SIAM J. SCI. COMPUT.
Vol. 42, No. 6, pp. B1429–B1461

STENCIL SCALING FOR VECTOR-VALUED PDES ON HYBRID GRIDS WITH APPLICATIONS TO GENERALIZED NEWTONIAN FLUIDS*

DANIEL DRZISGA†, ULRICH RÜDE‡, AND BARBARA WOHLMUTH†

Abstract. Matrix-free finite element implementations for large applications provide an attractive alternative to standard sparse matrix data formats due to the significantly reduced memory consumption. Here, we show that they are also competitive with respect to the run-time in the low-order case if combined with suitable stencil scaling techniques. We focus on variable coefficient vector-valued partial differential equations as they arise in many physical applications. The presented method is based on scaling constant reference stencils originating from a linear finite element discretization instead of evaluating the bilinear forms on the fly. This method assumes the usage of hierarchical hybrid grids, and it may be applied to vector-valued second-order elliptic partial differential equations directly or as a part of more complicated problems. We provide theoretical and experimental performance estimates showing the advantages of this new approach compared to the traditional on-the-fly integration and stored matrix approaches. In our numerical experiments, we consider two specific mathematical models, namely, linear elastostatics and incompressible Stokes flow. The final example considers a nonlinear shear-thinning generalized Newtonian fluid. For this type of nonlinearity, we present an efficient approach for computing a regularized strain rate which is then used to define the nodewise viscosity. Depending on the compute architecture, we could observe maximum speedups of 64% and 122% compared to the on-the-fly integration. The largest considered example involved solving a Stokes problem with 12288 compute cores on the state-of-the-art supercomputer SuperMUC-NG.

Key words. matrix-free, finite elements, variable coefficients, stencil scaling

AMS subject classifications. 65N30, 65N55, 65Y05, 65Y20

DOI. 10.1137/19M1267891

1. Introduction.

In this article, we study the efficiency of large-scale low-order finite element computations, and we examine which accuracy can be obtained at what cost. High performance computing is expensive, not only in terms of investments in supercomputer systems, but also in terms of operational cost. In particular, energy consumption is becoming a critical factor; see, e.g., the emerging rankings such as the GREEN500 list.[1] Therefore, it is crucial to rethink long-established computing practices and to study, quantify, and improve the efficiency of current numerical algorithms.

We primarily strive to reduce the absolute compute times. This is, of course, a viable goal in its own right, but the compute times are also directly related to the required energy for a computation. At this point, we note that while scalability is necessary for efficient large-scale parallel computing, scalability alone does not imply an efficient use of resources. In fact, inefficient codes are often found to scale better than efficient ones. Similarly, the asymptotic convergence rate of a discretization scheme is an important mathematical criterion affecting the accuracy, but ultimately

*Submitted to the journal's Computational Methods in Science and Engineering section June 13, 2019; accepted for publication (in revised form) August 27, 2020; published electronically December 1, 2020.

https://doi.org/10.1137/19M1267891

Funding: This work was partly supported by the German Research Foundation through the Priority Programme 1648 "Software for Exascale Computing" (SPPEXA), and by grant WO671/11-1.

†Institute for Numerical Mathematics (M2), Technische Universität München, 85748 Garching bei München, Germany (drzisga@ma.tum.de, wohlmuth@ma.tum.de).

‡Department of Computer Science 10, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, 91058, Germany (ulrich.ruede@fau.de).

[1] https://www.top500.org/green500/lists/2019/11/

only the error itself matters, including the constants involved. Such considerations gain additional relevance at a time when Moore's law slows down and technological progress is no longer producing computers that automatically run twice as fast with every new year. In this situation, innovation and improvements must rely increasingly on better implementations and on algorithms that are better suited for the available architectures.

Considering efficiency in this more rigorous sense, it is found that data transport is a critical factor in addition to the executed operations. Here, data transport includes not only message passing communication in a large parallel cluster but also the data transport within each node of such a cluster, i.e., from main memory to the CPU—and even within a CPU between the different layers of caches and the registers of the functional units [19]. The energy consumption for operations and data transport in a typical CPU architecture has been quantified in [1]. Additionally, it is, of course essential to exploit fine-grained concurrency in the form of multinode architectures and by the use of vectorization. In order to achieve optimal performance, we must be aware that the current speed of memory cannot keep up with the speed of processors and that most of the energy is spent on the data transfers. Therefore, an important characteristic relevant for the efficiency of numerical algorithms on modern computers is their *balance* or *floating-point intensity*, i.e., the ratio of floating-point operations (FLOPs) performed per byte of memory access [19].

Almost all traditional finite element libraries construct global stiffness matrices by looping over local elements and adding their contributions to the global matrix. Even when stored in compressed formats, these matrices require significantly more memory than storing the solution vectors. Not only the memory consumption does present a challenge, we also need to take into account the memory traffic and latency in loading the nonzero matrix indices and entries.

To improve on the memory consumption and memory access, matrix-free methods constitute a possible remedy where only the results of matrix vector products are computed without assembling and storing the whole global matrix. Different strategies exist to implement matrix-free methods, but the predominant candidate for low-order finite elements is the element-by-element approach [3, 9, 12, 15, 35], wherein local stiffness matrices are multiplied by local vectors and later added to the global solution vector. These local stiffness matrices may be either stored individually in memory—which actually requires more memory than storing the global matrix—or computed on the fly. When using high-order finite elements, the weak forms can be integrated on the fly using standard or reduced quadrature formulas [11, 24, 25, 26, 30]. This is a well-suited strategy for future architectures because of its high arithmetic intensity [27], but we present a method that can compete with matrix-based methods even in the low-order case. In [4], we presented an alternative matrix-free stencil scaling approach for accelerating low-order finite element implementations suited for scalar second-order elliptic partial differential equations (PDEs). There, it was shown that the method was able to reduce the computational cost significantly.

Here, we will expand on this idea and present a similar matrix-free approach for vector-valued second-order elliptic PDEs. The construction is based on the use of hierarchical hybrid grids (HHGs) which form the basis in the HHG [6, 7, 18] and Hybrid Tetrahedral Grids (HyTeG) [23] frameworks. These grids are constructed by starting with an initial, possibly unstructured, simplicial triangulation of a polygonal domain and refining each element multiple times uniformly in order to create a hierarchy of meshes. Ultimately, we associate to each of these meshes a piecewise linear finite element space. By exploiting the structure obtained by these uniform refinements, it is

possible to improve the performance of the finite element solver by using a stencil-based code. Vector-valued second-order elliptic PDEs arise in the modeling of elastostatics and fluid dynamics and play an important role in mathematical modeling. We show that the idea of the scalar stencil scaling cannot be applied to these equations, since for vector-valued PDEs the simple scaling results in the discretization of a different PDE if the coefficient is not constant. Thus, there is a need for a modified stencil scaling method that is also suited for matrix-free finite element implementations on HHGs. Although this vector-valued scaling is more complicated and more expensive than the scalar stencil scaling, it has the ability to reproduce the standard finite element solutions while requiring only a fraction of the time to obtain them.

The principal novelty of this paper is the presentation of an improved method for assembling stencils for vector-valued second-order elliptic PDEs suitable for matrix-free solvers on HHGs coupled with a linear finite element discretization. We provide theoretical and experimental performance comparisons which outline the advantages of the stencil scaling approach. Furthermore, we show the convergence and the run-times of this extended stencil scaling through numerical experiments. In these experiments, we consider two specific mathematical models, namely, linear elastostatics and generalized incompressible Stokes flow. In the final example, a nonlinear shear-thinning non-Newtonian example is considered, where the viscosity depends on the shear rate.

2. Model equations and discretization. The goal of this paper is to speed up matrix-free finite element implementations for solving vector-valued second-order elliptic PDEs in a domain $\Omega \subset \mathbb{R}^d$, $d \in \{2, 3\}$, of the form

$$\begin{aligned} -\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} &= \mathbf{f} \quad \text{in } \Omega, \\ \mathbf{u} &= \mathbf{g} \quad \text{on } \Gamma_{\mathrm{D}}, \\ \boldsymbol{\sigma} \cdot \mathbf{n} &= \hat{\mathbf{t}} \quad \text{on } \Gamma_{\mathrm{N}}, \end{aligned} \tag{2.1}$$

where the stress $\boldsymbol{\sigma} = \boldsymbol{\sigma}(\boldsymbol{\varepsilon})$ depends on the strain and additional material parameters. One particular example for $\boldsymbol{\sigma}$ that we investigate more thoroughly is the stress tensor for linear elasticity with isotropic continuous materials given by Hooke's law as $\boldsymbol{\sigma}(\boldsymbol{\varepsilon}) = 2\mu\boldsymbol{\varepsilon} + \lambda \operatorname{tr}(\boldsymbol{\varepsilon}) I$. Furthermore, generalized incompressible Stokes flow problems may also be cast in this form when adding additional constraints. In this case, the stress tensor is defined by $\boldsymbol{\sigma}(\boldsymbol{\varepsilon}) = 2\mu\boldsymbol{\varepsilon} - pI$, where an additional pressure variable p has been introduced, and the incompressibility constraint $\boldsymbol{\nabla} \cdot \mathbf{u} = 0$ in Ω is enforced. The domain boundary $\partial\Omega$ is split into two disjoint parts, the nontrivial Dirichlet boundary Γ_{D} and the Neumann boundary Γ_{N}. See Table 1 for a complete list of occurring variables and their definitions.

TABLE 1
Required symbols and their definitions.

Symbol	Definition
$\mathbf{u}$	displacement or velocity
p	pressure
$\boldsymbol{\varepsilon}$	strain: $\frac{1}{2}\left(\boldsymbol{\nabla}\mathbf{u} + \boldsymbol{\nabla}\mathbf{u}^{\top}\right)$
$\mathbf{f}$	body forces
$\mathbf{g}$	prescribed displacement or velocity
$\hat{\mathbf{t}}$	external forces
$\mathbf{n}$	outward-pointing unit-normal vector
λ	Lamé's first parameter
μ	shear modulus/dynamic viscosity

For the rest of this section, we restrict ourselves to the case of linear elastostatics, since the method may be applied to the momentum balance of the Stokes equations in the same way. This is demonstrated in the numerical results presented in subsection 5.2. For simplicity, we consider a homogeneous Dirichlet boundary Γ_{D} for the rest of this section, so $\mathbf{g} = \mathbf{0}$. The weak form of (2.1) in the case of linear elastostatics employing Hooke's law reads as follows: For a suitable space V incorporating the Dirichlet boundary conditions, find $\mathbf{u} \in V$ such that $a(\mathbf{u}, \mathbf{v}) = f(\mathbf{v}) \; \forall \, \mathbf{v} \in V$, where

$$
\begin{aligned}
a(\mathbf{u}, \mathbf{v}) &= \langle 2\mu \boldsymbol{\varepsilon}(\mathbf{u}), \boldsymbol{\varepsilon}(\mathbf{v}) \rangle_{\Omega} + \langle \lambda \boldsymbol{\nabla} \cdot \mathbf{u}, \boldsymbol{\nabla} \cdot \mathbf{v} \rangle_{\Omega}, \\
f(\mathbf{v}) &= \langle \mathbf{f}, \mathbf{v} \rangle_{\Omega} + \langle \hat{\mathbf{t}}, \mathbf{v} \rangle_{\Gamma_N}.
\end{aligned} \tag{2.2}
$$

By $\langle \cdot, \cdot \rangle_{\Omega}$ we denote the standard duality product in V.

In order to discretize the problem, we decompose the computational domain in the typical HHG manner [5, 6, 7]. Let $\mathcal{T}_H$ be a possibly unstructured simplicial triangulation of a bounded polygonal or polyhedral domain $\Omega \subset \mathbb{R}^d$, $d \in \{2, 3\}$. Based on this initial grid, we construct a hierarchy of $L+2$, $L \in \mathbb{N}$, grids $\mathcal{T} = \{\mathcal{T}_h, \; h = 2^0 H, \ldots, 2^{-L-1} H\}$ by successive global uniform refinement. As is standard, each of these refinements is achieved by subdividing all elements into 2^d subelements. For details of the refinement in 3D, we refer the reader to [8]. We also call $\mathcal{T}_H$ macro-triangulation and denote its elements by T, whereas the elements of $\mathcal{T}_h$ are denoted by t_h. For better readability, we drop the index h whenever its value is clear from the context. Since our data structures require at least one interior vertex per macro-element, we use the mesh $\mathcal{T}_h$ with $h = 2^{-2}H$ as the coarse grid in our multigrid solver hierarchy. Each of the $\mathcal{T}_h$ for $h \leq 2^{-2}H$ has some crucial properties we want to exploit. The element neighborhood at each vertex in the interior of a macro-element is always the same; cf. Figure 1. Provided that the coefficient in the PDE is constant, we can construct stencils as in a finite-difference scheme but with finite elements. Another important property is that the neighboring elements have some similarities. In 2D, there are always two elements attached to each stencil edge; see left-hand side of Figure 1. These two elements t and t^m are congruent and differ only by a reflection along the stencil edge. A similar structural property holds in 3D which we discuss later in subsection 2.4.

Associated with $\mathcal{T}_h$ is the space $V_h \subset V$ of piecewise linear finite elements. Let $\boldsymbol{e}_i \in \mathbb{R}^d$ be the canonical unit vector with $(\boldsymbol{e}_i)_j = \delta_{ij}$ for $1 \leq i, j \leq d$, where δ_{ij} is the Kronecker delta. Let further $\phi_i \in V_h$ and $\phi_j \in V_h$ be the scalar-valued linear nodal basis functions associated with the ith and jth mesh nodes. Denote by $\mathbf{v}_h = \sum_i \mathbf{v}^{(j)} \phi_j$ and $\mathbf{w}_h = \sum_j \mathbf{w}^{(i)} \phi_i$ linear combinations of the nodal basis function with vector-valued coefficients $\mathbf{v}^{(i)} \in \mathbb{R}^d$ and $\mathbf{w}^{(j)} \in \mathbb{R}^d$. We split the bilinear form (2.2) in terms of contributions of the bilinear form a^T restricted to each macro-element $T \in \mathcal{T}_H$, i.e.,

$$
\begin{aligned}
a(\mathbf{v}_h, \mathbf{w}_h) &= \sum_{T \in \mathcal{T}_H} a^T(\mathbf{v}_h, \mathbf{w}_h) = \sum_{T \in \mathcal{T}_H} \sum_{i,j} a^T(\mathbf{v}^{(j)} \phi_j, \mathbf{w}^{(i)} \phi_i) \\
&= \sum_{T \in \mathcal{T}_H} \sum_{i,j} \sum_{l,m=1}^{d} (\mathbf{v}^{(j)})_l (\mathbf{w}^{(i)})_m \, a^T(\phi_j \boldsymbol{e}_l, \phi_i \boldsymbol{e}_m).
\end{aligned} \tag{2.3}
$$

In order to simplify notation, we introduce the operator D in place of either differential operator, i.e., $D\mathbf{u} = \epsilon(\mathbf{u})$ or $D\mathbf{u} = \boldsymbol{\nabla} \cdot \mathbf{u}$, and the coefficient placeholder k, i.e., $k = \mu$ or $k = \lambda$. For the rest of this subsection, we restrict ourselves to the general bilinear

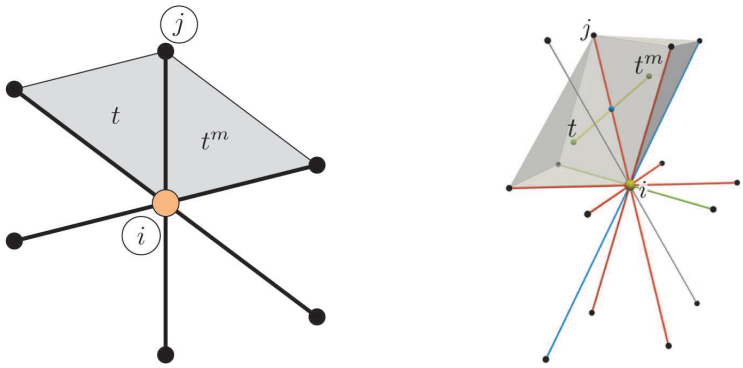

FIG. 1. *Element t and reflected element t^m along an edge between nodes i and j in the 2D case (left) and the 3D case (right).*

form

$$a(\mathbf{u},\mathbf{v}) = \langle k \cdot D(\mathbf{u}), D(\mathbf{v})\rangle_\Omega .$$

Using the standard finite element approach, this bilinear form is usually discretized in the following way. Let i_t and j_t be the local indices of an element $t \in \mathcal{T}_h$ associated with the global mesh nodes i and j. We denote by k^t the arithmetic mean over all the vertex coefficient values of an element t, i.e.,

$$\bar{k}^t = \frac{\sum_{p=1}^{d+1} k(\boldsymbol{x}_p^t)}{d+1}, \tag{2.4}$$

where $\boldsymbol{x}_p^t$ are the vertex coordinates of the local element t. Let ϕ_i^t and ϕ_j^t be the local scalar-valued linear nodal basis functions associated with the local vertices i_t and j_t. Because the derivatives of the linear basis functions are constant, employing (2.4) as a quadrature rule to approximate the bilinear form (2.3) yields

$$a_h(\mathbf{v}_h,\mathbf{w}_h) = \sum_{T\in\mathcal{T}_H}\sum_{i,j}\sum_{l,m=1}^{d} (\mathbf{v}^{(j)})_l \cdot (\mathbf{w}^{(i)})_m \sum_{t\in\mathcal{T}_h^{i,j;T}} \bar{k}^t \int_t (D(\phi_j^t \boldsymbol{e}_l), D(\phi_i^t \boldsymbol{e}_m))\,\mathrm{d}x, \tag{2.5}$$

where $\mathcal{T}_h^{i,j;T}$ is the set of all elements within a macro-element T adjacent to the edge through i and j. Note that the number of elements in this set is small due to the HHG structure. In 2D, there are only two elements adjacent to an interior edge, and in 3D there are two types of interior edges: one type with four elements and another with six elements adjacent to it. Please refer to subsection 2.4 for more details. Throughout the paper, we denote the bilinear form $a_h(\cdot,\cdot)$ defined in (2.5) by *nodal integration*. We note that the choice of $\bar{k}^t$ is natural in the case when the coefficient function is stored at the nodes. Alternative definitions, such as the evaluation of the coefficient function at the center of an element, are also suitable, and they are preferred if the coefficient function must be evaluated analytically.

