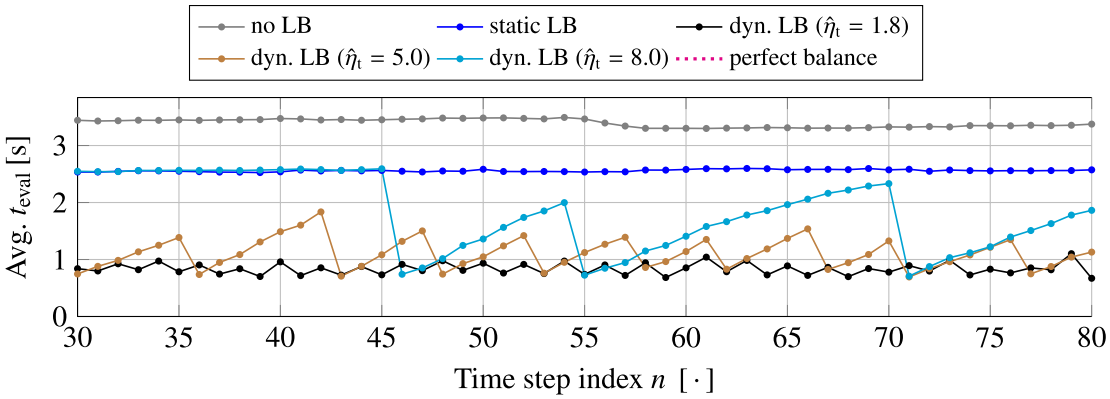
We compare the cases of no load balancing, an initial load balancing in the reference configuration, and the dynamic load balancing proposed in Sect. 4.4 on a mesh with 825,600 hexahedral elements consisting of 913,923 nodes and resulting in 2,741,769 displacement unknowns. We run the simulation on 96 MPI ranks on our in-house cluster. For the case of initial and dynamic load balancing, we limit the relative mismatch in subdomain size of the interface DD by setting the ZOLTAN parameter *IMBALANCE_TOL* to 1.03 [11]. For dynamic load balancing, we have tested different thresholds $\hat{\eta}_t \in \{1.01, 1.2, 1.5, 1.8, 2.5, 5.0, 8.0\}$ to trigger rebalancing, but will only report and discuss selected cases in the following for the sake of presentation, namely $\hat{\eta}_t \in \{1.01, 1.8, 5.0, 8.0\}$. To extend the ghosting of the master side's interface discretization, we rely on the *binning* strategy outlined in Sect. 3.5. A comparison of the different ghosting strategies is presented in Sect. 5.2.3.

We run the simulation for 200 time steps (20 time steps to close the initial gap, then 180 time steps of the rolling motion) to facilitate a rotation of 180°, such that the contact area on the outer cylinder surface substantially moves along the circumferential direction.

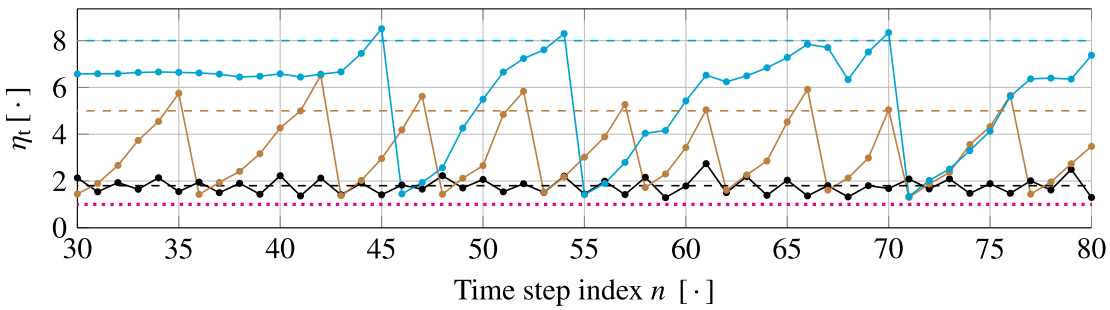
For every time step, Fig. 13a reports the average time per nonlinear iteration spent in contact evaluation (without considering the cost for load balancing). If no load balancing is performed, the average contact evaluation time is the largest. Since the slave side's interface DD is just adopted from the underlying volume discretization, some processes do never participate in contact evaluation. Moreover, the number of processes contributing to the contact evaluation changes over time, so the average contact evaluation time also changes over time steps. In contrast, static load balancing assures that all parallel processes hold their share of the slave side of the interface, such that the average contact evaluation time is roughly constant for all time steps (as soon as full contact is established). Since only a part of all processes contributes to the evaluation of the potentially active part of the slave interface, the average contact evaluation time is still rather large. Ultimately, dynamic load balancing triggers a rebalancing based on the current simulation status to aid a well-balanced distribution of the contact evaluation work to *all* parallel processes. In Fig. 13a, time steps just after a drop in t_{eval} are those, in which a rebalancing has occurred. Since the effort of mortar evaluation is now distributed to all processes, the time spent in mortar evaluation drops significantly on average. Of course, individual time steps with an imbalanced work distribution among processes might take longer, which will ultimately lead to rebalancing as soon as the rebalancing criterion (10) is met. In particular, a very low rebalancing threshold (e.g. $\hat{\eta}_t = 1.01$) requires to rebalance in basically every time step. Although this results in the overall fastest mortar evaluation, the additional effort for rebalancing limits the possible speed-up. On the other hand, a loose threshold (e.g. $\hat{\eta}_t \in \{5.0, 8.0\}$) triggers the rebalancing only a few times over the course of the simulation, however for some time steps the time spent in the mortar evaluation can grow by a factor of two or even three compared to the ideal case. In our numerical experiments, we have found the threshold $\hat{\eta}_t = 1.8$ to deliver a good compromise between imbalance in per-process workload and the frequency of rebalancing. Therefore, we will use this threshold value for all further studies.