With these considerations in mind, we define the reference stencil $\hat{S}_{ij}^T \in (\mathbb{R}^{d\times d})$

for $T \in \mathcal{T}_H$ as a $d \times d$ matrix for every pair of mesh nodes i and j by

$$\left(\hat{S}_{ij}^T\right)_{lm} = \int_T (D(\phi_j \boldsymbol{e}_l), D(\phi_i \boldsymbol{e}_m))\,\mathrm{d}x.$$

See Figure 1 for an illustration of stencils in 2D and 3D in the interior of a macro-element. Each edge in the stencil plots corresponds to a neighbor j of a central entry i in the mesh $\mathcal{T}_h$. Recall that, as described in the construction of HHGs, the structure in the interior of a single macro-element is always the same. It is also interesting to note that in [4] each stencil weight consists of a scalar real value, but here, each stencil weight consists of an $\mathbb{R}^{d\times d}$ matrix corresponding to the interaction between the dimensional components.

If $k \equiv 1$, the integrals in (2.5) may be replaced by the corresponding reference stencils, and the bilinear forms (2.5) and (2.3) are equal on the discrete space $V_h \times V_h$. The following lemma presents a decomposition of the bilinear form (2.5), which is better suited for matrix-free methods because it has a lower operational count while requiring a comparable amount of memory traffic. This decomposition is very similar to a decomposition of the displacement or velocity field into a symmetric strain rate part and an antisymmetric rotational part.

LEMMA 2.1. *Under the assumption that the coefficient k is affine linear on each local element patch $\omega_{i,j;T} = \bigcup_{t\in\mathcal{T}_h^{i,j;T}} \bar{t}$, the bilinear form* (2.5) *may be decomposed into a symmetric part with a scaled reference stencil and a remaining antisymmetric correction term R,*

$$\hat{a}_h(\mathbf{v}_h, \mathbf{w}_h) = \sum_{T\in\mathcal{T}_H} \sum_{i,j} \sum_{l,m=1}^{d} \left(\hat{k}_{ij}^T \cdot (\hat{S}_{ij}^T)_{lm} + \left(R^T(k)_{ij}\right)_{lm}\right) \cdot (\mathbf{v}^{(j)})_l (\mathbf{w}^{(i)})_m, \tag{2.6}$$

where $\hat{k}_{ij}^T$ is specified as in (2.8).

Proof. Let the local stiffness tensor of a local element t be given by

$$\left(a_{ij}^t\right)_{lm} = \int_t \left(D\left(\phi_j^t \boldsymbol{e}_l\right), D\left(\phi_i^t \boldsymbol{e}_m\right)\right)\,\mathrm{d}x.$$

In the following, we assume that $i \neq j$ and that k is linear on the patch $\omega_{i,j;T}$. Additionally, we introduce the symmetric part $a_{ij}^{s;t}$ and the antisymmetric part $a_{ij}^{a;t}$ of a_{ij}^t defined by

$$a_{ij}^{s;t} = \frac{1}{2}\left(a_{ij}^t + \left(a_{ij}^t\right)^\top\right) \quad \text{and} \quad a_{ij}^{a;t} = \frac{1}{2}\left(a_{ij}^t - \left(a_{ij}^t\right)^\top\right). \tag{2.7}$$

Due to our mesh structure, for each t in the interior of T, there exists a reflected element t^m; cf. Figure 1. Exploiting the fact that $\nabla\phi_i^t = -\nabla\phi_j^{t_m}$, one can show that the local stiffness tensors of these elements are related in the following way:

$$a_{ij}^{s;t} = a_{ij}^{s;t^m} \quad \text{and} \quad a_{ij}^{a;t} = -a_{ij}^{a;t^m}.$$

Before proceeding, we define the arithmetic mean of the coefficients on the patch $\omega_{i,j;T}$ as

$$\hat{k}_{ij}^T = \frac{1}{|\mathcal{T}_h^{i,j;T}|} \sum_{t\in\mathcal{T}_h^{i,j;T}} \bar{k}^t, \tag{2.8}$$

where $|\mathcal{T}_h^{i,j;T}|$ stands for the number of elements in $\mathcal{T}_h^{i,j;T}$. Using these properties, we can rewrite the last sum in (2.5) as

$$\begin{aligned}\sum_{t\in\mathcal{T}_h^{i,j;T}} \bar{k}^t (a_{ij}^t)_{lm} &= \frac{1}{2}\sum_{t\in\mathcal{T}_h^{i,j;T}} \bar{k}^t (a_{ij}^t)_{lm} + \bar{k}^{t^m}(a_{ij}^{t^m})_{lm} \\ &= \frac{1}{2}\sum_{t\in\mathcal{T}_h^{i,j;T}} \bar{k}^t (a_{ij}^{s;t})_{lm} + \bar{k}^t (a_{ij}^{a;t})_{lm} + \bar{k}^{t^m}(a_{ij}^{s;t^m})_{lm} + \bar{k}^{t^m}(a_{ij}^{a;t^m})_{lm} \\ &= \frac{1}{2}\sum_{t\in\mathcal{T}_h^{i,j;T}} (\bar{k}^t + \bar{k}^{t^m})(a_{ij}^{s;t})_{lm} + (\bar{k}^t - \bar{k}^{t^m})(a_{ij}^{a;t})_{lm} \\ &= \hat{k}_{ij}^T \cdot \hat{S}_{ij} + \frac{1}{2}\sum_{t\in\mathcal{T}_h^{i,j;T}} (\bar{k}^t - \bar{k}^{t^m})(a_{ij}^{a;t})_{lm}.\end{aligned}$$

In the last step, we exploited the facts that for an affine linear k, we have $\bar{k}^t + \bar{k}^{t^m} = 2k(\frac{x_i+x_j}{2}) = 2\hat{k}_{ij}^T$, and that

$$\sum_{t\in\mathcal{T}_h^{i,j;T}} (a_{ij}^{s;t})_{lm} = (\hat{S}_{ij}^T)_{lm}.$$

With these considerations in mind, we define the tensor $R^T(k)$ for each i and j by

$$\big(R^T(k)\big)_{ij} = \frac{1}{2}\sum_{t\in\mathcal{T}_h^{i,j;T}} (\bar{k}^t - \bar{k}^{t^m}) a_{ij}^{a;t}. \tag{2.9}$$

In the case when $i = j$, we set the correction term $\big(R^T(k)\big)_{ii}$ to zero and the scaling term $\hat{k}_{ii}^T$ to 1 and redefine the central stencil entry as

$$S_{ii}^T = -\sum_{j\neq i} \hat{k}_{ij}^T \cdot \hat{S}_{ij}^T + \big(R^T(k)\big)_{ij}.$$

This zero-row sum property ensures that translational body motions lie in the kernel of the discrete operator induced by (2.6). ∎

In addition to the bilinear form (2.6), we define the following form where the correction term R has been omitted:

$$\tilde{a}_h(\mathbf{v}_h, \mathbf{w}_h) = \sum_{T\in\mathcal{T}_H}\sum_{i,j}\sum_{l,m=1}^{d} \hat{k}_{ij}^T \cdot (\hat{S}_{ij}^T)_{lm} (\mathbf{v}^{(j)})_l (\mathbf{w}^{(i)})_m. \tag{2.10}$$

Henceforth, we refer to the bilinear form (2.6) as *physical scaling* and to the form (2.10) as *unphysical scaling*.

2.1. Interpretation of the unphysical scaling in 2D. It may be shown that the bilinear form corresponding to the unphysical form belongs to a different PDE. We illustrate this for the differential operator $-\boldsymbol{\nabla}\cdot(k\,\boldsymbol{\varepsilon}(\mathbf{u}))$ in 2D. Particularly, in 2D, a straightforward computation shows, for a differentiable coefficient k and smooth $\mathbf{u}$, that the identity

$$\boldsymbol{\nabla}\cdot(k\,(\boldsymbol{\nabla}\cdot\mathbf{u})I) = \boldsymbol{\nabla}\cdot(k\,\boldsymbol{\nabla}\mathbf{u}^\top) + \begin{pmatrix} (u_2)_{,y} & -(u_2)_{,x} \\ -(u_1)_{,y} & (u_1)_{,x}\end{pmatrix}\boldsymbol{\nabla}k = \boldsymbol{\nabla}\cdot(k\,\boldsymbol{\nabla}\mathbf{u}^\top) + (\boldsymbol{\nabla}\times O\mathbf{u})\boldsymbol{\nabla}k$$

holds true, with the identity matrix I and

$$\boldsymbol{\nabla} \times \mathbf{w} = \begin{pmatrix} (w_1)_{,y} & -(w_1)_{,x} \\ (w_2)_{,y} & -(w_2)_{,x} \end{pmatrix} \text{ and } O\mathbf{w} = \begin{pmatrix} w_2 \\ -w_1 \end{pmatrix}.$$

Using the above identity, we find

$$\begin{aligned} -\boldsymbol{\nabla} \cdot (k\, \boldsymbol{\varepsilon}(\mathbf{u})) &= -\frac{1}{2}\boldsymbol{\nabla} \cdot (k\, \boldsymbol{\nabla}\mathbf{u}) - \frac{1}{4}\boldsymbol{\nabla} \cdot (k\, \boldsymbol{\nabla}\mathbf{u}^\top) - \frac{1}{4}\boldsymbol{\nabla} \cdot (k\, \boldsymbol{\nabla}\mathbf{u}^\top) \\ &= \underbrace{-\frac{1}{2}\boldsymbol{\nabla} \cdot (k\, \boldsymbol{\nabla}\mathbf{u}) - \frac{1}{4}\boldsymbol{\nabla} \cdot (k\, (\boldsymbol{\nabla} \cdot \mathbf{u})I) - \frac{1}{4}\boldsymbol{\nabla} \cdot (k\, \boldsymbol{\nabla}\mathbf{u}^\top)}_{-\nabla \cdot \mathsf{A}(\mathbf{u})} + \underbrace{\frac{1}{4}(\boldsymbol{\nabla} \times O\mathbf{u})\boldsymbol{\nabla} k}_{\mathsf{B}(\mathbf{u})} \\ &= -\boldsymbol{\nabla} \cdot \mathsf{A}(\mathbf{u}) + \mathsf{B}(\mathbf{u}). \end{aligned}$$

The first term $\boldsymbol{\nabla} \cdot \mathsf{A}(\mathbf{u})$ corresponds to a second-order differential operator for which we have

$$\mathsf{A}\begin{pmatrix} u \\ 0 \end{pmatrix} : \boldsymbol{\nabla}\begin{pmatrix} 0 \\ v \end{pmatrix} = \mathsf{A}\begin{pmatrix} 0 \\ u \end{pmatrix} : \boldsymbol{\nabla}\begin{pmatrix} v \\ 0 \end{pmatrix}.$$

This property guarantees that the antisymmetric part of the associated local stencil term is zero, and thus it can be simply scaled. The second term $\mathsf{B}(\mathbf{u})$ is obviously equal to zero if k is constant; otherwise, it corresponds to a first-order differential operator. Here we find

$$\begin{pmatrix} v & 0 \end{pmatrix} \mathsf{B}\begin{pmatrix} 0 \\ u \end{pmatrix} = -\begin{pmatrix} 0 & v \end{pmatrix} \mathsf{B}\begin{pmatrix} u \\ 0 \end{pmatrix} \text{ and } \begin{pmatrix} v & 0 \end{pmatrix} \mathsf{B}\begin{pmatrix} u \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & v \end{pmatrix} \mathsf{B}\begin{pmatrix} 0 \\ u \end{pmatrix} = 0,$$

which yields that the symmetric part of the associated local stencil term is zero. A more detailed comparison shows that it corresponds to the antisymmetric correction term R. Therefore, omitting the correction term R in (2.6) with $D = \boldsymbol{\varepsilon}$ does not result in a discretization of the PDE $-\boldsymbol{\nabla} \cdot (k\, \boldsymbol{\varepsilon}(\mathbf{u})) = \mathbf{f}$, but of $-\boldsymbol{\nabla} \cdot (\mathsf{A}(\mathbf{u})) = \mathbf{f}$, for a general k. Only in the case when k is constant in Ω does the correction term vanish and is the original PDE recovered. We note that the splitting of the differential operator in the terms associated with the operators A and B corresponds exactly to the splitting of the stencil entries in symmetric and antisymmetric parts. Similar considerations can be worked out in 3D.

2.2. Stencil-based matrix-free methods. These newly introduced bilinear forms are very well suited for stencil-based matrix-free methods on HHGs, since the reference stencil and the correction terms are always the same for a single macro-element, and only the scaling terms depending on the coefficient need to be recomputed. In section 4, we present a short analysis of the computational cost of the standard approach by nodal integration compared to the scaling-based approaches.

In Lemma 2.1, we assume that the coefficient k is affine linear on each local patch of elements adjacent to an edge. Therefore, if k is a global affine linear function, both bilinear forms $a_h(\cdot,\cdot)$ and $\hat{a}_h(\cdot,\cdot)$ are equal. Since we use linear finite elements, optimal convergence rates may still be observed if a linear local interpolant of a nonlinear k is used. The unphysical scaling $\tilde{a}_h(\cdot,\cdot)$, however, is only equal to the other bilinear forms when the coefficient k is constant on the whole domain.

Remark 2.2. The definition in (2.8) may be replaced by $\hat{k}_{ij}^T = \frac{1}{2}\,(k(\boldsymbol{x}_i) + k(\boldsymbol{x}_j))$. This approach requires fewer FLOPs, but numerical experiments suggest that using

(2.8) yields better accuracy while not having a huge impact on performance. The memory traffic for either approach is the same because the coefficients need to be loaded from memory anyway in order to compute the correction term. This assumes that the *layer condition* [19] is satisfied when traversing the HHG data structures, so the values of k need to be read from memory only once.

In practice, this scaling of the reference stencil is only done in the interior of macro-elements where, asymptotically, most computations are performed in order to evaluate the bilinear form. The physically scaled form $\hat{a}_h(\cdot,\cdot)$ is thus redefined as

$$\hat{a}_h(\phi_j \boldsymbol{e}_l, \phi_i \boldsymbol{e}_m) = \begin{cases} a_h(\phi_j \boldsymbol{e}_l, \phi_i \boldsymbol{e}_m) & \text{if } x_i \in \partial T \text{ and } x_j \in \partial T \text{ of at least one } T \in \mathcal{T}_H, \\ \hat{a}_h(\phi_j \boldsymbol{e}_l, \phi_i \boldsymbol{e}_m) & \text{otherwise.} \end{cases}$$

This definition enforces global symmetry of the matrix but requires taking into account special boundary cases when iterating over the interior of macro-elements. In practice, we therefore employ an alternative definition where we use the standard bilinear form only if $x_i \in \partial T$ of at least one $T \in \mathcal{T}_H$. This loss of global symmetry across macro-element interfaces may cause problems for iterative solvers relying on symmetric matrices. However, this symmetry loss can be regarded as higher-order perturbation and in the numerical experiments provided in section 5, no degradation of the convergence of the employed iterative solvers could be observed.

In the following two subsections, we show how to efficiently precompute most parts of the correction term (2.9) in order for them to be suitable for stencil-based codes. Since the correction term depends on the space dimension, we derive it separately for 2D and 3D.

Remark 2.3. The presented idea of scaling the reference stencils and thus obtaining an accurate enough approximation of the stiffness matrix entries is only valid for linear finite elements. However, for higher-order discretizations it is still possible to exploit the HHG structure. There, we assume that the reference basis functions, their gradients, and the coefficient are evaluated and stored at the quadrature points. Using these values, the stencils are assembled similarly to those in [24]. Let $\mathcal{Q}_t$ be the set of quadrature points in an element $t \in \mathcal{T}_h$. Due to the HHG structure, for each $q_t \in \mathcal{Q}_t$ there exists a reflected quadrature point $q_{t^m} \in \mathcal{Q}_{t^m}$. For each pair of reflected elements t and t^m attached to an edge between two nodes x_i and x_j, we need to store the matrices $(E_{q_t})_{lm} = (D(\phi_j(x_{q_t})\boldsymbol{e}_l), D(\phi_i(x_{q_t})\boldsymbol{e}_m))$ for $q_t \in \mathcal{Q}_t$. Let $\mathcal{T}_h^{i,j;T;m}$ be the set of half of the elements within a macro-element T adjacent to the edge through i and j which have a unique corresponding reflected element which is not in the set. Additionally, let ω_{q_t} be the quadrature weights corresponding to the quadrature points in $\mathcal{Q}_t$. The stencil S_{ij}^T may then by computed via

$$S_{ij}^T = \sum_{t \in \mathcal{T}_h^{i,j;T;m}} \sum_{q_t \in \mathcal{Q}_t} |t| \omega_{q_t} \left(k_{q_t} + k_{q_{t^m}}\right) E_{q_t}.$$

2.3. Correction term in 2D. In this subsection, we consider the correction term (2.9) in the case of two dimensions, i.e., $d = 2$, and present a closed form of its values. The antisymmetric part $a_{ij}^{a;t}$ of a_{ij}^t defined through (2.7) is determined by a single variable $\gamma^{(i,j);t}$ and is of the following form:

$$a_{ij}^{a;t} = \begin{pmatrix} 0 & -\gamma^{(i,j);t} \\ \gamma^{(i,j);t} & 0 \end{pmatrix}.$$

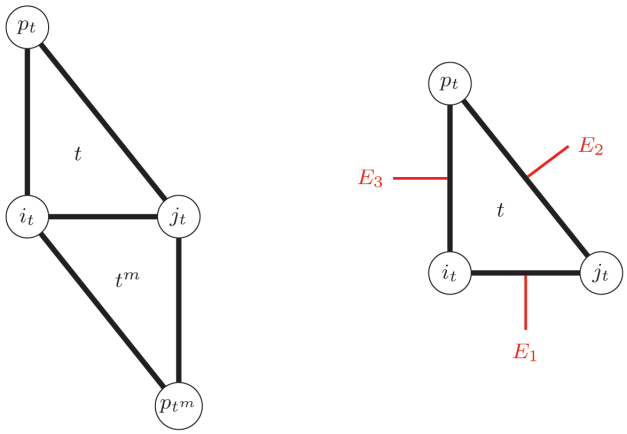

FIG. 2. *Local indices of an element t and its corresponding reflected element t^m (left). An element t with the three edges (right).*

Let $\mathbf{n}(x) = (n_1(x), n_2(x))^\top$ be the outward-pointing unit-normal of an element $t \in \mathcal{T}_h$ for $x \in \partial t$, and let $\boldsymbol{\tau}(x) = (-n_2(x), n_1(x))^\top$ be the corresponding tangential vector. In the following, we assume that the differential operator D is given by $D\mathbf{u} = \boldsymbol{\varepsilon}(\mathbf{u})$. Additionally, in the constant coefficient reference case, we have $k = 1$. Doing the same computations with $D\mathbf{u} = \boldsymbol{\nabla} \cdot \mathbf{u}$ results in the same values but now just with a flipped sign. The value $\gamma^{(i,j);t}$ can be rewritten as

$$\gamma^{(i,j);t} = \int_t \boldsymbol{\varepsilon}(\phi_j \boldsymbol{e}_1) : \boldsymbol{\varepsilon}(\phi_i \boldsymbol{e}_2) - \boldsymbol{\varepsilon}(\phi_j \boldsymbol{e}_2) : \boldsymbol{\varepsilon}(\phi_i \boldsymbol{e}_1) \, \mathrm{d}x = \frac{1}{2} \int_t \phi_{j,y}\phi_{i,x} - \phi_{j,x}\phi_{i,y} \, \mathrm{d}x$$

$$= \frac{1}{2} \int_{\partial t} \phi_i \left(\phi_{j,y} n_1 - \phi_{j,x} n_2\right) \mathrm{d}s = \frac{1}{2} \int_{\partial t} \phi_i \left(\phi_{j,y}\tau_2 + \phi_{j,x}\tau_1\right) \mathrm{d}s = \frac{1}{2} \int_{\partial t} \phi_i \boldsymbol{\nabla}\phi_j \cdot \boldsymbol{\tau} \, \mathrm{d}s.$$

We denote the three edges of an element $t \in \mathcal{T}_h$ by E_1, E_2, and E_3 as illustrated in Figure 2. Since $\phi_i = 0$ on E_2 and $\boldsymbol{\nabla}\phi_j \cdot \boldsymbol{\tau} = 0$ on E_3, the integral is reduced to

$$\gamma^{(i,j);t} = \frac{1}{2} \int_{\partial t} \phi_i \boldsymbol{\nabla}\phi_j \cdot \boldsymbol{\tau} \, \mathrm{d}s = \frac{1}{2} \int_{E_1} \phi_i \boldsymbol{\nabla}\phi_j \cdot \boldsymbol{\tau} \, \mathrm{d}s = \frac{1}{2|E_1|} \int_{E_1} \phi_i \, \mathrm{d}s = \frac{1}{4}.$$

This constant antisymmetric part $a_{ij}^{a;t}$ needs to be scaled by a difference of coefficients evaluated at the vertices. Using the notation from Figure 2, the difference is obtained by

$$\bar{k}^t - \bar{k}^{t^m} = \frac{1}{3} \left(k\left(\boldsymbol{x}_{p_t}\right) - k\left(\boldsymbol{x}_{p_{t^m}}\right)\right).$$

Finally, the correction term evaluates to

$$\left(R^T(k)\right)_{ij} = \frac{1}{12} \left(k\left(\boldsymbol{x}_{p_t}\right) - k\left(\boldsymbol{x}_{p_{t^m}}\right)\right) \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}. \tag{2.11}$$

Note that in the 2D case, the correction term is independent of the geometry, and thus no extra information has to be stored in memory. As can be seen in the next subsection, this is no longer the case in 3D.

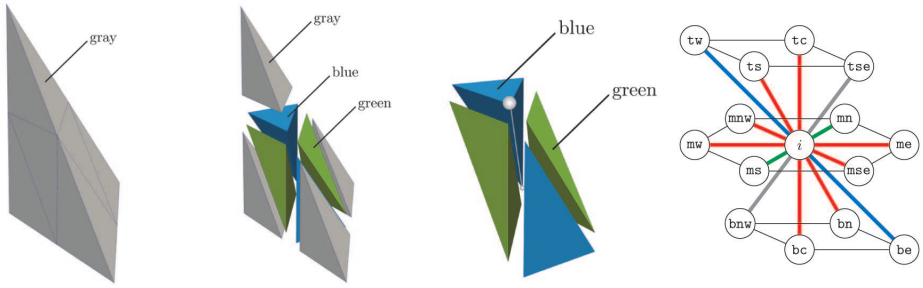

FIG. 3. *Uniform refinement of one macro-element (left) into three subclasses (second from left). Gray edge adjacent to blue and green subtetrahedra (third from left). Stencil at an interior node i of a macro-tetrahedron with off-center nodes $j \in \{\texttt{me}, \texttt{mnw}, \texttt{mn}, \texttt{ts}, \texttt{tse}, \texttt{tw}, \texttt{tc}, \texttt{bc}, \texttt{be}, \texttt{bnw}, \texttt{bn}, \texttt{ms}, \texttt{mse}, \texttt{mw}\}$ (right).*

2.4. Correction term in 3D. In the 3D case, the uniform grid refinement rule following [8] yields three subclasses of tetrahedra for each macro-element. We denote each of these classes by a color, namely gray, blue, and green; cf. left and second from left in Figure 3. We always associate the class corresponding to the macro-element to the gray color. The remaining classes are arbitrarily associated to the colors blue and green. This uniform refinement results in a stencil, which is the same for each interior node of a macro-element. The resulting stencil is illustrated on the right in Figure 3. We denote the edges adjacent to elements of classes blue and green only as edges of gray type; cf. third from left in Figure 3. The edges of green and gray types are defined similarly: An edge of blue type is adjacent only to elements of classes green and gray, whereas an edge of green type is adjacent only to elements of classes blue and gray. All other remaining edges are denoted as red-type edges.

In contrast to the 2D case, the general structure of the antisymmetric part of the local stiffness tensor $a^{a;t}$ as defined in (2.7) for an element $t \in \mathcal{T}_h^{i,j;T}$ in 3D is determined by three independent values γ, β, and δ:

$$a_{ij}^{a;t} = \begin{pmatrix} 0 & -\gamma^{(i,j);t} & -\beta^{(i,j);t} \\ \gamma^{(i,j);t} & 0 & -\delta^{(i,j);t} \\ \beta^{(i,j);t} & \delta^{(i,j);t} & 0 \end{pmatrix}. \tag{2.12}$$

Let $\mathbf{n}(x) = (n_1(x), n_2(x), n_3(x))^\top$ be the outward-pointing unit-normal of an element $t \in \mathcal{T}_h$ for $x \in \partial t$. In the case of $D\mathbf{u} = \boldsymbol{\varepsilon}(\mathbf{u})$ and $k = 1$, the nonzero components of (2.12) evaluate to

$$\begin{aligned} \beta^{(i,j);t} &= \int_t \boldsymbol{\varepsilon}(\phi_j \boldsymbol{e}_1) : \boldsymbol{\varepsilon}(\phi_i \boldsymbol{e}_3) - \boldsymbol{\varepsilon}(\phi_j \boldsymbol{e}_3) : \boldsymbol{\varepsilon}(\phi_i \boldsymbol{e}_1) \, \mathrm{d}x = \frac{1}{2} \int_t \phi_{j,z}\phi_{i,x} - \phi_{j,x}\phi_{i,z} \, \mathrm{d}x \\ &= -\frac{1}{2} \int_t \phi_{j,xz}\phi_i - \phi_{j,xz}\phi_i \, \mathrm{d}x + \frac{1}{2} \int_{\partial t} \phi_{j,z}\phi_i n_1 - \phi_{j,x}\phi_i n_3 \, \mathrm{d}s \\ &= \frac{1}{2} \int_{\partial t} \phi_i (\phi_{j,z} n_1 - \phi_{j,x} n_3) \, \mathrm{d}s. \end{aligned}$$

Similarly, the other components may be rewritten in terms of boundary integrals

$$\gamma^{(i,j);t} = \frac{1}{2} \int_{\partial t} \phi_i (\phi_{j,y} n_1 - \phi_{j,x} n_2) \, \mathrm{d}s \quad \text{and} \quad \delta^{(i,j);t} = \frac{1}{2} \int_{\partial t} \phi_i (\phi_{j,z} n_2 - \phi_{j,y} n_3) \, \mathrm{d}s.$$

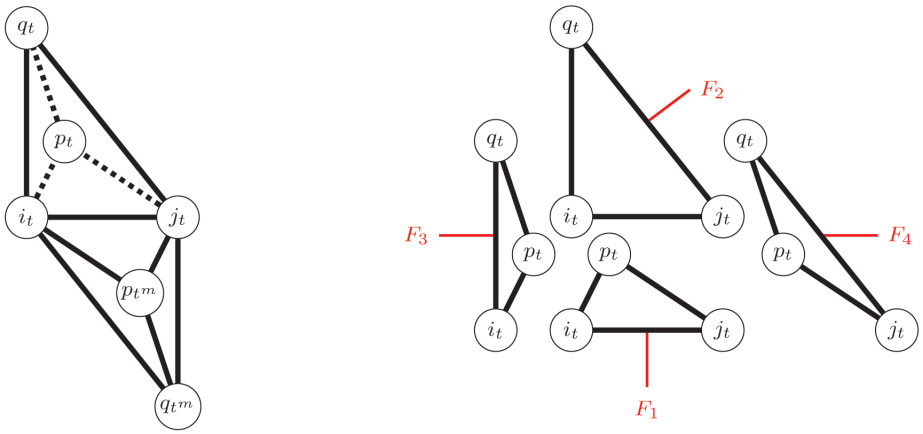

FIG. 4. *Local indices of an element t and its corresponding reflected element t^m (left). Exploded view of an element t depicting the four faces (right).*

Equivalently, as in 2D, in the case when $D\mathbf{u} = \boldsymbol{\nabla} \cdot \mathbf{u}$, the values of all three variables are the same, and only the sign is flipped.