Figure 13b reports the time t_{gh} spent for ghosting of the master side of the contact interface plus the time t_{LB} for rebalancing of the interface DD (if applicable). For the clarity of the presentation, we concentrate on three selected cases. While the time component t_{gh} for the master side ghosting is rather constant for all three cases, the time component t_{LB} varies: For static load balancing, only the first time step requires rebalancing, while all later time steps do not perform load balancing anymore. Hence, this curve peaks in the first time step and then drops and remains at low values. For dynamic load balancing with the strict imbalance threshold $\hat{\eta}_t = 1.01$, rebalancing occurs in every time step, such that this case consistently delivers high values for $t_{\text{LB}} + t_{\text{gh}}$. Obviously, these two cases can be interpreted as a lower and upper bound as evident from Fig. 13b. The case of dynamic load balancing with $\hat{\eta}_t = 1.8$ positions itself in between, since some time steps require rebalancing, but some do not.

While all cases with dynamic load balancing spend additional time on the redistribution of the interface subdomains, these additional timings are easily amortized. To this end, Fig. 13c plots the time $t_{\text{acc}} = \sum_{c=1}^{C} (t_{\text{eval}} + t_{\text{LB}} + t_{\text{gh}})_c, c \in \{1, \dots, C\}$, of all time components related to mortar evaluation over all time steps accrued over all C contact evaluations of the entire simulation. The end point markers are intended to highlight also small differences between curves. Naturally, a strict monotone increase is expected, while one aims for an as low as possible slope. Similar to the average contact evaluation time, static load balancing is beneficial compared to no load balancing at all, while dynamic load balancing results in the lowest contact evaluation times. Clearly, the better parallelization due to the dynamic load balancing strategy pays off the additional cost for occasional rebalancing. The lower the acceptable imbalance $\hat{\eta}_t$ is, the lower is the accumulated contact evaluation time t_{acc}. Overall, a maximum



(a) Avgerage contact evaluation time per time step



(b) Max/min ratio η_t in contact evaluation time across all parallel processes (solid lines) for different imbalance thresholds $\hat{\eta}_t$ (dashed lines)

Fig. 14 Detailed view of contact evaluation timings and its imbalance

reduction up to 71% in t_{acc} can be achieved through proper dynamic load balancing. We note that the difference in t_{acc} between $\hat{\eta}_t = 1.01$ and $\hat{\eta}_t = 1.8$ is very small, indicating that load balancing in every time step does not bring much additional value.

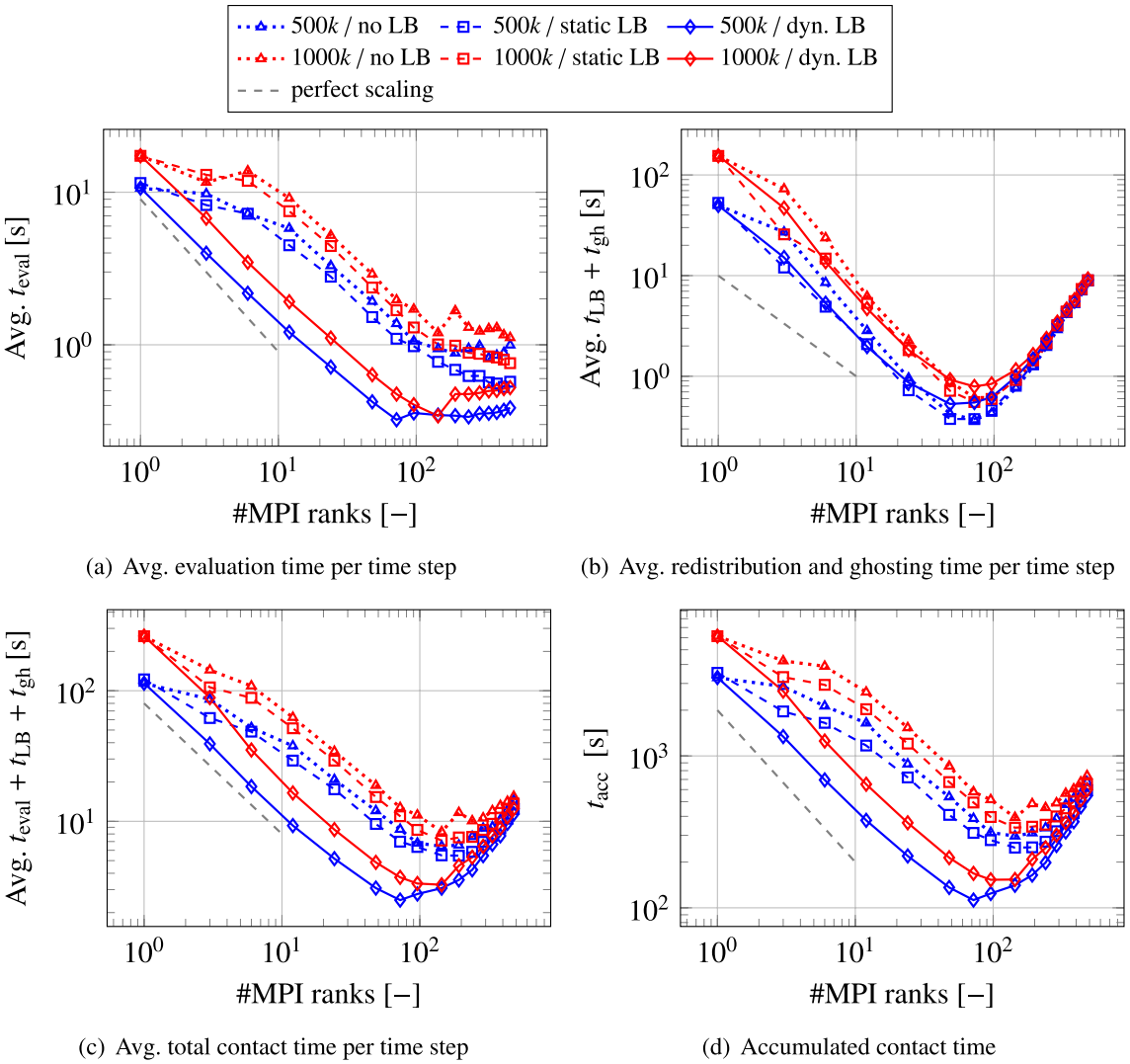
To demonstrate the effect of the rebalancing trigger $\hat{\eta}_t$ in detail, Fig. 14 shows a close-up of the results in Fig. 13a as well as the evolution of the max/min ratio η_t in contact evaluation time across all parallel processes. For a clearer visualization, only a subset of the results is plotted. In Fig. 14a, data points after a drop in t_{eval} correspond to time steps, where load balancing has occurred since the max/min ratio η_t exceeded the threshold $\hat{\eta}_t$ in the previous time step. This is in line with Fig. 14b, where η_t is plotted over time along with dashed lines to indicate the different thresholds $\hat{\eta}_t$. We observe that η_t drops close to the perfect balance (i.e. $\eta_t = 1.0$) just after it exceeded the threshold level $\hat{\eta}_t$. In favor of an uncluttered view, Fig. 14b shows only results obtained with dynamic load balancing.