Let i_t, j_t, p_t, and q_t be the vertex indices of a tetrahedron $t \in \mathcal{T}_h$, where i_t and j_t correspond to the global nodes i and j; see Figure 4. Additionally, the corresponding vertex coordinates of these nodes are denoted by an $\boldsymbol{x}$ with a subscript. The four faces of t are defined by the following triplets of vertices:

$$F_1 \equiv \{i_t, j_t, p_t\},\ F_2 \equiv \{i_t, j_t, q_t\},\ F_3 \equiv \{i_t, p_t, q_t\},\ \text{and } F_4 \equiv \{j_t, p_t, q_t\}.$$

Since $\phi_i = 0$ on F_4 and $(\mathbf{n} \times \boldsymbol{\nabla}\phi_j) = 0$ on F_3, applying Stokes' theorem yields

$$\begin{aligned}
\begin{pmatrix} \delta^{(i,j);t} \\ -\beta^{(i,j);t} \\ \gamma^{(i,j);t} \end{pmatrix} &= \frac{1}{2} \int_{\partial t} \phi_i \cdot (\mathbf{n} \times \boldsymbol{\nabla}\phi_j)\, \mathrm{d}s = \frac{1}{6} \sum_{f=1}^{2} \int_{F_f} \mathbf{n} \times \boldsymbol{\nabla}\phi_j \, \mathrm{d}s \\
&= \frac{1}{6} \sum_{f=1}^{2} \begin{pmatrix} \int_{F_f} \mathbf{n} \cdot (\boldsymbol{\nabla} \times \phi_j \boldsymbol{e}_1)\, \mathrm{d}s \\ \int_{F_f} \mathbf{n} \cdot (\boldsymbol{\nabla} \times \phi_j \boldsymbol{e}_2)\, \mathrm{d}s \\ \int_{F_f} \mathbf{n} \cdot (\boldsymbol{\nabla} \times \phi_j \boldsymbol{e}_3)\, \mathrm{d}s \end{pmatrix} = \frac{1}{6} \sum_{f=1}^{2} \begin{pmatrix} \int_{\partial F_f} \boldsymbol{\tau} \cdot (\phi_j \boldsymbol{e}_1)\, \mathrm{d}s \\ \int_{\partial F_f} \boldsymbol{\tau} \cdot (\phi_j \boldsymbol{e}_2)\, \mathrm{d}s \\ \int_{\partial F_f} \boldsymbol{\tau} \cdot (\phi_j \boldsymbol{e}_3)\, \mathrm{d}s \end{pmatrix} \\
&= \frac{1}{12} \left(\boldsymbol{x}_{p_t} - \boldsymbol{x}_{q_t} \right),
\end{aligned}$$

where $\boldsymbol{\tau}$ is the unit tangent. The antisymmetric part of the local stiffness tensor then reduces to

$$a_{ij}^{a;t} = \frac{1}{12} \begin{pmatrix} 0 & (\boldsymbol{x}_{q_t} - \boldsymbol{x}_{p_t})_3 & (\boldsymbol{x}_{p_t} - \boldsymbol{x}_{q_t})_2 \\ (\boldsymbol{x}_{p_t} - \boldsymbol{x}_{q_t})_3 & 0 & (\boldsymbol{x}_{q_t} - \boldsymbol{x}_{p_t})_1 \\ (\boldsymbol{x}_{q_t} - \boldsymbol{x}_{p_t})_2 & (\boldsymbol{x}_{p_t} - \boldsymbol{x}_{q_t})_1 & 0 \end{pmatrix}.$$

In Figure 5, the six elements adjacent to an edge of red type are shown. Each of these tetrahedra belongs to a class which we denote by the colors gray, green, and blue, respectively. In each color class, we have eight tetrahedra adjacent to the inner mesh node i. We define $a_{ij}^{a;t}$ to be zero if elements of the same class as t are not adjacent to an edge, which is only the case at blue-, gray-, and green-type edges. We introduce

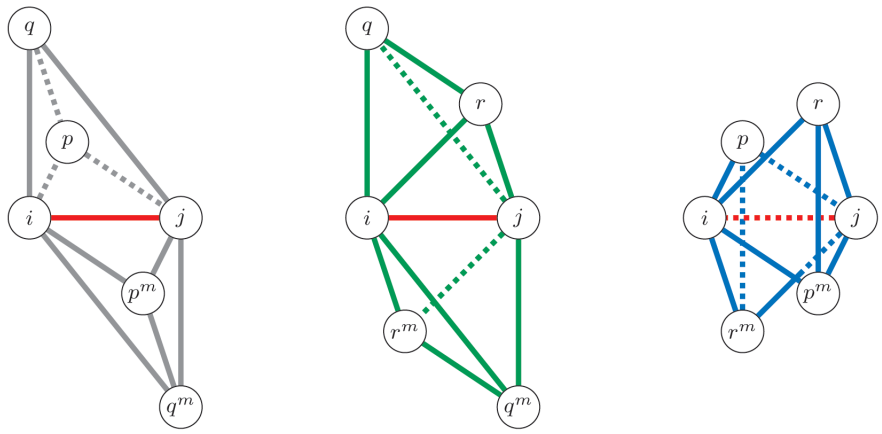

FIG. 5. *Tetrahedra adjacent to an edge of type red with local indexing of the neighboring nodes.*

a local indexing of the nodes surrounded by the edge as depicted in Figure 5 in the following considerations.

Scaling and summing over all elements adjacent to the edge through mesh nodes i and j yields

(2.13)
$$\left(R^T(k)\right)_{ij} = (\bar{k}^{t_{gray}} - \bar{k}^{t^m_{gray}})a_{ij}^{a;t_{gray}} + (\bar{k}^{t_{green}} - \bar{k}^{t^m_{green}})a_{ij}^{a;t_{green}} + (\bar{k}^{t_{blue}} - \bar{k}^{t^m_{blue}})a_{ij}^{a;t_{blue}}.$$

Since

$$\bar{k}^t - \bar{k}^{t^m} = \frac{1}{4}(k(\boldsymbol{x}_{p_t}) + k(\boldsymbol{x}_{q_t}) - k(\boldsymbol{x}_{q_{t^m}}) - k(\boldsymbol{x}_{p_{t^m}})),$$

we can rewrite (2.13) by combining terms using the index notation from Figure 5 as

$$\begin{aligned}\left(R^T(k)\right)_{ij} =& \frac{1}{4}(k(\boldsymbol{x}_p) + k(\boldsymbol{x}_q) - k(\boldsymbol{x}_{p^m}) - k(\boldsymbol{x}_{q^m}))a_{ij}^{a;t_{gray}} \\ &+ \frac{1}{4}(k(\boldsymbol{x}_q) + k(\boldsymbol{x}_r) - k(\boldsymbol{x}_{q^m}) - k(\boldsymbol{x}_{r^m}))a_{ij}^{a;t_{green}} \\ &+ \frac{1}{4}(k(\boldsymbol{x}_r) + k(\boldsymbol{x}_p) - k(\boldsymbol{x}_{r^m}) - k(\boldsymbol{x}_{p^m}))a_{ij}^{a;t_{blue}}.\end{aligned}$$

After eliminating common subexpressions, we define the three additional stencils as

$$\mathcal{S}_{ij}^{T;1} = \frac{1}{4}\left(a_{ij}^{a;t_{gray}} + a_{ij}^{a;t_{blue}}\right),\ \mathcal{S}_{ij}^{T;2} = \frac{1}{4}\left(a_{ij}^{a;t_{gray}} + a_{ij}^{a;t_{green}}\right),\ \text{and}\ \mathcal{S}_{ij}^{T;3} = \frac{1}{4}\left(a_{ij}^{a;t_{green}} + a_{ij}^{a;t_{blue}}\right).$$

To simplify the notation, we rename the following variables according to Figure 5:

$$k_{\mathcal{S}^1}^{(1)} = k(\boldsymbol{x}_p),\ k_{\mathcal{S}^1}^{(2)} = k(\boldsymbol{x}_{p^m}),\ k_{\mathcal{S}^2}^{(1)} = k(\boldsymbol{x}_q),\ k_{\mathcal{S}^2}^{(2)} = k(\boldsymbol{x}_{q^m}),\ k_{\mathcal{S}^3}^{(1)} = k(\boldsymbol{x}_r),\ \text{and}\ k_{\mathcal{S}^3}^{(2)} = k(\boldsymbol{x}_{r^m}).$$

This yields the following form of the correction term R^T in 3D:

(2.14)
$$\left(R^T(k)\right)_{ij} = \left(k_{\mathcal{S}^1}^{(1)} - k_{\mathcal{S}^1}^{(2)}\right) \cdot \mathcal{S}_{ij}^{T;1} + \left(k_{\mathcal{S}^2}^{(1)} - k_{\mathcal{S}^2}^{(2)}\right) \cdot \mathcal{S}_{ij}^{T;2} + \left(k_{\mathcal{S}^3}^{(1)} - k_{\mathcal{S}^3}^{(2)}\right) \cdot \mathcal{S}_{ij}^{T;3}.$$

Ultimately, in addition to the constant reference stencil $\hat{S}^T$, we need to store three stencils $\mathcal{S}^{T;1}$, $\mathcal{S}^{T;2}$, and $\mathcal{S}^{T;3}$ per macro-element in memory. Each of these stencils needs to be scaled appropriately to obtain a computationally cheaper approximation of the bilinear form (2.5).

3. Mapping piecewise constant coefficients to nodal values. In many physical applications, the coefficient typically depends on the strain rate $|D(\mathbf{u})|^2$ of the velocity $\mathbf{u}$ and is therefore a constant on each element when using a linear finite element discretization. Since we rely on a stencil-based implementation with coefficient values attached to the nodes in $\mathcal{T}_h$, we shall discuss efficient techniques to map piecewise constant values to nodal values. In this way, index computations that are needed to access the discrete solution can be reused to access the coefficient. Furthermore, this method is inspired by the Zienkiewicz–Zhu (ZZ) error estimator [38], where the piecewise constant gradient of a piecewise linear function is lifted to a continuous gradient. Under certain assumptions, this may improve the accuracy of the coefficient. The straightforward approach would be to find the best approximation of $|D(\mathbf{u})|^2$ in the space of piecewise linear and globally continuous functions with respect to the L^2 norm. This, however, involves solving a global linear system and thus is too costly when the coefficient changes after each iteration when solving nonlinear problems. Therefore, we refrain from a global L^2 projection and focus only on a local technique that is better suited for efficient parallel processing. One possibility is to assign to each node the volume weighted average of $|D(\mathbf{u})|^2$ over its adjacent elements. However, we present an alternative method, which we also use in our numerical experiments.

In this approach, the discrete function $\mathbf{u}$ is locally projected to an affine linear or quadratic function $\tilde{\mathbf{u}}$, and its derivative is evaluated in order to obtain an approximate value of $|D(\mathbf{u})|^2$ at a node $\boldsymbol{x}_i$. Let $\mathcal{P}_m(\omega_i)$ be the space of polynomials of order m on the patch ω_i. The jth component of $\tilde{\mathbf{u}}$ is obtained by solving the minimization problem

$$\tilde{u}_j = \operatorname*{arg\,min}_{p \in \mathcal{P}_m(\omega_i)} \sum_{i \in \mathcal{I}_i} \left(p(\boldsymbol{x}_i) - u_j(\boldsymbol{x}_i)\right)^2 \text{ for } m \in \{1, 2\}, \tag{3.1}$$

where $\mathcal{I}_i$ is the index set containing all indices of the nodal patch ω_i. Recall that in the case of a uniform refinement in 3D, this involves 15 nodes. Solving this minimization problem corresponds to solving a small least squares problem for each node $\boldsymbol{x}_i$. The approximate value of $|D(\mathbf{u})|^2$ evaluated at $\boldsymbol{x}_i$ is then given by $|D(\tilde{\mathbf{u}})(\boldsymbol{x}_i)|^2$, since by construction $D(\tilde{\mathbf{u}})$ is continuous on ω_i. As before, the coefficient k_i is obtained in a pointwise fashion according to the physical model.

In the interior of a macro-element, the quadratic and affine linear approximations are equivalent. Particularly, the quadratic minimizing polynomial p_2 of (3.1) on a patch ω_i in the interior of a macro-element may be written as $p_2(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{x}_i)^\top \mathsf{A}(\boldsymbol{x} - \boldsymbol{x}_i) + \boldsymbol{b}^\top(\boldsymbol{x} - \boldsymbol{x}_i) + c$ for some $\mathsf{A} \in \mathbb{R}^{d \times d}$, $\boldsymbol{b} \in \mathbb{R}^d$, and $c \in \mathbb{R}$. Similarly, a minimizing affine linear function on the same ω_i may be written as $p_1(\boldsymbol{x}) = \tilde{\boldsymbol{b}}^\top(\boldsymbol{x} - \boldsymbol{x}_i) + \tilde{c}$ for some $\tilde{\boldsymbol{b}} \in \mathbb{R}^d$ and $\tilde{c} \in \mathbb{R}$. Due to the symmetry of the nodes in ω_i, the quadratic and linear parts are decoupled, and it follows that $\boldsymbol{b} = \tilde{\boldsymbol{b}}$ and $c = \tilde{c}$. The derivatives of p_2 and p_1 are given by $\nabla p_2(\boldsymbol{x}) = \mathsf{A}(\boldsymbol{x} - \boldsymbol{x}_i) + \boldsymbol{b}$ and $\nabla p_1(\boldsymbol{x}) = \tilde{\boldsymbol{b}}$. Evaluating the derivatives at $\boldsymbol{x}_i$, we obtain $\nabla p_2(\boldsymbol{x}_i) = \boldsymbol{b} = \tilde{\boldsymbol{b}} = \nabla p_1(\boldsymbol{x}_i)$. Therefore, the quadratic approximation is only required on the lower-dimensional primitives, and the computationally much cheaper affine linear approximation may be used in the interior of macro-elements.

4. Computational cost analysis. Since the stencil scaling approach for vector-valued PDEs has been introduced as a means of reducing the computational cost for

matrix-free finite element implementations, we will present a concise cost analysis. Asymptotically, most of the computational work is done in the interior of macro-elements. Therefore, we restrict our performance analysis to the interior of a single macro-element. Furthermore, we ignore all performance impacts stemming from the required communication between processes and focus our analysis on multiple independent processes on a single compute node. We start with an estimation of the number of required operations to compute the residual $\mathsf{y} = \mathsf{f} - \mathsf{A}\mathsf{x}$, where the matrix A results from a discretization of the vector-valued PDEs with a single scalar coefficient. Additionally, theoretical estimates on the required memory and the memory traffic are given, which are validated by experimental measurements in subsection 5.1.1.

4.1. Number of operations. We start by counting the number of required operations to compute the residual $\mathsf{y} = \mathsf{f} - \mathsf{A}\mathsf{x}$ when using the presented method or when storing all the stencils in memory, which corresponds to storing the global matrix A. In the case of nodal integration, we assume that the local stiffness matrices (two in 2D and six in 3D) are precomputed and stored in memory. In the scaling approach, we assume that the reference stencils and the additional correction stencils are stored in memory instead. We do not precompute and store values like the average of k in (2.8) or the differences of k in (2.14). Since the stencil code is already memory bound, storing only the nodal values of k and computing the averages and differences on the fly compared to reading the precomputed values from memory increases the arithmetic intensity and reduces the required memory and the pressure on the memory buses. The required numbers of operations for the different methods are summarized in Table 2. Please note that these numbers only give estimates on the actual number of instructions performed by the processor since optimizing compilers may reorder, fuse, and vectorize FLOPs, meaning that multiple FLOPs may be performed in a single cycle. We also ignore the effect of fused multiply-add operations that are typical for most modern CPU architectures.

Let $\mathsf{y}_i \in \mathbb{R}^d$ be the target vector components at position i, let $\mathsf{x}_i \in \mathbb{R}^d$ be the input vector components at position i, and let $\mathsf{f}_i \in \mathbb{R}^d$ be the components of the right-hand-side vector at position i. Furthermore, let $S_{ij} \in \mathbb{R}^{d\times d}$ be the stencil which acts on a vector at position j in order to obtain the result at position i. The residual for all the degrees of freedom (DoFs) at position i is computed via

$$\mathsf{y}_i = \mathsf{f}_i - \sum_j S_{ij}\mathsf{x}_j. \tag{4.1}$$

Assuming that all the S_{ij} for a fixed i are already computed, the number of FLOPs for evaluating (4.1) is the same for all three approaches, as can be seen in the fifth column of Table 2. In 2D, there are seven stencils S_{ij} for a fixed i, and thus seven local matrix vector multiplications have to be performed, and the results are added for a total of 26 additions and 28 multiplications. The subtraction from the right-hand side takes two extra additions. Since there are 15 stencils in 3D, similar considerations yield that $15 \cdot 9 = 135$ multiplications need to be performed. The number of additions is made up of $15 \cdot 2 \cdot 3 = 90$ additions in the matrix-vector products, $14 \cdot 3 = 42$ additions in the sum over its results, and three additions from the subtraction of the right-hand side.

In the following, we estimate the number of required operations to compute the stencil entries S_{ij} for the matrix-free variants and begin with the physical scaling case.

4.1.1. Number of operations in the physical scaling case. Recall that in the unphysical scaling case, the stencil is defined as $S_{ij} = \hat{k}_{ij}^T \hat{S}_{ij}$ for $j \neq i$. The

TABLE 2
Operation count for residual computation.

Method	Dimension	Noncentral entries	Central entry	Residual	Total
physical scaling	2D	36 add / 36 mul	24 add / 0 mul	28 add / 28 mul	152
	3D	242 add / 220 mul	123 add / 0 mul	135 add / 135 mul	855
nodal integration	2D	33 add / 48 mul	24 add / 24 mul	28 add / 28 mul	185
	3D	754 add / 648 mul	207 add / 216 mul	135 add / 135 mul	2095
stored stencils	2D	0 add / 0 mul	0 add / 0 mul	28 add / 28 mul	56
	3D	0 add / 0 mul	0 add / 0 mul	135 add / 135 mul	270

central entry for $j = i$ is defined in a way to enforce the zero-row sum property, i.e., $S_{ii} = -\sum_{j \neq i} S_{ij}$.

Computing a stencil entry for a fixed i and j with $j \neq i$ requires computing the value of $\hat{k}_{ij}^T$ and scaling the reference stencil. The calculation of $\hat{k}_{ij}^T$ requires two multiplications and three additions in 2D, and in 3D the number of operations depends on the number of elements adjacent to the edge through the nodes i and j. Since we are interested in an upper bound only, we assume the worst case of two multiplications and seven additions. Finally, due to symmetry, the scaling requires $\frac{d(d+1)}{2}$ multiplications. These values need to be multiplied by the number of off-center stencils, which results in the numbers shown in the third column of Table 2. Computing the central entry requires 12 additions in each component, totaling 24 additions in 2D. In 3D, the number of additions per component is 41, resulting in a total of 123 additions.