So far, we have studied the impact of the load balancing strategy and the imbalance threshold $\hat{\eta}_t$ onto the time spent in the computational treatment of all mortar terms. In all cases, dynamic load balancing is worth the effort. Since the present example shows very good behavior for $\hat{\eta}_t = 1.8$, we continue to use this value throughout this example. We note that the optimal choice of $\hat{\eta}_t$ is problem-dependent. Yet, we generally recommend to use dynamic load balancing for contact problems with changing contact zones and select $\hat{\eta}_t$ on a case-by-case basis.

5.2.2 Strong scaling behavior under dynamic load balancing

Now, we study the strong scaling behavior of the contact evaluation time when dynamic load balancing is active. We therefore study two different problem sizes: 517,185 displacement unknowns referred to as “500k” and 1,005,993 displacement degrees of freedom denoted by “1000k”. While we keep the problem sizes fixed, we solve the problem on an increasing number of MPI ranks on our in-house cluster.

We will compare different load balancing strategies, namely no load balancing (“no LB”), initial load balancing in the reference configuration (“static LB”), and dynamic



(a) Avg. evaluation time per time step

(b) Avg. redistribution and ghosting time per time step

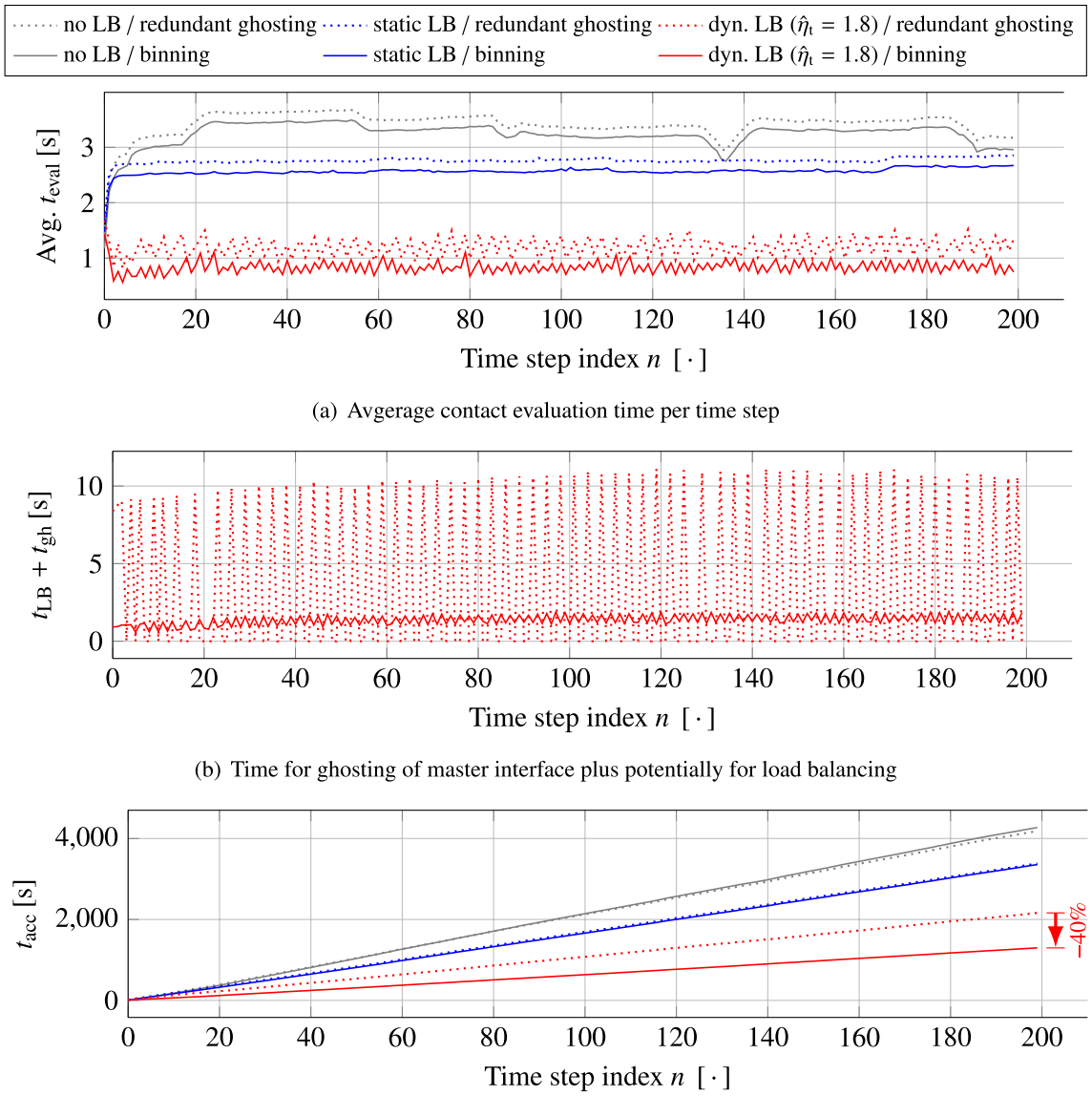
(c) Avg. total contact time per time step

(d) Accumulated contact time

Fig. 15 Strong scaling of the contact timings under different load balancing strategies

load balancing (with $\hat{\eta}_{\text{t}} = 1.8$ ("dyn. LB") as found useful in Sect. 5.2.1).

Figure 15 shows the strong scaling behavior. Again, we consider the average contact evaluation time t_{eval} per time step, the time $t_{\text{LB}} + t_{\text{gh}}$ spent in redistribution and ghosting of the interface discretizations, the average total time $t_{\text{total}} = t_{\text{eval}} + t_{\text{LB}} + t_{\text{gh}}$ per time step, and finally the total contact time $t_{\text{acc}} = \sum_{c=1}^{C} (t_{\text{eval}} + t_{\text{LB}} + t_{\text{gh}})_c, c \in \{1, \ldots, C\}$, accumulated over all C contact evaluations of the entire simulation. For both problem sizes as well as all quantities of interest, we observe good strong scaling behavior when using dynamic load balancing: starting from a small number of MPI ranks, the time spent on a given task (e.g. contact evaluation, redistribution and ghosting, total contact time, accumulated contact time) is reduced when adding more MPI ranks to tackle the computations, while the reduction rate is linked to the increase in MPI ranks, i.e. delivering perfect strong scaling [2]. As expected, both meshes reached their strong scaling limit at some point, such that adding more hardware resources does not reduce, but actually increase the execution time, e.g. due to a deteriorating computation-to-communication ratio. Naturally, the strong scaling limit of the large problem (1000k) is located at twice the number of MPI ranks as for the small, half-sized problem (500k). The beneficial effect of dynamic load balancing becomes evident in comparison to "no LB" and "static LB": Without any load balancing or just an initial rebalancing of the interface discretizations, the initial slope in the scaling diagrams is far from optimal. Once again, this originates from the curse of dimensionality, since the additional hardware resources do not necessarily participate in the interface evaluation. For an intermediate number of MPI ranks, strong scaling is recovered, however absolute timings



(a) Avgerage contact evaluation time per time step

(b) Time for ghosting of master interface plus potentially for load balancing

(c) Contact time t_{acc} accumulated over all time steps

Fig. 16 Effect of ghosting strategies on the contact timings: the combination of dynamic load balancing with ghosting via binning consistently delivers the fastest timings for contact evaluation

are much higher than for the same setup with dynamic load balancing. As already observed in Sect. 5.1.1, static load balancing is consistently a bit faster than using no load balancing at all, yet it is by far slower than dynamic load balancing.

As demonstrated, the proposed dynamic load balancing scheme is the key factor to achieve strong scalability of the evaluation of mortar terms in a nonlinear and time-dependent contact simulation. To the authors' best knowledge, this constitutes the first time that strong scalability in such a complex setting could be demonstrated.

5.2.3 Comparison of strategies to extend the master side's ghosting

While the influence of the load balancing strategy has already been discussed previously, we now aim to assess the impact of the ghosting strategy on the overall performance of the contact evaluation. Therefore, we exemplarily consider the mesh from Sect. 5.2.1 run on 96 MPI ranks. Now, we compare the fully redundant storage of the master side

of the interface (cf. Sect. 3.4) to the geometrically motivated binning approach (cf. Sect. 3.5). We study again the cases of no, static, and dynamic load balancing. For the clarity of the presentation, we only show the case of dynamic load balancing scenario with $\hat{\eta}_t = 1.8$, but note that other values for $\hat{\eta}_t$ exhibit similar behavior.

Figure 16 summarizes the wall clock time spent on contact evaluation. For the pure contact evaluation time reported in Fig. 16a, the fully redundant ghosting increases the evaluation time for all cases, since the contact detection needs to account for all master elements, while ghosting via binning pre-sorts the master elements based on their geometric proximity within neighboring bins.

Figure 16b depicts the time spent in redistribution and ghosting of interface data. For the sake of a clear presentation and to really focus on the most relevant case, we show only the curves for dynamic load balancing. Evidently, ghosting via binning is faster by a factor of $\approx 8-10\times$ than fully redundant ghosting.