In order to complete the cost consideration of the physical scaling, the cost of the correction term needs to be assessed. In 2D, the correction term (2.11) just consists of the scaled difference of two coefficient values, which results in a total of six subtractions and six multiplications. Adding the correction term to the scaled reference stencil requires 12 additional additions.

For the 3D case, recall that the physically scaled stencil is defined as

$$S_{ij} = \hat{k}_{ij}^T \cdot \hat{S}_{ij} + \left(k_{\mathcal{S}^1}^{(1)} - k_{\mathcal{S}^1}^{(2)}\right) \cdot \mathcal{S}_{ij}^{T;1} + \left(k_{\mathcal{S}^2}^{(1)} - k_{\mathcal{S}^2}^{(2)}\right) \cdot \mathcal{S}_{ij}^{T;2} + \left(k_{\mathcal{S}^3}^{(1)} - k_{\mathcal{S}^3}^{(2)}\right) \cdot \mathcal{S}_{ij}^{T;3}$$

for $j \neq i$. There, we have three correction terms with three unique nonzero entries for red edges and two correction terms with three unique nonzero entries for the remaining edges which need to be scaled. The scaling term of each correction stencil requires one addition only. This leads to $3 \cdot 3 = 9$ extra multiplications and $3 + 3 \cdot 3 = 12$ additions per red-edge stencil entry. For the edges of other colors, $2 \cdot 3 = 6$ extra multiplications and $2 + 2 \cdot 3 = 8$ additions are needed. Since there are eight red edges and six other edges per stencil, the total number of operations in the third column of Table 2 is obtained.

4.1.2. Number of operations in the nodal integration case. For the number of required operations in the nodal integration case, we recall some of the calculations from the scalar case in [4]. There, the total number of operations is reduced by eliminating common subexpressions to compute the coefficient value at the quadrature point. The number of additions required to obtain the noncentral stencil entries in the scalar case is 15 in 2D and 98 in 3D. In the vector-valued case, almost all of these numbers need to be multiplied by 4 in 2D or 9 in 3D; only the sums of the coefficients are computed once per updated node. The computation of the common subexpressions in 2D requires nine additions and 16 in 3D. Since the common subexpressions of

the coefficient only need to be computed once per node, this results in a total of $4 \cdot (15-9)+9 = 33$ in 2D and $9 \cdot (98-16)+16 = 754$ in 3D. The number of operations for the multiplications is obtained by just multiplying the number of scalar operations by 4 or 9, yielding $4 \cdot 12 = 48$ in 2D and $9 \cdot 72 = 648$ in 3D. In our case, the central entries are not computed by the computationally cheaper method of enforcing the zero row-sum property because the rigid body mode kernel is preserved in this way. These values are obtained by manually counting the number of operations for the central entries, yielding the values presented in the third and fourth rows of Table 2.

4.1.3. Number of operations in the stored stencils case. In this scenario, the whole global matrix A is stored in memory. Therefore, we assume that no costs are involved in computing the stencil entries and only the operations to compute the residual are required. Note that this scenario is the preferred one with respect to the number of operations but it consumes the most memory and it has the largest impact on memory traffic from main memory; cf. subsection 4.2.

4.1.4. Comparison of total required operations. The theoretical analysis of the required operations yields estimates of how much CPU time could be saved in the case when the memory bandwidth is not limited and when the overhead stemming from index calculations is ignored. As can be seen, the savings in FLOPs are minor in 2D, but in 3D they are quite significant. For 2D, Table 2 shows that the physical scaling requires 82% of the FLOPs that are needed by the on-the-fly nodal integration. In 3D, the physical scaling requires 41% of the FLOPs needed by the nodal integration. However, as can be seen in the measurements in subsection 5.1.1, the compiler reduces the number of theoretically estimated FLOPs. Using the values reported by the Intel Advisor,[2] we see that the physical scaling requires 44% of the FLOPs needed by the nodal integration.

4.2. Memory consumption and memory access. For the best performance, it is not only required that the number of FLOPs is small. The memory traffic from the main memory also has to be small relative to the required FLOPs. Therefore, we first give a short summary on the required number of double precision variables for a residual computation in the interior of a single macro-element in Table 3, where N is the number of scalar degrees of freedom in the interior of a single macro-element. The third column summarizes the number of variables required to store the discretized functions f, x, y, and k. The fourth column summarizes the number of variables required to store the discretized operator A. Note that only for the stored stencils approach does the memory required to store the operator grow with the mesh size. The total memory footprint is worst for the stored stencils approach. In this scenario 135 extra scalar variables must be stored, a number that would alternatively permit an extra level of refinement of the mesh when using one of the matrix-free approaches. Even if storing all stencils is cheapest in terms of FLOPs, it creates a severe restriction on the size of the problems that can be solved and leads to a very large amount of data that must be transferred from the main memory in each matrix-vector product.

In Table 4, we present estimates on the average number of bytes which need to be loaded from and stored in the main memory to compute the residual at a single mesh node in 3D. We split the estimation into two extreme cases. In the optimistic case, we assume perfect caching and that all previously loaded values stay in the fast cache levels. In the pessimistic case, we assume no caching at all and that all the data have to be loaded from the slow main memory. This analysis gives lower and

[2] https://software.intel.com/intel-advisor-xe

TABLE 3
Number of double precision variables required on a single macro-element with N scalar degrees of freedom for a residual computation.

Method	Dimension	Variables (f, x, y, k)	Operators
physical scaling	2D	$7 \cdot N$	28
	3D	$10 \cdot N$	540
nodal integration	2D	$7 \cdot N$	72
	3D	$10 \cdot N$	864
stored stencils	2D	$7 \cdot N$	$28 \cdot N$
	3D	$10 \cdot N$	$135 \cdot N$

upper bounds on the required main memory traffic, and the value observed in practice will lie somewhere between these bounds. Note that stores and loads of temporary variables required for the computation of the stencil weights are not considered in these estimates. These values only present estimates for the number of bytes that must be transferred from the main memory, but they are not necessarily proportional to the time required to load and store them. In modern architectures, data are moved in terms of cache lines that may, for example, be 64 bytes large. If numerical data are stored contiguously, successive values can be accessed more efficiently from cache lines that are already loaded. Furthermore, modern CPU micro-architectures employ prefetching that can accelerate the access to regularly strided data. A detailed analysis of such effects on the speed of numerical kernels, as presented in, e.g., [2], is beyond the scope of this article. In Table 4, we present estimated values for the bytes to be transferred in the optimistic and pessimistic scenarios.

For the matrix-free variants, the precomputed stencil values or local stiffness matrices need to be loaded. In the physical scaling case, the 15 reference stencil weights for nine block operators are required, which results in a total of 1080 bytes. Additionally, the three additional correction stencils need to be loaded, resulting in 4320 bytes. In the nodal integration case, six local stiffness matrices with 16 entries each need to be loaded for each of the nine operators, resulting in 6912 bytes. In the optimistic case, these data stay in the caches and are loaded from the main memory only in the pessimistic case.

Only one coefficient has to be loaded in the optimistic case, but in the worst case all 15 coefficients adjacent to a mesh node need to be loaded from the main memory, which results in 120 bytes per mesh node. In the stored stencils approach, for each mesh node all 15 stencil weights for nine operators need to be loaded even in the optimistic case.

Additionally, the variables f, x, and y are accessed during an iteration. In the optimistic and pessimistic cases, 24 bytes of f need to be loaded from the main memory. Additionally, because of write allocation, 24 bytes from y need to be loaded before they are stored, resulting in traffic of 48 bytes. Reusing cached values of x in the optimistic case requires loading 24 bytes, but in the pessimistic case all 15 neighboring values need to be loaded, resulting in 360 bytes.

These estimates show that with poor cache reuse, the matrix-free approaches must be expected to produce even more main memory traffic than the stored stencils approach. However, when the layer condition is satisfied for the data traversal, and the caches are used efficiently, the matrix-free methods may lead to reduced main memory traffic.

TABLE 4
Average number of bytes required to load from and store to main memory when computing the residual at a mesh node in 3D assuming the usage of 64-bit double precision floating-point variables.

Method	Optimistic	Pessimistic
physical scaling	$8 + 96 = 104$	$4320 + 120 + 432 = 4872$
nodal integration	$8 + 96 = 104$	$6912 + 120 + 432 = 7464$
stored stencils	$1080 + 96 = 1176$	$1080 + 432 = 1512$

5. Numerical results and applications. In this section, we provide numerical results to illustrate the accuracy and run-time of the new scaling approaches in comparison to the assembly by nodal integration in a matrix-free framework. Throughout this section, we denote the time-to-solution by tts, and by relative tts we denote the ratio of the time-to-solution of the stencil scaling approach with respect to the nodal integration. The numerical solutions obtained by the corresponding bilinear forms $a_h(\cdot,\cdot)$ and $\hat{a}_h(\cdot,\cdot)$ are always denoted by $\mathbf{u}_h$ and $\hat{\mathbf{u}}_h$, respectively.

We use two machines to obtain the run-time measurements presented in the following subsections. Most of the measurements are conducted on the newer SuperMUC-NG system equipped with Skylake nodes. The following values are taken from [28]. Each node has two Intel Xeon Platinum 8174 processors with a nominal clock rate of 3.1 GHz. Each processor has 24 physical cores, which results in 48 cores per node. Each core has a dedicated L1 (data) cache of size 32 kB and a dedicated L2 cache of size 1024 kB. Each of the two processors has an L3 cache of size 33 MB shared across all its cores. The total main memory of 94 GB is split into equal parts across two NUMA domains with one processor each. We use the Intel 19.0 compiler, together with the Intel 2019 MPI library, and specify the compiler flags `-O3, -march=native, -xHost`.

The second machine we use for some measurements is the older SuperMUC Phase 2 system equipped with Haswell nodes. The following values are taken from [29]. Each node has two Intel Xeon E5-2697 v3 processors with a nominal clock rate of 2.6 GHz. Each processor has 14 physical cores, which results in 28 cores per node. Each core has a dedicated L1 (data) cache of size 32 kB and a dedicated L2 cache of size 256 kB. The theoretical bandwidths are 343 GB/s and 92 GB/s, respectively. The CPUs are running in cluster-on-die mode. Thus, each node represents four NUMA domains each consisting of seven cores with a separate L3 cache of size 18 MB and a theoretical bandwidth of 39 GB/s. On top of this, each NUMA domain has 16 GB of main memory with a theoretical bandwidth of 6.7 GB/s available. On this second machine, we use the Intel 18.0 compiler, together with the Intel 2018 MPI library, and specify the compiler flags `-O3, -march=native, -xHost`. Note that the serial runs using only a single compute core are not limited to running on large machines such as SuperMUC but can also be run on usual modern desktop workstations with enough memory.

All the following experiments were implemented in the HHG framework [5, 6, 7]. If not otherwise specified, we solve the linear systems by applying geometric multigrid V-cycles directly to the system until a specified relative tolerance of the norm of the residual is obtained. The transfers from a coarser to a finer grid are performed by a matrix-free linear interpolation, and the restriction is performed by the corresponding transposed matrix-free operation. As a smoother, we employ the hybrid Gauss–Seidel method, meaning that in the interior of macro-elements standard Gauss–Seidel iterations are performed. Across the interfaces not all dependencies are updated, which results in a Jacobi-like method. On the coarse grid, we perform iterations of the diagonally preconditioned conjugate gradient method up to a fixed large relative tolerance or for a fixed number of iterations in order to avoid unnecessary oversolving.

5.1. Linear elastostatics. In this subsection, we first perform benchmarks to verify the performance models from section 4, and we consider two problems in linear elasticity. The first is a benchmark problem where we have an analytical solution at hand and can compute the discretization errors directly. In the second, a more relevant problem is investigated, where an external force is applied to a metal foam.

5.1.1. Memory traffic and roofline analysis. In subsection 4.2, we presented theoretical estimates on the number of FLOPs and the memory accesses required to compute the residual of a linear system using different strategies to obtain the matrix entries. In this subsection, we verify these results experimentally using a specially designed benchmark, executed on a single compute node of SuperMUC-NG. With this benchmark, we compare the performance of the methods analyzed in subsection 4.2, i.e., physical stencil scaling, standard nodal integration, and stored stencils approach. The floating-point performance measurements were conducted using the Intel Advisor 2019 [21], and the memory traffic measurements were conducted by accessing the hardware performance counters using the Intel VTune Amplifier 2019 [22].

The benchmark computes the residual $y = f - Ax$ for a vector-valued operator A in 3D with the same sparsity pattern as the discretized linear elasticity operator. As in the theoretical analysis, we only consider the DoFs in the interiors of macro-tetrahedra. The residual computation is iterated 500 times in order to obtain an averaged value, reducing errors stemming from small fluctuations in the run-time. The benchmark is executed using 48 MPI ranks, pinned to the 48 physical cores of a single node. This is essential to avoid optimistic bandwidth values when only a single core accesses the memory. Measurements with the Intel Advisor and VTune Amplifier are carried out solely on rank 0. Moreover, all measurements are restricted to the innermost update loop, i.e., where the actual nodal updates take place. This does not influence the results since the outer loops are identical in all variants. Note that in each update, the DoFs corresponding to a single mesh node are updated all at once, i.e., three DoFs per update. We choose $L = 5$ as the refinement level, which yields $1.09 \cdot 10^6$ DoFs per macro-element. The computation involves three vector-valued variables x, y, and f where each of them requires about 8.4 MiB of storage per macro-tetrahedron. In addition to this, the scalar-valued coefficient k requires about 2.8 MiB of storage.

We assign three macro-elements to each MPI rank, which is the maximum possible for the stored stencils approach on SuperMUC-NG. In practice, the memory limit of a compute node would be reached even faster, since all the lower-dimensional primitives, the multigrid hierarchy, and the communication buffers require extra memory. Using these settings, each innermost loop is executed 500,062,500 times per MPI rank.

In Figure 6, we summarize the recorded performance results of the three approaches. In the leftmost plot of Figure 6, the FLOPs per update are shown, which are close to the theoretically estimated values from Table 2. The second-from-left plot presents the total number of transferred bytes from the main memory per update. The total memory consumption is shown in the third-from-left plot. The stored stencils approach requires almost 15 times more memory than the matrix-free approaches. At first sight, the stored stencils approach appears to be the most attractive with respect to the required operations when enough main memory is available. However, the rightmost plot shows that the physical scaling approach has a slightly lower time per update than the stored stencils approach. This is due to the large amount of data which needs to be transferred from the main memory in each update; cf. second-from-left plot in Figure 6. This means that, in fact, the caches are more efficiently used in the matrix-free approaches. Note that these measured values are also close to the

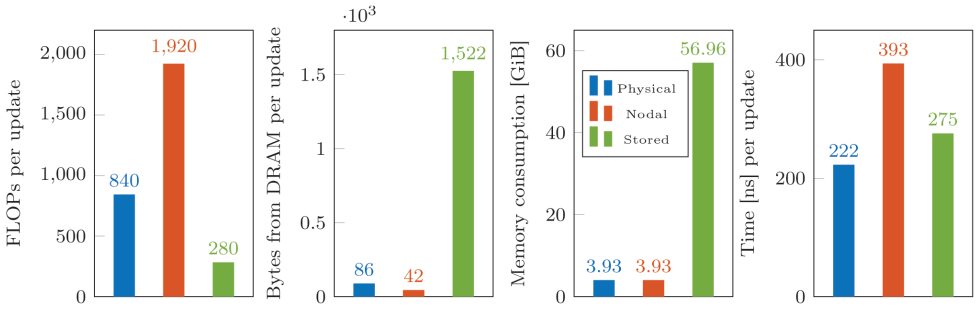

FIG. 6. *Bar plots for comparing the performance and memory consumption of the nodal integration, physical stencil scaling, and stored stencils approaches on SuperMUC-NG. Three macro-tetrahedra are assigned to each MPI rank.*

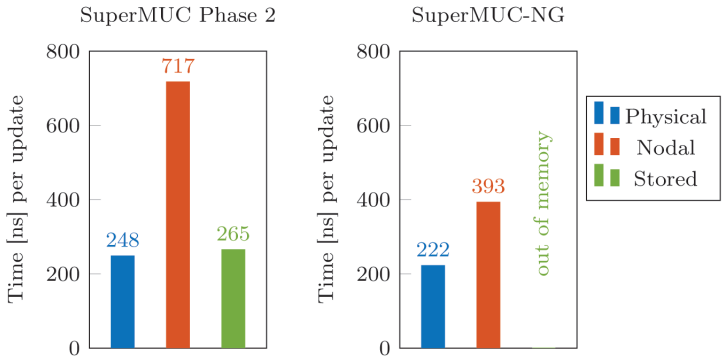

FIG. 7. *Time per update on different machines with four macro-tetrahedra attached to each MPI rank. Left: SuperMUC Phase 2. Right: SuperMUC-NG.*

theoretically estimated values reported in Table 4.