Figure 16c shows the accumulated time for contact evaluation, load balancing, and ghosting, i.e. $t_{\text{acc}} = \sum_{c=1}^{C} (t_{\text{eval}} + t_{\text{LB}} + t_{\text{gh}})_c, c \in \{1, \dots, C\}$, to assess the overall accumulated time spent on all C evaluations of the contact interface over the course of the entire simulation. For the cases with no and static load balancing, the ghosting strategy does not impact the overall performance significantly. For dynamic load balancing though, the necessity of ghosting after each redistribution makes the difference: the performance difference between fully redundant ghosting and ghosting via binning as observed in Fig. 16b now accumulates over time, such that the use of binning results in the overall lowest time spent on contact evaluation. So, additional savings of 40% of the contact evaluation time can be achieved. Summing up the study of contact timings, the best case scenario of dynamic load balancing with ghosting via binning is faster than

- dynamic load balancing with fully redundant ghosting by a factor of ≈ 1.67,
- static load balancing by a factor of ≈ 2.61,
- no load balancing by a factor of ≈ 3.30,

which strongly emphasizes the benefits of dynamic load balancing and ghosting via binning in dynamic contact problems.

Finally, we briefly summarize the impact of the load balancing scheme and the ghosting strategy onto the cost for storage and parallel communication: If no load balancing is performed ("no LB"), the maximum number of owned nodes per process is roughly $10\times$ larger than its average across all processes, since not all processes hold a portion of the interface. This imbalance is alleviated for static or dynamic load balancing. Regarding the impact of the ghosting strategy, ghosting via binning reduces down the number of nodes/elements to be ghosted by a factor of $100\times$ compared to the fully redundant case, which ultimately also impacts the global memory footprint of the application.

In sum, dynamic contact problems require a good choice of load balancing strategy as well as a suitable ghosting strategy. In particular, load balancing highly impacts the time spent in contact evaluation. Despite the additional cost of performing the load balancing operation, the overall fastest contact evaluation is achieved with dynamic load balancing based on a user-given imbalance threshold $\hat{\eta}_t$. While we have found $\hat{\eta}_t = 1.8$ to deliver very good results in our numerical studies, the optimal choice of $\hat{\eta}_t$ can depend on details of the computing hardware, the software implementation, and also the example at hand. To reduce the amount of parallel communication as well as the memory demand per compute node, ghosting via binning is by far superior to a fully redundant storage of the master side of the interface discretization. The overall best performance with respect to both phenomena (run time and communication/memory demand) is obtained through the combination of dynamic load balancing with ghosting via binning.

6 Concluding remarks

Recognizing the tremendous computational effort to evaluate mortar integrals in the context of non-matching interface discretizations as they exemplarily arise in contact mechanics, this paper proposes strategies for efficient storage and parallel computational kernels for mortar interface problems. Starting from a closer look at the tasks and the computational effort to evaluate mortar integrals, we have derived two basic requirements for computations on parallel machines with distributed memory architecture: On the one hand, one needs to enable access to the appropriate interface data to guarantee a correct identification of all master/slave pairs at the mortar interface. On the other hand, the available parallel hardware needs to be used efficiently, such that parallel scalability of the mortar evaluation can be achieved.

We have found the combination of efficient ghosting of data from the master side of the mortar interface using as few as possible parallel communication together with a scalable evaluation of mortar integrals to be crucial for the overall efficiency and performance of mortar evaluations. Regarding the finite element assembly of the mortar operators, special care needs to be taken for off-process values. In our implementation, we strictly follow the traditional implementation of overlapping DD, i.e. we ghost elements whose nodes are owned by more than one MPI rank and then evaluate these elements on each of the participating processes,

such that each process only needs to write into rows of the matrices that reside on this process. As an alternative, specialized data structures for sparse matrices (e.g. from Trilinos' Tpetra package[3]) allow for off-process assembly. Then, shared elements are only evaluated once and result values need to be communicated to other processes after the evaluation. The faster option for finite element assembly in parallel highly depends on the implementation and software stack at hand, so we cannot give a general recommendation. For both approaches, a sparse communication pattern as proposed in [44] could speed-up the parallel communication even further.

For the ghosting of interface data, we have discussed techniques to guarantee access to all required master/slave pairs during the contact search and mortar evaluation. While fully redundant ghosting is conceptually easy and straightforward to implement, it suffers from elevated memory demands and tremendous communication overhead at large scale, which ultimately increases the overall time-to-solution. A geometrically motivated approach using a background grid of Cartesian bins allows for the efficient identification of nearby master elements, reduces the per-process memory demand as well as limits the number of master elements to be ghosted. The binning approach has shown the best timings in weak and strong scaling studies and consistently reduces the amount of data to be communicated between parallel processes as well as to be stored within a process.

Regarding a scalable evaluation of the mortar integrals on parallel architectures, we have then discussed the curse of dimensionality in overlapping DDs of interface problems, which requires a special treatment of the interface subdomains. To this end, we have proposed to use an interface DD independent from the underlying volume DD and were able to demonstrate optimal weak and strong scalability of the mortar evaluation time. To account for dynamic changes in the contact zone, we have designed a dynamic load balancing scheme for contact problems, which tracks imbalances among parallel processes and rebalances the computational work as soon as user-given imbalance thresholds are exceeded. We have tested the proposed algorithms on a time-dependent nonlinear contact problem undergoing large deformations. In time measurements on such large-scale examples, dynamic load balancing outperforms the case of no or only initial load balancing by factors up to 2−4×. Wall clock time is the lowest, when only small imbalances are allowed, although even a large imbalance tolerance delivers faster computations than simulations without any load balancing at all. For the first time, strong and weak scalability could be shown for time-dependent nonlinear contact problems undergoing large deformations and dynamically evolving contact zones through the application of the proposed dynamic load balancing scheme.

In our numerical experiments, we have studied representative test cases from computational contact mechanics. We have performed weak and strong scaling studies up to 480 MPI ranks as well as have assessed the impact of different algorithmic parameters. From our numerical experiments, we extract several findings:

- Ghosting via binning is favorable due to its reduced communication overhead, which also directly reduces the time-to-solution.
- Load balancing is crucial for optimal contact evaluation times. In particular, systems with a static contact zone benefit from an initial redistribution of the interface, while contact problems with dynamically evolving contact zones require the proposed dynamic load balancing scheme for optimal performance.
- For static contact problems, we have found the combination of static load balancing and ghosting via binning to deliver the best results.
- For dynamic contact problems, we have found the combination of dynamic load balancing and ghosting via binning to deliver the best results.

In sum, we recommend to apply static load balancing in combination with ghosting via binning for problems with static contact zones, while dynamic load balancing in combination with ghosting via binning is preferable for problems with dynamically evolving contact zones. Following these recommendations, a fast time-to-solution as well as good weak and strong scaling behavior can be achieved.

Acknowledgements This work has been partially funded by the Deutsche Forschungsgemeinschaft (DFG-German Research Foundation) within the project "Experimental characterization and numerical simulation of the automated fiber placement (AFP) process for thermoplastic fiber-reinforced plastics" (Project number 325153381).

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability statement All data are available from the corresponding author upon reasonable request.

Declarations

Conflict of interest The authors have no other competing interests directly or indirectly related to this manuscript.

[3] https://trilinos.github.io/tpetra.html.

References

1. Achdou Y, Maday Y, Widlund OB (1999) Iterative substructuring preconditioners for mortar element methods in two dimensions. SIAM J Numer Anal 36(2):551–580
2. Amdahl GM (1967) Validity of the single-processor approach to achieving large scale computing capabilities. In: AFIPS conference proceedings. AFIPS Press, Reston/Va, vol 30, pp 483–485
3. BACI: a comprehensive multi-physics simulation framework. https://baci.pages.gitlab.lrz.de/website/. Accessed 3 Feb 2022
4. Bavier E, Hoemmen M, Rajamanickam S, Thornquist H (2012) Amesos2 and Belos: direct and iterative solvers for large sparse linear systems. Sci Program 20(3):241–255
5. Belgacem FB (1999) The Mortar finite element method with Lagrange multipliers. Numerische Mathematik 84(2):173–197
6. Belgacem FB, Hild P, Laborde P (1998) The mortar finite element method for contact problems. Math Comput Model 28(4–8):263–271
7. Bellmann R (1957) Dynamic programming. Princeton University Press, Princeton
8. Benson DJ, Hallquist JO (1990) A single surface contact algorithm for the post-buckling analysis of shell structures. Comput Methods Appl Mech Eng 78:141–163
9. Berger-Vergiat L, Glusa CA, Hu JJ, Mayr M, Prokopenko A, Siefert CM, Tuminaro RS, Wiesner TA (2019) MueLu user's guide 2.0. Technical Report SAND2019-0537, Sandia National Laboratories, Albuquerque, NM 87185, USA
10. Bernardi C, Maday Y, Patera AT (1993) Domain decomposition by the mortar element method. In: Kaper HG, Garbey M, Pieper GW (eds) Asymptotic and numerical methods for partial differential equations with critical parameters. Springer, Dordrecht, pp 269–286
11. Boman EG, Çatalyürek UV, Chevalier C, Devine KD (2012) The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: partitioning, ordering and coloring. Sci Program 20(2):29–150
12. Brivadis E, Buffa A, Wohlmuth B, Wunderlich L (2015) The influence of quadrature errors on isogeometric mortar methods. In: Jüttler B, Simeon B (eds) Isogeometric analysis and applications 2014. Springer, Berlin
13. Brucker P (2007) Scheduling algorithms, 5th edn. Springer, Berlin
14. Casarin MA, Widlund OB (1996) A hierarchical preconditioner for the mortar finite element method. Electron Trans Numer Anal 4:75–88
15. Chung J, Hulbert G (1993) A time integration algorithm for structural dynamics with improved numerical dissipation: the generalized-$\alpha $ method. J Appl Mech 60(2):371–375
16. De Lorenzis L, Wriggers P, Hughes TJ (2014) Isogeometric contact: a review. GAMM-Mitteilungen 37(1):85–123
17. De Lorenzis L, Wriggers P, Zavarise G (2012) A mortar formulation for 3D large deformation contact using NURBS-based isogeometric analysis and the augmented Lagrangian method. Comput Mech 49:1–20
18. Dickopf T, Krause R (2009) Efficient simulation of multi-body contact problems on complex geometries: a flexible decomposition approach using constrained minimization. Int J Numer Methods Eng 77(13):1834–1862
19. Dittmann M, Franke M, Temizer I, Hesch C (2014) Isogeometric analysis and thermomechanical Mortar contact problems. Comput Methods Appl Mech Eng 274:192–212
20. Dittmann M, Schuß S, Wohlmuth B, Hesch C (2019) Weak C^n coupling for multipatch isogeometric analysis in solid mechanics. Int J Numer Methods Eng 118(11):678–699
21. Dittmann M, Schuß S, Wohlmuth B, Hesch C (2020) Crosspoint modification for multi-patch isogeometric analysis. Comput Methods Appl Mech Eng 360:112768
22. Dolean V, Jolivet P, Nataf F (2015) An introduction to domain decomposition methods: algorithms, theory and parallel implementation. SIAM, Philadelphia
23. Dornisch W, Stöckler J, Müller R (2017) Dual and approximate dual basis functions for B-splines and NURBS—comparison and application for an efficient coupling of patches with the isogeometric mortar method. Comput Methods Appl Mech Eng 315:449–496
24. Dornisch W, Vitucci G, Klinkel S (2015) The weak substitution method—an application of the mortar method for patch coupling in NURBS-based isogeometric analysis. Int J Numer Meth Eng 103(3):205–234
25. Duong TX, De Lorenzis L, Sauer RA (2019) A segmentation-free isogeometric extended mortar contact method. Comput Mech 63:383–407
26. Ehrl A, Popp A, Gravemeier V, Wall WA (2014) A dual mortar approach for mesh tying within a variational multiscale method for incompressible flow. Int J Numer Methods Fluids 76(1):1–27
27. Fang R, Farah P, Popp A, Wall WA (2018) A monolithic, mortar-based interface coupling and solution scheme for finite element simulations of lithium-ion cells. Int J Numer Methods Eng 114(13):1411–1437
28. Farah P, Popp A, Wall WA (2015) Segment-based vs. element-based integration for mortar methods in computational contact mechanics. Comput Mech 55(1):209–228
29. Farah P, Vuong A-T, Wall WA, Popp A (2016) Volumetric coupling approaches for multiphysics simulations on non-matching meshes. Int J Numer Methods Eng 108(12):1550–1576
30. Farah P, Wall WA, Popp A (2016) An implicit finite wear contact formulation based on dual mortar methods. Int J Numer Methods Eng 111(4):325–353
31. Flemisch B, Wohlmuth BI (2007) Stable Lagrange multipliers for quadrilateral meshes of curved interfaces in 3D. Comput Methods Appl Mech Eng 196(8):1589–1602
32. Foley J (1997) Computer graphics: principles and practice. Addison-Wesley, Boston
33. Gee MW, Siefert CM, Hu JJ, Tuminaro RS, Sala MG (2006) ML 5.0 smoothed aggregation user's guide. Technical Report SAND2006-2649, Sandia National Laboratories, Albuquerque, NM, 87185, USA
34. Gitterle M, Popp A, Gee MW, Wall WA (2010) Finite deformation frictional mortar contact using a semi-smooth Newton method with consistent linearization. Int J Numer Methods Eng 84(5):543–571
35. Gustafson JL (1988) Reevaluating Amdahl's law. Commun ACM 31(5):532–533
36. Hansen GA, Xavier PG, Mish SP, Voth TE, Heinstein MW, Glass MW (2016) An MPI+X implementation of contact global search using Kokkos. Eng Comput 32:295–311
37. Hartmann S, Brunssen S, Ramm E, Wohlmuth BI (2007) Unilateral non-linear dynamic contact of thin-walled structures using a primal-dual active set strategy. Int J Numer Methods Eng 70(8):883–912