The same benchmark was conducted on SuperMUC Phase 2 but with four macro-tetrahedra assigned to each of the 28 MPI ranks. This was possible, since this machine has more memory available per core than SuperMUC-NG. In Figure 7, we contrast the time per update on SuperMUC Phase 2 with the time per update on SuperMUC-NG using equal problem sizes per MPI rank. Note that both of the matrix-free methods worked on SuperMUC-NG, but the stored stencil approach required too much memory. Furthermore, the time per update of the physical scaling did not improve much, but the time per update of the nodal on-the-fly integration was reduced by about 45%. This is due to the increased clock rate of SuperMUC-NG compared to SuperMUC Phase 2 and the larger arithmetic intensity of the on-the-fly integration.

In order to further visualize these results, we present a roofline analysis in Figure 8 conducted on SuperMUC-NG; see [20, 36]. The abscissa shows the arithmetic intensity, i.e., the number of FLOPs divided by the number of bytes loaded and stored in the innermost loop. The ordinate gives the measured performance as FLOPs performed per second. For reference, we added measured saturated memory bandwidth rooflines as reported by the Intel Advisor. Obviously, these measured values are smaller than the theoretically optimal ones given in the hardware specifications. The maximum performance for double precision vectorized fused multiply-add operations is reported

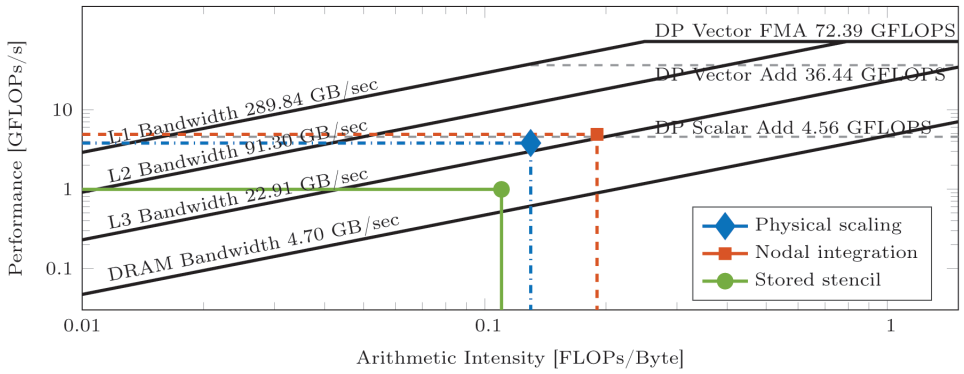

FIG. 8. *Roofline analysis of the residual computation using nodal integration, physical stencil scaling, and stored stencils approaches.*

by the Intel Advisor tool as 72.39 GFLOPs/s. From the roofline analysis one can see that the nodal integration yields the best performance with respect to FLOPs per second but is still slower in practice, since it requires more than twice the number of operations compared to the physical scaling. The physical scaling has a smaller arithmetic intensity and a slightly worse performance in GFLOPs/s, while the stored stencils approach has the lowest arithmetic intensity with the worst performance. The compiler could not autovectorize the physical scaling and stored stencils kernels. In the nodal integration kernel, however, one of the fixed-length inner loops could be autovectorized. This explains why the performance of the nodal integration is slightly better than the double precision scalar add performance. Of course, the roofline analysis constitutes only a first quantitative evaluation of the performance. Other performance models, such as the execution-cache-memory performance model [34], can give deeper insight.

Remark 5.1. As can be seen in Figure 8, the physical scaling approach reaches about 5.25% of the peak performance based on double precision vector fused multiply-add instructions. However, considering other rooflines, the physical scaling almost reaches the double precision scalar add performance and reaches about 10.43% of the double precision vector add performance. Further performance optimizations are difficult because of the less than ideal mix of multiplies and adds and the challenging vectorization due to the index calculations in tetrahedral elements. These investigations and performance optimizations are beyond the scope of this paper but are part of future work and ongoing development of software structures in HyTeG [23].

5.1.2. Linear elastostatics benchmark problem. As a first benchmark problem, we consider a compressible linear elasticity problem on the unit cube $\Omega = (0,1)^3$ modeled by (2.1) with $\Gamma_D = \partial\Omega$ and $\Gamma_N = \emptyset$. The material of the block is assumed to be isotropic and heterogeneous with a varying elastic modulus E but constant Poisson's ration ν. In this scenario, the stress tensor $\boldsymbol{\sigma} = 2\mu\boldsymbol{\varepsilon} + \lambda \operatorname{tr}(\boldsymbol{\varepsilon})I$ is given by Hooke's law, and the Lamé constants μ and λ are

$$\mu(E) = \frac{E}{2(1+\nu)} \quad \text{and} \quad \lambda(E) = \frac{\nu E}{(1+\nu)(1-2\nu)}.$$

Since the stress tensor $\boldsymbol{\sigma}$ depends linearly on E, we factor it out and rewrite the stress tensor such that it depends only on the single spatially variable coefficient E, i.e.,

$$\boldsymbol{\sigma} = E(x,y,z) \cdot \left(\frac{1}{(1+\nu)} \boldsymbol{\varepsilon} + \frac{\nu}{(1+\nu)(1-2\nu)} \mathrm{tr}(\boldsymbol{\varepsilon}) I \right).$$

The associated bilinear form in the *constant case* $E = 1$ is thus given by a linear combination of the discussed forms, yielding

$$a^{E=1}(\mathbf{u},\mathbf{v}) = \frac{1}{(1+\nu)} \left\langle \boldsymbol{\varepsilon}(\mathbf{u}), \boldsymbol{\varepsilon}(\mathbf{v}) \right\rangle_\Omega + \frac{\nu}{(1+\nu)(1-2\nu)} \left\langle \boldsymbol{\nabla} \cdot \mathbf{u}, \boldsymbol{\nabla} \cdot \mathbf{v} \right\rangle_\Omega . \tag{5.1}$$

The scaling is then performed on the bilinear form (5.1) with E as the varying scalar coefficient. In the following, we perform a quantitative comparison of the three approaches by investigating their accuracy and run-time. For this purpose, we let $\mathbf{u}^*$ be a manufactured solution and set the right-hand side $\mathbf{f}$ of (2.1) accordingly to $\mathbf{f} = -\boldsymbol{\nabla} \cdot \boldsymbol{\sigma}(\mathbf{u}^*)$. The Dirichlet boundary condition is set to $\mathbf{g} = \mathbf{u}^*\big|_{\partial\Omega}$. This allows for a direct computation of errors and a quantitative study on accuracy of the different methods. By $\mathcal{I}_h$, we denote the interpolation operator of a function on the mesh $\mathcal{T}_h$ and by $\|\cdot\|_2$ the discrete L^2 norm defined as

$$\|\mathbf{u}\|_2 = \left(h^3 \sum_{i \in \mathcal{N}_h} \|\mathbf{u}(\mathbf{x}_i)\|_2^2 \right)^{\frac{1}{2}},$$

where $\mathcal{N}_h$ is the set of all vertices in the mesh $\mathcal{T}_h$. As material parameters, we choose the Poisson's ratio of aluminum, i.e., $\nu = 0.34$, and a Young's modulus of the following: form

$$E(x,y,z) = \cos(m\,\pi\,x\,y\,z) + 2,\ m \in \{1,2,3,8\}.$$

The manufactured solution $\mathbf{u}^*$ is chosen as

$$\mathbf{u}^*(x,y,z) = \frac{1}{xyz+1} \begin{pmatrix} x^3 y + z^2 \\ x^4 y + 2z \\ 3x + yz^3 \end{pmatrix}.$$

It is important to note that the coefficient and exact solution do not lie in the ansatz spaces and therefore cannot be exactly reproduced.

We discretize the computational domain by 384 tetrahedra on the coarsest level $\ell = 0$. The finest level considered in this subsection is $L = 6$. Each system of equations is solved using a single rank on SuperMUC-NG and by employing a geometric $V(3,3)$ multigrid solver until a relative residual of 10^{-8} is obtained. As a smoother, we employ the hybrid Gauss–Seidel method, and on the coarsest level we employ a diagonally preconditioned conjugate gradient method, since the problem is symmetric and positive definite.

In Table 5, we report on the errors, convergence rates, number of required V-cycle iterations, and run-times for different refinement levels ℓ and coefficient parameters m. The error on level ℓ is defined as $\|\mathcal{I}_{h_L}\mathbf{u}^* - \mathcal{I}_{h_L}\mathbf{v}_{h_\ell}\|_2$, where $\mathbf{v}_{h_\ell}$ denotes the numerical solution obtained with one of the two matrix-free approaches, i.e., $\mathbf{u}_h$ and $\hat{\mathbf{u}}_h$. We do not consider the stored stencil approach in this comparison, since the problem sizes on the finest level are too large, and the matrices could not be stored in memory. We observe quadratic convergence in the discrete L^2 norm for the assembly through

nodal integration and the physical scaling. On the last level $L = 6$, the convergence rate is higher, since here we compare two discrete approximations on the same level. We still keep these rows in the table for completeness and in order to compare the relative tts even if the error is not directly comparable to the errors on the coarser levels. Independent of the coefficient frequency, we observe a relative tts of about 61% for $m \in \{1, 2, 3, 8\}$.

In order to emphasize that using the unphysical scaling results in a wrong solution, we computed the errors in this benchmark using the unphysical scaling (2.10) without reporting them in Table 5. In this case the errors on level $\ell = 6$ were $3.95 \cdot 10^{-3}$ for $m = 1$, $1.25 \cdot 10^{-2}$ for $m = 2$, $1.83 \cdot 10^{-2}$ for $m = 3$, and $2.32 \cdot 10^{-3}$ for $m = 8$.

For a performance comparison, we performed the same experiment on a compute node of the older SuperMUC Phase 2. The number of V-cycles and errors are the same as on SuperMUC-NG, and thus we only present the tts for both matrix-free approaches in Table 6. On this machine, we observe a relative tts of about 45% independent of the coefficient frequency. This larger speedup is due to the lower clock rate of the processor, which has already been discussed in subsection 5.1.1 and illustrated in Figure 7.

5.1.3. Linear elastostatics with external forces. In this subsection, we present an application of our scaling approach where an external force is applied to an isotropic and heterogeneous material. As before, we consider the stress tensor of Hooke's law and model the problem by (2.1), where $\partial\Omega = \Gamma_{\mathrm{D}} \cup \Gamma_{\mathrm{N}}$ and $\Omega = (0, 4) \times (0, 2) \times (0, 1)$; cf. Figure 9 (left). The Dirichlet boundary is chosen as $\Gamma_{\mathrm{D}} = \{(x, y, z) \in \Omega \mid z = 0\}$ and the Neumann boundary as $\Gamma_{\mathrm{N}} = \partial\Omega \backslash \Gamma_{\mathrm{D}}$. In this scenario, we ignore volume forces, and thus we set $\mathbf{f} = \mathbf{0}$. The material block is clamped at the bottom, and therefore we set $\mathbf{g} = \mathbf{0}$. Further, the following planar force $\hat{\mathbf{t}}$ is applied to the top plane of the foam:

$$\hat{\mathbf{t}}(x, y, z) = \begin{cases} (0, 0, -1)^\top, & z = 1 \\ (0, 0, 0)^\top & \text{else} \end{cases} \text{GPa}.$$

We assume that the material of interest is a metal foam, and thus we apply the *Gibson and Ashby model* [37, 17] which assumes the following relationship between the elastic modulus of the metal foam E_f and of the matrix E_m:

$$\frac{E_f}{E_m} \approx \phi^2 \left(\frac{\rho_f}{\rho_m} \right)^2 + (1 - \phi) \frac{\rho_f}{\rho_m}, \tag{5.2}$$

where ρ_f is the foam's density, ρ_m the matrix density, and ϕ the porosity of the foam. Again, we assume the matrix to consist of aluminum with a Poisson's ratio of $\nu = 0.34$ and the elastic modulus $E_m = 70\,\text{GPa}$. Additionally, we assume that the ratio of foam- and matrix-density is given by a radially symmetric function of the form

$$\frac{\rho_f}{\rho_m} = \frac{1}{16} x(4 - x)z(2 - y) + \frac{1}{2},$$

and $\phi = 1 - \frac{\rho_f}{\rho_m}$. The foam's elastic modulus E_f is then obtained by relationship (5.2).

We discretize the block with 3072 tetrahedra on the coarsest level $\ell = 0$. The finest level considered in this subsection is $L = 5$. Each system of equations is solved using 48 compute cores with the same multigrid solver as in the previous subsection. Since no analytical solution is available, we assume that the solution obtained with the reference bilinear form $a_h(\cdot, \cdot)$ is the true solution and compare it to the solutions

TABLE 5

Errors for the linear elastostatics benchmark problem in the discrete L^2 norm, convergence rates, number of V-cycle iterations, and time-to-solution and relative time-to-solution for nodal integration and physical scaling recorded for different refinement levels ℓ and parameters m. The measurements were conducted on SuperMUC-NG.

		Nodal integration				Physical scaling				Rel.
ℓ	DoFs	error	eoc	iter	tts [s]	error	eoc	iter	tts [s]	tts
					$m = 1$					
1	$1.52 \cdot 10^3$	$8.11 \cdot 10^{-3}$	0.00	12	0.07	$8.07 \cdot 10^{-3}$	0.00	12	0.07	0.93
2	$1.21 \cdot 10^4$	$2.12 \cdot 10^{-3}$	1.94	18	0.35	$2.10 \cdot 10^{-3}$	1.94	18	0.29	0.81
3	$9.72 \cdot 10^4$	$5.43 \cdot 10^{-4}$	1.97	25	2.97	$5.39 \cdot 10^{-4}$	1.97	25	2.03	0.68
4	$7.77 \cdot 10^5$	$1.38 \cdot 10^{-4}$	1.97	29	26.50	$1.37 \cdot 10^{-4}$	1.97	29	17.13	0.65
5	$6.22 \cdot 10^6$	$3.53 \cdot 10^{-5}$	1.97	32	230.78	$3.51 \cdot 10^{-5}$	1.97	32	143.04	0.62
6	$4.97 \cdot 10^7$	$4.71 \cdot 10^{-6}$	2.91	32	1848.95	$4.65 \cdot 10^{-6}$	2.92	32	1125.28	0.61
					$m = 2$					
1	$1.52 \cdot 10^3$	$8.26 \cdot 10^{-3}$	0.00	12	0.08	$8.23 \cdot 10^{-3}$	0.00	12	0.07	0.93
2	$1.21 \cdot 10^4$	$2.16 \cdot 10^{-3}$	1.93	19	0.37	$2.15 \cdot 10^{-3}$	1.94	19	0.30	0.81
3	$9.72 \cdot 10^4$	$5.54 \cdot 10^{-4}$	1.97	25	2.96	$5.50 \cdot 10^{-4}$	1.97	25	2.02	0.68
4	$7.77 \cdot 10^5$	$1.41 \cdot 10^{-4}$	1.97	29	26.40	$1.40 \cdot 10^{-4}$	1.97	29	16.88	0.64
5	$6.22 \cdot 10^6$	$3.59 \cdot 10^{-5}$	1.97	32	233.40	$3.57 \cdot 10^{-5}$	1.97	32	142.98	0.61
6	$4.97 \cdot 10^7$	$4.92 \cdot 10^{-6}$	2.87	33	1910.68	$4.86 \cdot 10^{-6}$	2.87	33	1173.80	0.61
					$m = 3$					
1	$1.52 \cdot 10^3$	$8.28 \cdot 10^{-3}$	0.00	12	0.07	$8.30 \cdot 10^{-3}$	0.00	12	0.07	0.95
2	$1.21 \cdot 10^4$	$2.16 \cdot 10^{-3}$	1.94	19	0.37	$2.17 \cdot 10^{-3}$	1.94	19	0.30	0.81
3	$9.72 \cdot 10^4$	$5.54 \cdot 10^{-4}$	1.97	25	2.95	$5.54 \cdot 10^{-4}$	1.97	25	2.01	0.68
4	$7.77 \cdot 10^5$	$1.41 \cdot 10^{-4}$	1.97	30	27.58	$1.41 \cdot 10^{-4}$	1.97	29	16.87	0.61
5	$6.22 \cdot 10^6$	$3.59 \cdot 10^{-5}$	1.97	32	230.73	$3.59 \cdot 10^{-5}$	1.97	32	145.07	0.63
6	$4.97 \cdot 10^7$	$4.99 \cdot 10^{-6}$	2.85	33	1902.81	$5.04 \cdot 10^{-6}$	2.84	33	1160.50	0.61
					$m = 8$					
1	$1.52 \cdot 10^3$	$8.67 \cdot 10^{-3}$	0.00	11	0.07	$9.04 \cdot 10^{-3}$	0.00	11	0.06	0.94
2	$1.21 \cdot 10^4$	$2.26 \cdot 10^{-3}$	1.94	18	0.36	$2.44 \cdot 10^{-3}$	1.89	18	0.29	0.79
3	$9.72 \cdot 10^4$	$5.75 \cdot 10^{-4}$	1.98	24	2.83	$6.36 \cdot 10^{-4}$	1.94	24	1.96	0.69
4	$7.77 \cdot 10^5$	$1.46 \cdot 10^{-4}$	1.98	28	25.69	$1.63 \cdot 10^{-4}$	1.96	28	16.46	0.64
5	$6.22 \cdot 10^6$	$3.72 \cdot 10^{-5}$	1.97	31	223.97	$4.14 \cdot 10^{-5}$	1.98	31	138.66	0.62
6	$4.97 \cdot 10^7$	$5.54 \cdot 10^{-6}$	2.75	32	1844.60	$6.90 \cdot 10^{-6}$	2.58	32	1124.61	0.61

obtained using the form $\hat{a}_h(\cdot,\cdot)$. We denote the solutions by $\mathbf{u}_h$ and $\hat{\mathbf{u}}_h$, respectively. Again, we do not consider the stored stencil approach in this comparison, since the problem sizes on the finest level are too large, and the matrices could not be stored in memory. The error on level ℓ is defined by $\|\mathcal{I}_{h_L}\mathbf{v}_{h_\ell} - \mathbf{u}_{h_L}\|$ for $\mathbf{v} \in \{\mathbf{u}, \hat{\mathbf{u}}\}$ and $\ell \leq L$. See Figure 9 (right) for an illustration of the deformed metal foam computed on level $\ell = 4$.

In Table 7, we report on the errors, convergence rates, number of V-cycle iterations, and run-times for different refinement levels ℓ. We do not observe optimal quadratic convergence in the discrete L^2 norm, even in the nodal integration case, because of the lower regularity of the problem. The solution obtained by the physical scaling, however, has the same convergence rate but with a relative tts of about 64%.

5.2. Generalized incompressible Stokes problem. In order to show that the new approach is also applicable to indefinite problems, we consider a generalized incompressible Stokes problem with a variable viscosity. The stress tensor of a generalized Newtonian fluid with viscosity μ is given by $\boldsymbol{\sigma}(\mathbf{u}, p) = 2\mu\boldsymbol{\varepsilon}(\mathbf{u}) - pI$ and

TABLE 6
Time-to-solution and relative time-to-solution in the linear elastostatics benchmark for nodal integration and physical scaling recorded for different refinement levels ℓ and parameters m. The measurements were conducted on SuperMUC Phase 2.

ℓ	Nodal integration tts [s]	Physical scaling tts [s]	Rel. tts
	$m = 1$		
1	0.10	0.08	0.88
2	0.84	0.34	0.40
3	4.48	2.39	0.53
4	42.36	20.62	0.49
5	381.03	176.84	0.46
6	3065.13	1394.86	0.46
	$m = 2$		
1	0.09	0.09	0.93
2	0.53	0.36	0.68
3	4.79	2.35	0.49
4	44.31	20.73	0.47
5	381.06	174.61	0.46
6	3170.62	1420.91	0.45
ℓ	Nodal integration tts [s]	Physical scaling tts [s]	Rel. tts
	$m = 3$		
1	0.09	0.08	0.91
2	0.58	0.35	0.61
3	4.76	2.43	0.51
4	45.51	21.02	0.46
5	379.64	174.09	0.46
6	3159.07	1420.09	0.45
	$m = 8$		
1	0.09	0.08	0.91
2	0.49	0.42	0.85
3	4.53	2.26	0.50
4	42.36	20.35	0.48
5	369.05	168.28	0.46
6	3079.40	1374.16	0.45

depends not only on the velocity $\mathbf{u}$ but also on the pressure p. The problem considered in this section is modeled by the equations

$$\begin{aligned} -\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} &= \mathbf{f} \quad \text{in } \Omega, \\ \boldsymbol{\nabla} \cdot \mathbf{u} &= 0 \quad \text{in } \Omega, \\ \mathbf{u} &= \mathbf{g} \quad \text{on } \Gamma_{\mathrm{D}}, \\ \boldsymbol{\sigma} \cdot \mathbf{n} &= \hat{\mathbf{t}} \quad \text{on } \Gamma_{\mathrm{N}} \end{aligned}$$

on a domain $\Omega \subset \mathbb{R}^3$ with a Dirichlet boundary Γ_{D} and Neumann boundary Γ_{N}. For the well posedness of the problem, the finite element spaces need to meet a uniform inf-sup condition, which is not the case for an equal-order $P1$ discretization. Therefore, we add a level-dependent residual-based stabilization term c_ℓ [10] to the mass conservation equation, i.e., $c_\ell(p, q) = -\frac{h_\ell^2}{12} \langle \boldsymbol{\nabla} p, \boldsymbol{\nabla} q \rangle_\Omega$. If $\Gamma_{\mathrm{N}} = \emptyset$, then the pressure is not unique

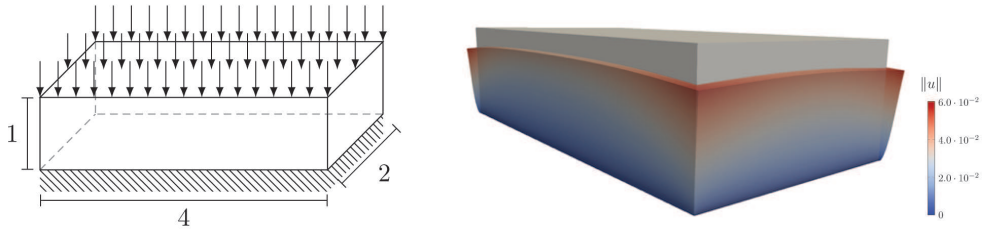

FIG. 9. *Experimental setup and dimensions of the metal foam (left). Initial (gray) and displaced (colored) foam after applying the force on top. The displacement is magnified by a factor of 5. The numerical solution was computed on refinement level $\ell = 4$ with standard nodal integration (right).*

TABLE 7
Errors of the linear elastostatics example with external forces in the discrete L^2 norm, convergence rates, number of V-cycle iterations, time-to-solution, and relative time-to-solution for nodal integration, and physical scaling recorded for different refinement levels ℓ.

		Nodal integration				Physical scaling				Rel.
ℓ	DoFs	error	eoc	iter	tts [s]	error	eoc	iter	tts [s]	tts
1	$1.23 \cdot 10^4$	$4.46 \cdot 10^{-4}$	0.00	12	0.38	$4.44 \cdot 10^{-4}$	0.00	12	0.42	1.11
2	$9.86 \cdot 10^4$	$1.70 \cdot 10^{-4}$	1.39	16	0.64	$1.69 \cdot 10^{-4}$	1.39	16	0.67	1.05
3	$7.89 \cdot 10^5$	$6.45 \cdot 10^{-5}$	1.40	18	1.27	$6.42 \cdot 10^{-5}$	1.40	18	1.20	0.94
4	$6.31 \cdot 10^6$	$2.31 \cdot 10^{-5}$	1.48	19	4.87	$2.30 \cdot 10^{-5}$	1.48	19	3.74	0.77
5	$5.05 \cdot 10^7$	$6.59 \cdot 10^{-6}$	1.81	19	30.02	$6.57 \cdot 10^{-6}$	1.81	19	19.22	0.64

up to a constant, and we enforce uniqueness by demanding that the mean value of the pressure be zero. This equal-order discretization is inconsistent and does not conserve the mass locally; however, the discrete solutions still converge with the optimal order. There are possibilities of obtaining local mass conservation by a postprocess, which is discussed in [32].

5.2.1. Stationary geophysics example. To demonstrate that the presented method is also suitable for solving geophysical problems, we present an example inspired by convection in Earth's mantle. The domain is chosen as $\Omega = (0,1)^3$ with $\Gamma_D = \partial\Omega$ and $\Gamma_N = \emptyset$. In this scenario, the viscosity and the volume forces depend on the temperature. Therefore, we construct a temperature field ϑ, resembling a temperature plume in Earth's mantle given by the formula

$$\vartheta(x,y,z) = \frac{89\,\mathrm{e}^{-30\left(z+\left(\frac{3r}{2}+\frac{3}{4}\right)\left(r-\frac{1}{2}\right)-\frac{3}{10}\right)^2-10\,r^2}}{100} + \frac{49\,\mathrm{e}^{-100\,r^2}}{50\left(\mathrm{e}^{17\,z-\frac{1819}{200}}+1\right)},$$

with $r(x,y,z) = \sqrt{\frac{13\,(x-\frac{1}{2})^2}{10} + \frac{27\,(y-\frac{1}{2})^2}{10}}$. Note that the temperature field is not radially symmetric, and therefore no problem reduction due to symmetry is possible. The viscosity μ of the fluid is then given by an exponential law with a jump across a horizontal plane, i.e.,

$$\mu(x,y,z) = \mathrm{e}^{-\vartheta(x,y,z)} \cdot \begin{cases} 10^{-2}, & z > \frac{3}{4}, \\ 1 & \text{else.} \end{cases}$$

See Figure 10 (left) for an illustration. In cases like this, where the location of a jump is known a priori, it is possible to resolve the jump via the macro-mesh, since the

standard on-the-fly integration is performed across these interfaces. If the locations of jumps are not known beforehand, it is still possible to locally mark elements where the standard on-the-fly integration should be performed. This solution will, of course, reduce the performance because of the additional branches in the code and possible load imbalances between processes. Stokes flow problems with larger viscosity jumps or a highly heterogeneous viscosity require more sophisticated preconditioners than the geometric multigrid solver used here. See, e.g., [33] for how to construct a robust iterative solver in this case. Additionally, we assume a gravitational source term $\mathbf{f} = \vartheta \cdot (0, 0, 10)^\top$ arising from a Boussinesq approximation [32, 31]. Figure 10 (right) shows the velocity streamlines of the numerical solution using nodal integration computed on a mesh with 50,331,648 tetrahedra.

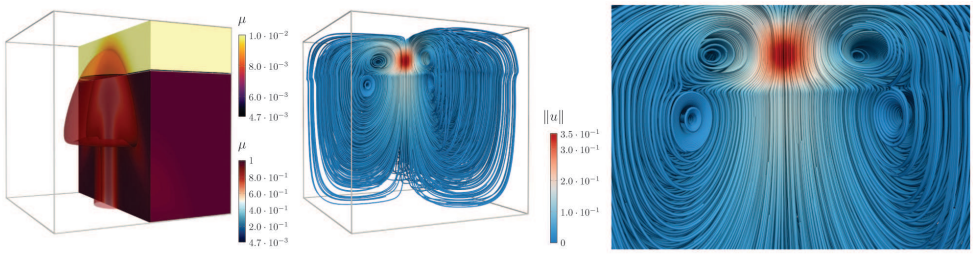

FIG. 10. *Viscosity μ depending on the given plume temperature field ϑ with isosurface $\mu = 0.85$ in the lower part and $\mu = 0.0085$ in the upper part (left). Velocity streamlines of the numerical solution computed on a mesh with* 50,331,648 *tetrahedra using nodal integration (middle). Zoom on the velocity streamlines at the center (right).*

In the following scenario, the coarsest level $\ell = 0$ is discretized by 786,432 tetrahedra, and each system is solved using 12,288 compute cores on SuperMUC-NG. The finest level $\ell = 6$ involved solving a system with about $1.03 \cdot 10^{11}$ DoFs. In order to solve the systems, we employ the inexact Uzawa solver presented in [14], with variable $V(3, 3)$ cycles where two smoothing steps are added to each coarser refinement level which enforces convergence of the method. As a smoother, we again employ the hybrid Gauss–Seidel method but with a relaxation parameter of 0.3 in the pressure part. On the coarsest level, we employ the diagonally preconditioned MINRES method, since the problem is not positive definite but symmetric. The solutions obtained with both approaches do not show any visual differences on all levels. In Table 8, we report on the number of inexact Uzawa iterations and tts for different refinement levels ℓ. The relative tts of the physical scaling is based on the tts of the nodal integration. The worse relative tts in comparison to the linear elasticity examples is due to the fact that the cost of the divergence matrices and the stabilization matrix need to be taken into account. Since the stencils of these matrices are constant on each macro-primitive and do not depend on a coefficient, we employ specialized kernels for them in all of the approaches. This part of the cost remains unchanged for both approaches. Therefore, assuming an optimal relative tts of 61% for the velocity block results in a theoretical optimal relative tts of only about 78% for the total Stokes operator. This value is close to the value observed for $\ell = 6$ in Table 8. We observe that the number of V-cycle iterations required to obtain the relative residual tolerance is larger for multigrid hierarchies with fewer levels compared to a larger number of levels. Furthermore, the tts for the levels $\ell = 1$ to $\ell = 4$ are very close to each other. Since the number of MPI ranks is fixed and the coarse grid solver does not scale optimally, most of the time is

spent on the coarse grid if the number of multigrid levels is small. This cost of the coarse grid solver decreases relatively if the number of multigrid levels increases and more time is spent for smoothing on the finer levels.

TABLE 8
Velocity and pressure errors of the stationary geophysics example in the discrete L^2 norm, convergence rates, number of inexact Uzawa iterations, time-to-solution, and relative time-to-solution for nodal integration and physical scaling recorded for different refinement levels ℓ.

		Nodal integration		Physical scaling		Rel.
ℓ	DoFs	iter	tts [s]	iter	tts [s]	tts
1	$4.19 \cdot 10^6$	12	36.58	12	39.30	1.07
2	$3.35 \cdot 10^7$	11	32.58	11	34.98	1.07
3	$2.68 \cdot 10^8$	10	31.69	10	32.22	1.02
4	$2.15 \cdot 10^9$	9	32.60	9	32.38	0.99
5	$1.72 \cdot 10^{10}$	9	62.84	9	54.91	0.87
6	$1.03 \cdot 10^{11}$	8	262.74	8	197.58	0.75

5.2.2. Nonlinear generalized Stokes problem. In this section, we consider the scenario of a nonlinear incompressible Stokes problem where the fluid is assumed to be of generalized Newtonian type, modeled by a shear-thinning Carreau model,

$$\mu(\mathbf{u}) = \eta_\infty + (\eta_0 - \eta_\infty)\left(1 + \kappa|\boldsymbol{\varepsilon}(\mathbf{u})|^2\right)^r.$$

The considered parameters in dimensionless form are specified in Figure 11 (left). These parameters stem from experimental results; cf. [16, Chapter II].