38. Heroux MA (2007) AztecOO user guide. Technical Report SAND2004-3796, Sandia National Laboratories, Albuquerque, NM, 87185, USA
39. Hesch C, Betsch P (2009) A mortar method for energy–momentum conserving schemes in frictionless dynamic contact problems. Int J Numer Methods Eng 77(10):1468–1500
40. Hesch C, Betsch P (2011) Transient three-dimensional contact problems: mortar method. Mixed methods and conserving integration. Comput Mech 48:461–475
41. Hesch C, Betsch P (2012) Isogeometric analysis and domain decomposition methods. Comput Methods Appl Mech Eng 213–216:104–112
42. Hesch C, Gil AJ, Arranz Carreño A, Bonet J, Betsch P (2014) A mortar approach for fluid–structure interaction problems: immersed strategies for deformable and rigid bodies. Comput Methods Appl Mech Eng 278:853–882
43. Hintermüller M, Ito K, Kunisch K (2003) The primal-dual active set strategy as a semismooth Newton method. SIAM J Optim 13(3):865–888
44. Hoefler T, Siebert C, Lumsdaine A (2010) Scalable communication protocols for dynamic sparse data exchange. ACM SIGPLAN Not 45(5):159–168
45. Hüeber S, Stadler G, Wohlmuth BI (2008) A primal-dual active set algorithm for three-dimensional contact problems with Coulomb friction. SIAM J Sci Comput 30(2):527–596
46. Hüeber S, Wohlmuth BI (2005) A primal-dual active set strategy for non-linear multibody contact problems. Comput Methods Appl Mech Eng 194(27–29):3147–3166
47. Klöppel T, Popp A, Küttler U, Wall WA (2011) Fluid–structure interaction for non-conforming interfaces based on a dual mortar formulation. Comput Methods Appl Mech Eng 200(45–46):3111–3126
48. Krause R, Zulian P (2016) A parallel approach to the variational transfer of discrete fields between arbitrarily distributed unstructured finite element meshes. SIAM J Sci Comput 38(3):C307–C333
49. Lamichhane BP, Stevenson RP, Wohlmuth BI (2005) Higher order mortar finite element methods in 3D with dual Lagrange multiplier bases. Numerische Mathematik 102(1):93–121
50. Lamichhane BP, Wohlmuth BI (2007) Biorthogonal bases with local support and approximation properties. Math Comput 76(257):233–249
51. Maday Y, Rapetti F, Wohlmuth BI (2002) The influence of quadrature formulas in 2D and 3D mortar element methods. In: Pavarino LF, Toselli A (eds) Recent developments in domain decomposition methods, vol 23. Lecture notes in computational science and engineering. Springer, Berlin
52. Mayr M, Klöppel T, Wall WA, Gee MW (2015) A temporal consistent monolithic approach to fluid–structure interaction enabling single field predictors. SIAM J Sci Comput 37(1):B30–B59
53. McDevitt TW, Laursen TA (2000) A mortar-finite element formulation for frictional contact problems. Int J Numer Methods Eng 48(10):1525–1547
54. Message Passing Interface Forum (2021) MPI: a message-passing interface standard version 4.0
55. Plimpton S (1995) Fast parallel algorithms for short-range molecular dynamics. J Comput Phys 117(1):1–19
56. Popp A, Gee MW, Wall WA (2009) A finite deformation mortar contact formulation using a primal-dual active set strategy. Int J Numer Methods Eng 79(11):1354–1391
57. Popp A, Gitterle M, Gee MW, Wall WA (2010) A dual mortar approach for 3D finite deformation contact with consistent linearization. Int J Numer Methods Eng 83(11):1428–1465
58. Popp A, Wohlmuth BI, Gee MW, Wall WA (2012) Dual quadratic mortar finite element methods for 3D finite deformation contact. SIAM J Sci Comput 34(4):B421–B446
59. Puso MA (2004) A 3D mortar method for solid mechanics. Int J Numer Methods Eng 59(3):315–336
60. Puso MA, Laursen TA (2003) Mesh tying on curved interfaces in 3D. Eng Comput 20(3):305–319
61. Puso MA, Laursen TA (2004) A mortar segment-to-segment contact method for large deformation solid mechanics. Comput Methods Appl Mech Eng 193(6–8):601–629
62. Puso MA, Laursen TA (2004) A mortar segment-to-segment frictional contact method for large deformations. Comput Methods Appl Mech Eng 193(45–47):4891–4913
63. Puso MA, Laursen TA, Solberg J (2008) A segment-to-segment mortar contact method for quadratic elements and large deformations. Comput Methods Appl Mech Eng 197(6):555–566
64. Quarteroni A, Valli AMP (2005) Domain decomposition methods for partial differential equations. Clarendon, Oxford
65. Scott LR, Zhang S (1990) Finite element interpolation of nonsmooth functions satisfying boundary conditions. Math Comput 54(190):483–493
66. Seitz A, Farah P, Kremheller J, Wohlmuth BI, Wall WA, Popp A (2016) Isogeometric dual mortar methods for computational contact mechanics. Comput Methods Appl Mech Eng 301:259–280
67. Smith B, Bjørstad P, Gropp W (2008) Domain decomposition: parallel multilevel methods for elliptic partial differential equations. Cambridge University Press, Cambridge
68. Stefanica D (2001) A numerical study of FETI algorithms for mortar finite element methods. SIAM J Sci Comput 23(4):1135–1160
69. Toselli A, Widlund OB (2005) Domain decomposition methods: algorithms and theory, vol 34. Springer series in computational mathematics. Springer, Berlin
70. Wieners C, Wohlmuth BI (1999) A general framework for multigrid methods for mortar finite elements. Technical Report 415, Institut für Mathematik, Universität Augsburg
71. Wieners C, Wohlmuth BI (2003) Duality estimates and multigrid analysis for saddle point problems arising from mortar discretizations. SIAM J Sci Comput 24(6):2163–2184
72. Wiesner TA, Mayr M, Popp A, Gee MW, Wall WA (2021) Algebraic multigrid methods for saddle point systems arising from mortar contact formulations. Int J Numer Methods Eng 122(15):3749–3779
73. Wiesner TA, Popp A, Gee MW, Wall WA (2018) Algebraic multigrid methods for dual mortar finite element formulations in contact mechanics. Int J Numer Methods Eng 114(4):399–430
74. Wilking C, Bischoff M (2017) Alternative integration algorithms for three-dimensional mortar contact. Comput Mech 59:203–218
75. Williams JR, O'Connor R (1995) A linear complexity intersection algorithm for discrete element simulation of arbitrary geometries. Eng Comput 12:185–201
76. Williams JR, O'Connor R (1999) Discrete element simulation and the contact problem. Arch Comput Methods Eng 6:279–304
77. Wohlmuth BI (2000) A mortar finite element method using dual spaces for the Lagrange multiplier. SIAM J Numer Anal 38(3):989–1012
78. Wohlmuth BI (2000) A multigrid method for saddle point problems arising from mortar finite element discretizations. Electron Trans Numer Anal 11:43–54
79. Wohlmuth BI (2001) Discretization methods and iterative solvers based on domain decomposition, vol 17. Springer, Heidelberg
80. Wohlmuth BI (2011) Variationally consistent discretization schemes and numerical algorithms for contact problems. Acta Numer 20:569–734
81. Wohlmuth BI, Popp A, Gee MW, Wall WA (2012) An abstract framework for a priori estimates for contact problems in 3d with quadratic finite elements. Comput Mech 49(6):735–747
82. Wriggers P (2006) Computational contact mechanics, 2nd edn. Springer, Berlin

83. Wunderlich L, Seitz A, Alaydın MD, Wohlmuth B, Popp A (2019) Biorthogonal splines for optimal weak patch-coupling in isogeometric analysis with applications to finite deformation elasticity. Comput Methods Appl Mech Eng 346:197–215
84. Yang B, Laursen TA (2008) A contact searching algorithm including bounding volume trees applied to finite sliding mortar formulations. Comput Mech 41:189–205
85. Yang B, Laursen TA (2008) A large deformation mortar formulation of self contact with finite sliding. Comput Methods Appl Mech Eng 197(6):756–772
86. Yang B, Laursen TA, Meng X (2005) Two dimensional mortar contact methods for large deformation frictional sliding. Int J Numer Methods Eng 62(9):1183–1225
87. Zhong Z-H, Nilsson L (1989) A contact searching algorithm for general contact problems. Comput Struct 33(1):197–209
88. Zhong Z-H, Nilsson L (1990) A contact searching algorithm for general 3-D contact-impact problems. Comput Struct 34(2):327–335
89. Zou Z, Scott MA, Borden MJ, Thomas DC, Dornisch W, Brivadis E (2018) Isogeometric Bézier dual mortaring: refineable higher-order spline dual bases and weakly continuous geometry. Comput Methods Appl Mech Eng 333:497–534