The computational domain Ω is depicted in Figure 11 (right), discretized by 14,208 tetrahedra on the coarsest level $\ell = 0$. The boundary $\partial\Omega$ is composed of Dirichlet and Neumann parts, i.e., $\partial\Omega = \Gamma_\mathrm{D} \cup \Gamma_\mathrm{N}$ with $\Gamma_\mathrm{D} = \{(x, y, z) \in \partial\Omega \mid x < 5\}$ and $\Gamma_\mathrm{N} = \partial\Omega \backslash \Gamma_\mathrm{D}$. The volume force term $\mathbf{f}$, the external forces $\hat{\mathbf{t}}$, and the Dirichlet boundary term $\mathbf{g}$ are set to

$$\mathbf{f} = (0, 0, 100)^\top,\ \hat{\mathbf{t}} = (0, 0, 0)^\top,\ \text{and}\ \mathbf{g} = 16\, y\,(1 - y)\, z\,(1 - z) \cdot (1, 0, 1)^\top.$$

We solve this nonlinear system by applying an inexact fixed-point iteration similar to the nonlinear solver described in [13], where the underlying linear systems are only solved approximately to prevent oversolving. The pseudocode of our approach is presented in Algorithm 5.1. The inexact Uzawa multigrid solver described in the previous subsection is used for the computations in this subsection. The problem is solved using a total of 111 MPI ranks with three compute nodes on SuperMUC-NG. Following the standard notation, the discretized saddle-point problem in a single fixed-point iteration reads

$$\begin{pmatrix} \mathsf{A}\left(\mu^{(n)}\right) & \mathsf{B}^\top \\ \mathsf{B} & \mathsf{C} \end{pmatrix} \begin{pmatrix} \mathsf{u}^{(n+1)} \\ \mathsf{p}^{(n+1)} \end{pmatrix} = \begin{pmatrix} \mathsf{f} \\ \mathsf{0} \end{pmatrix}. \tag{5.3}$$

In Figure 12, we plot the final viscosity profile and y-component of the velocity along the line $\theta = [0, 5] \times \{0.5\} \times \{0.42\}$ for both approaches computed on $\ell = 4$. We see that the solutions of the nodal integration and physical scaling approaches coincide. For the sake of completeness, we also present the unphysical scaling results, which yield different curves in both the top and bottom of Figure 12.

Parameter	Value
η_0	140.764
η_∞	1.0
κ	212.2
r	−0.325

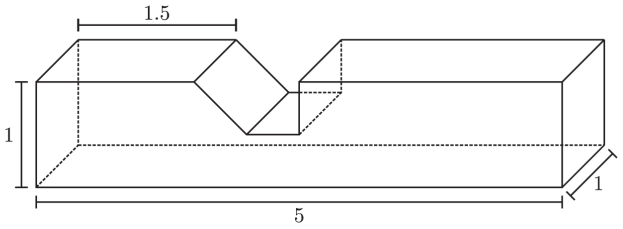

FIG. 11. *Dimensionless parameters for the Carreau viscosity model (left). Experimental setup and dimensions of channel (right).*

Algorithm 5.1 Fixed-point iterations coupled with a multigrid solver.

Set $\mathsf{u}^{(0)} = \mathsf{0}$, $\mathsf{p}^{(0)} = \mathsf{0}$, $n = 0$
Set $\mu^{(0)} = \mu(\mathsf{u}^{(0)})$ by employing the local approximation from section 3
repeat
  Solve system (5.3) for $\mathsf{u}^{(n+1)}$ and $\mathsf{p}^{(n+1)}$ by applying an inexact Uzawa V(3,3)-cycle
  Set $\mu^{(n+1)} = \mu\left(\mathsf{u}^{(n+1)}\right)$ by employing the local approximation from section 3
  Set $n = n + 1$
until $\max_i\{\|\mathsf{u}_i^{(n)} - \mathsf{u}_i^{(n-1)}\|_2\} < 10^{-3} \cdot \max_i\{\|\mathsf{u}_i^{(n)}\|_2\}$
Solve system (5.3) for $\mathsf{u}^{(n+1)}$ and $\mathsf{p}^{(n+1)}$ by applying final Uzawa V(3,3)-cycles until a relative residual of 10^{-3} is obtained
Set $\mu^{(n+1)} = \mu\left(\mathsf{u}^{(n+1)}\right)$ by employing the local approximation from section 3
return $\mathsf{u}^{(n+1)}$, $\mathsf{p}^{(n+1)}$, and $\mu^{(n+1)}$

In Table 9, we report on the number of fixed-point iterations, the number of final iterations, and the absolute and relative tts for the nodal integration and physical scaling. We observe a relative tts of about 87% on the finest level $\ell = 6$.

Since the coefficient μ changes after each multigrid V-cycle, the caching of face stencils as was done in the previous sections is not possible. This has a large impact on the run-time for lower levels in the hierarchy, since the cost may be dominated by the face primitives. Only asymptotically, for fine levels, the cost of the face primitives is small compared to the cost of the element primitives. In this numerical experiment, the solver performance is worse than in the previous examples, because of the inherent difficulty of the nonlinear problem and the expensive on-the-fly nodal integration of the bilinear form on the macro-faces.

6. Conclusion. We have presented a new method to improve the performance of matrix-free operator applications for vector-valued second-order elliptic PDEs. Although we restricted ourselves to linear finite elements on hierarchical hybrid grids, the idea of exploiting structure and symmetry in the mesh applies for higher orders as well, which we described in Remark 2.3. The method is based on scaling reference stencils originating from a constant coefficient discretization by variable coefficients. We showed that in theory a correction term is required in cases where the coefficient is not constant. Furthermore, we presented how to precompute these correction terms in the case of HHGs and how to rescale them using the coefficient. This new approach was aimed at reducing memory traffic and at reducing the number of required FLOPs. In order to show this, we first derived theoretical models about the required number of

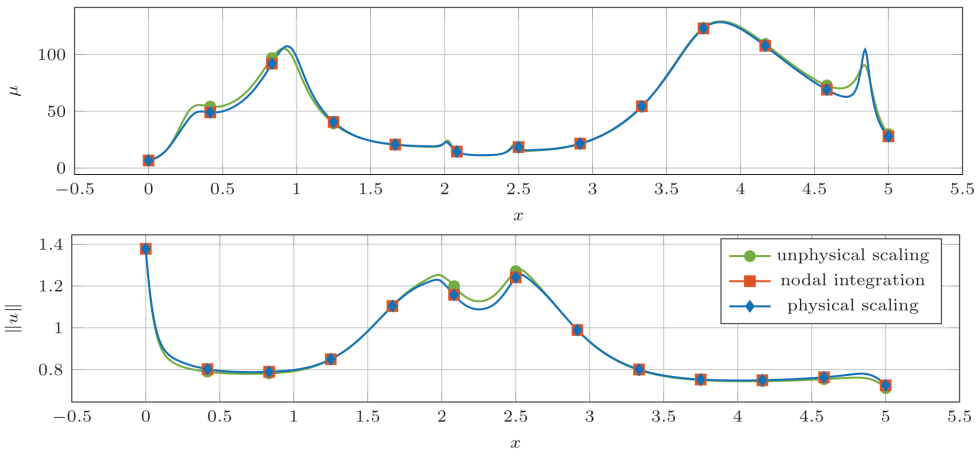

FIG. 12. *Viscosity profile (top) and profile of the velocity magnitude (bottom) plotted over θ for different assembly approaches.*

TABLE 9
Relative time-to-solution comparison of the nodal integration and physical scaling approach for the nonlinear generalized Stokes problem.

ℓ	DoFs	Nodal integration			Physical scaling			Rel.
		fixed-point iter	final iter	tts [s]	fixed-point iter	final iter	tts [s]	tts
3	$4.69 \cdot 10^6$	26	14	42.50	26	14	41.80	0.98
4	$3.82 \cdot 10^7$	29	7	127.98	29	7	121.87	0.95
5	$3.08 \cdot 10^8$	25	8	571.84	29	8	578.57	1.01
6	$2.47 \cdot 10^9$	24	7	3349.39	25	6	2925.09	0.87

FLOPs and the memory traffic. We validated these estimates using benchmarks and numerical experiments on the supercomputer SuperMUC-NG. The results have shown that specially designed matrix-free methods such as ours may be beneficial compared to matrix-based methods not only for higher-order discretizations but also for low-order discretizations, where the arithmetic intensity is lower. The numerical benchmarks and experiments involving linear elasticity and Stokes flow also showed that our method is faster than and comparably accurate to standard methods. Although we applied this method using only a geometric multigrid solver, it can also be used with more scalable coarse grid solvers, such as algebraic multigrid, or combined with preconditioners designed for large viscosity jumps or heterogeneous viscosities in Stokes flow problems. This makes our method attractive for highly resolved simulations on current and future machines where the available memory is limited compared to the compute power.

Acknowledgments. The authors wish to thank the referees for the detailed feedback they gave during the review process. Their insights significantly improved the quality of the final manuscript. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (GCS, https://www.gauss-centre.eu/) for funding this project by providing computing time on the GCS supercomputer SuperMUC at Leibniz Supercomputing Centre (LRZ, www.lrz.de).

REFERENCES

[1] A. Afzal, *The Cost of Computation: Metrics and Models for Modern Multicore-Based Systems in Scientific Computing*, Master's thesis, Department Informatik, Friedrich Alexander Universität Erlangen-Nürnberg, 2015.

[2] C. L. Alappat, J. Hofmann, G. Hager, H. Fehske, A. R. Bishop, and G. Wellein, *Understanding HPC Benchmark Performance on Intel Broadwell and Cascade Lake Processors*, preprint, https://arxiv.org/abs/2002.03344, 2020.

[3] P. Arbenz, G. H. van Lenthe, U. Mennel, R. Müller, and M. Sala, *A scalable multi-level preconditioner for matrix-free μ-finite element analysis of human bone structures*, Int. J. Numer. Methods Engrg., 73 (2008), pp. 927–947.

[4] S. Bauer, D. Drzisga, M. Mohr, U. Rüde, C. Waluga, and B. Wohlmuth, *A stencil scaling approach for accelerating matrix-free finite element implementations*, SIAM J. Sci. Comput., 40 (2018), pp. C748–C778, https://doi.org/10.1137/17M1148384.

[5] B. Bergen, *Hierarchical Hybrid Grids: Data Structures and Core Algorithms for Efficient Finite Element Simulations on Supercomputers*, SCS Publishing House, 2005.

[6] B. Bergen and F. Hülsemann, *Hierarchical hybrid grids: Data structures and core algorithms for multigrid*, Numer. Linear Algebra Appl., 11 (2004), pp. 279–291.

[7] B. Bergen, G. Wellein, F. Hülsemann, and U. Rüde, *Hierarchical hybrid grids: Achieving TERAFLOP performance on large scale finite element simulations*, Int. J. Parallel Emergent Distrib. Syst., 22 (2007), pp. 311–329.

[8] J. Bey, *Tetrahedral grid refinement*, Computing, 55 (1995), pp. 355–378.

[9] J. Bielak, O. Ghattas, and E.-J. Kim, *Parallel octree-based finite element method for large-scale earthquake ground motion simulation*, CMES Comput. Model. Eng. Sci., 10 (2005), pp. 99–112.

[10] F. Brezzi and J. Pitkäranta, *On the stabilization of finite element approximations of the Stokes equations*, in Efficient Solutions of Elliptic Systems, Springer, 1984, pp. 11–19.

[11] J. Brown, *Efficient nonlinear solvers for nodal high-order finite elements in 3D*, J. Sci. Comput., 45 (2010), pp. 48–63.

[12] G. F. Carey and B.-N. Jiang, *Element-by-element linear and nonlinear solution schemes*, Comm. Appl. Numer. Methods, 2 (1986), pp. 145–153.

[13] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), pp. 400–408, https://doi.org/10.1137/0719025.

[14] D. Drzisga, L. John, U. Rüde, B. Wohlmuth, and W. Zulehner, *On the analysis of block smoothers for saddle point problems*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 932–960, https://doi.org/10.1137/16M1106304.

[15] C. Flaig and P. Arbenz, *A highly scalable matrix-free multigrid solver for μFE analysis based on a pointer-less octree*, in Large-Scale Scientific Computing: 8th International Conference (LSSC 2011), Sozopol, Bulgaria, Revised Selected Papers, I. Lirkov, S. Margenov, and J. Waśniewski, eds., Springer, 2012, pp. 498–506.

[16] G. P. Galdi, R. Rannacher, A. M. Robertson, and S. Turek, *Hemodynamical Flows: Modeling, Analysis and Simulation*, Birkhäuser, 2008.

[17] L. J. Gibson and M. F. Ashby, *Cellular Solids: Structure and Properties*, Cambridge University Press, 1999.

[18] B. Gmeiner, T. Gradl, H. Köstler, and U. Rüde, *Highly parallel geometric multigrid algorithm for hierarchical hybrid grids*, in NIC Symposium 2012 - Proceedings, Jülich, Germany, NIC Ser. 45, 2012, pp. 323–330.

[19] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, CRC Press, 2010.

[20] A. Ilic, F. Pratas, and L. Sousa, *Cache-aware Roofline model: Upgrading the loft*, IEEE Comput. Architecture Lett., 13 (2013), pp. 21–24.

[21] Intel Corp., *Intel Advisor*, https://software.intel.com/en-us/intel-advisor-xe, 2019.

[22] Intel Corp., *Intel VTune Profiler*, https://software.intel.com/en-us/vtune, 2019.

[23] N. Kohl, D. Thönnes, D. Drzisga, D. Bartuschat, and U. Rüde, *The HyTeG finite-element software framework for scalable multigrid solvers*, Int. J. Parallel Emergent Distrib. Syst., 34 (2019), pp. 477–496.

[24] M. Kronbichler and K. Kormann, *A generic interface for parallel cell-based finite element operator application*, Comput. & Fluids, 63 (2012), pp. 135–147.

[25] K. Ljungkvist, *Matrix-free finite-element computations on graphics processors with adaptively refined unstructured meshes*, in Proceedings of the 25th High Performance Computing Symposium (HPC '17), Society for Computer Simulation International, 2017, pp. 1:1–1:12.

[26] K. Ljungkvist and M. Kronbichler, *Multigrid for Matrix-Free Finite Element Computations*

on Graphics Processors, Tech. report 2017-006, Department of Information Technology, Uppsala University, 2017.

[27] J. Loffeld and J. Hittinger, *On the arithmetic intensity of high-order finite-volume discretizations for hyperbolic systems of conservation laws*, Int. J. High Performance Comput. Appl., 33 (2019), pp. 25–52.

[28] LRZ, *Hardware of SuperMUC-NG*, https://doku.lrz.de/display/PUBLIC/Hardware+of+SuperMUC-NG (retrieved 25 February 2020).

[29] LRZ, *SuperMUC Petascale System*, https://www.lrz.de/services/compute/supermuc/systemdescription/ (retrieved 29 November 2018).

[30] D. A. May, J. Brown, and L. L. Pourhiet, *A scalable, matrix-free multigrid preconditioner for finite element discretizations of heterogeneous Stokes flow*, Comput. Methods Appl. Mech. Engrg., 290 (2015), pp. 496–523.

[31] Y. Ricard, *Physics of mantle convection*, Treatise on Geophysics, 7 (2007), pp. 31–81.

[32] U. Rüde, C. Waluga, and B. Wohlmuth, *Mass-corrections for the conservative coupling of flow and transport on collocated meshes*, J. Comput. Phys., 305 (2016), pp. 319–332.

[33] J. Rudi, G. Stadler, and O. Ghattas, *Weighted BFBT preconditioner for Stokes flow problems with highly heterogeneous viscosity*, SIAM J. Sci. Comput., 39 (2017), pp. S272–S297, https://doi.org/10.1137/16M108450X.

[34] H. Stengel, J. Treibig, G. Hager, and G. Wellein, *Quantifying performance bottlenecks of stencil computations using the execution-cache-memory model*, in Proceedings of the 29th ACM International Conference on Supercomputing, ACM, 2015, pp. 207–216.

[35] B. van Rietbergen, H. Weinans, R. Huiskes, and B. Polman, *Computational strategies for iterative solutions of large FEM applications employing voxel data*, Int. J. Numer. Methods Engrg., 39 (1996), pp. 2743–2767.

[36] S. Williams, A. Waterman, and D. Patterson, *Roofline: An insightful visual performance model for multicore architectures*, Commun. ACM, 52 (2009), pp. 65–76.

[37] Y. Zhang, D. Rodrigue, and A. Ait-Kadi, *High density polyethylene foams.* II. *Elastic modulus*, J. Appl. Polymer Sci., 90 (2003), pp. 2120–2129.

[38] O. C. Zienkiewicz and J. Z. Zhu, *The superconvergent patch recovery and a posteriori error estimates: Part* 1*: The recovery technique*, Int. J. Numer. Methods Engrg., 33 (1992), pp. 1331–1364.