A matrix-free approach for finite-strain hyperelastic problems using geometric multigrid

Denis Davydov[a,*], Jean-Paul Pelteret[a], Daniel Arndt[b], Paul Steinmann[a,c]

[a] *Chair of Applied Mechanics, Friedrich-Alexander-Universität Erlangen-Nürnberg, Egerlandstr. 5, 91058 Erlangen, Germany*
[b] *Interdisciplinary Center for Scientific Computing (IWR), Heidelberg University, Im Neuenheimer Feld 205, 69120 Heidelberg, Germany*
[c] *Glasgow Computational Engineering Center (GCEC), University of Glasgow, G12 8QQ Glasgow, United Kingdom*

Abstract

The performance of finite element solvers on modern computer architectures is typically memory bound for sufficiently large problems. The main cause for this is that loading matrix elements from RAM into CPU cache is significantly slower than performing the arithmetic operations when solving the problem. In order to improve the performance of iterative solvers within the high-performance computing context, so-called matrix-free methods are widely adopted in the fluid mechanics community, where matrix-vector products are computed on-the-fly.

To date, there have been few (if any) assessments into the applicability of the matrix-free approach to problems in solid mechanics. In this work, we perform an initial investigation on the application of the matrix-free approach to problems in quasi-static finite-strain hyperelasticity to determine whether it is viable for further extension. Specifically, we study different numerical implementations of the finite element tangent operator, and determine whether generalized methods of incorporating complex constitutive behavior might be feasible. In order to improve the convergence behavior of iterative solvers, we also propose a method by which to construct level tangent operators and employ them to define a geometric multigrid preconditioner. The performance of the matrix-free operator and the ge-

*Corresponding author.
Email addresses: denis.davydov@fau.de (Denis Davydov), jean-paul.pelteret@fau.de (Jean-Paul Pelteret), daniel.arndt@iwr.uni-heidelberg.de (Daniel Arndt), paul.steinmann@fau.de (Paul Steinmann)

ometric multigrid preconditioner is compared to the matrix-based implementation with an algebraic multigrid preconditioner on a single node for a representative numerical example of a heterogeneous hyperelastic material in two and three dimensions. We conclude that the application of matrix-free methods to finite-strain solid mechanics is promising, and that is it possible to develop numerically efficient implementations that are independent of the hyperelastic constitutive law.

Keywords: adaptive finite element method, geometric multigrid, finite-strain, matrix-free, hyperelasticity

1. Introduction

The performance of finite element solvers on modern computer architectures is typically memory bound for sufficiently large problems. The main cause for this is that loading pre-computed matrix elements from RAM into CPU cache is significantly slower than performing the arithmetic operations when solving the problem. In order to improve the performance of iterative solvers, so-called matrix-free methods, where matrix-vector products are formed on-the-fly [1–5], are widely adopted in the fluid mechanics community. Such methods can also be efficiently implemented on GPUs [6, 7].

To the best of our knowledge, matrix-free methods are not widely adopted within the solid mechanics community. We are only aware of a single paper currently in preparation that deals with small strain linear elasticity [8]. One possible reason is that the tangent operators[1] (stemming from linearization of the non-linear balance equations) are more elaborate as compared to the operators used often in both linear and non-linear fluid mechanics, and it is not obvious whether matrix-free methods can be advantageous in this case. Another reason could be the frequent use of lower-order elements in solid mechanics due to lacking smoothness of the underlying solution. The finite element method (FEM) with linear or quadratic elements (that reduce the potential for locking), or mixed or enhanced formulations (that avoid locking) are often adopted in the solid mechanics community, whereas higher order elements are rarely employed. This can potentially

[1]Here and below 'operator' denotes a finite-dimensional linear operator defined on the vector space $\mathbb{R}^N$, where N is the number of unknown degrees of freedom in the FE discretization. Although any linear mapping on a finite-dimensional space is representable by a matrix, we adopt the 'operator' term to avoid confusion between its matrix-free and matrix-based numerical implementation.

reduce the advantage of matrix-free methods using sum-factorization techniques, that are generally more competitive for higher order elements [1, 9, 10]. However, we note that, even with linear FEM, large scale computations with 10^{12} unknowns inevitably require a matrix-free approach as there is simply not enough memory to store the sparse tangent matrix [5]. Therefore, we believe that for large scale computations in solid mechanics it is crucial to consider matrix-free approaches and develop efficient multigrid solvers.

In this work, we perform a preliminary investigation of different implementations of the finite-strain hyperelastic tangent operator with respect to their computational cost and performance on a single node of a high performance computing cluster. The `deal.II` [11] finite element library, which provides MPI parallelization together with a generalized matrix-free framework that incorporates SIMD vectorization and an interface to high-performance, MPI distributed matrix-based libraries, is used for this study. In order to improve the convergence of iterative solvers, we also propose a method by which to construct level matrix-free tangent operators and employ them to define a geometric multigrid preconditioner.

The paper is organized as follows: In Section 2 we briefly introduce the partial differential equations used in finite-strain solid mechanics. Sections 3 and 4 cover the finite element discretization and provide details of the numerical framework with which this study is implemented. In 5 we describe the different matrix-free operator implementations that are evaluated in this study, and in section 6 we propose a geometric multigrid preconditioner that is suitable for the matrix-free implementation of finite-strain hyperelasticity. The performance of the matrix-free operator and the geometric multigrid preconditioner is compared to the matrix-based implementation with an algebraic multigrid preconditioner for a representative numerical example of a heterogeneous hyperelastic material simulated in two and three dimensions in Section 7. Lastly, the results are summarized in Section 8.

2. Theoretical background

2.1. Weak form

The deformation of a body $\mathcal{B}$ from the referential configuration $\mathcal{B}_0$ to the spatial configuration $\mathcal{B}_t$ at time t is defined via the deformation map $\boldsymbol{\varphi}_t : \mathcal{B}_0 \rightarrow \mathcal{B}_t$, which places material points of $\mathcal{B}$ into the Euclidean space $\mathbb{E}^3$. The spatial location of a material particle $\boldsymbol{X}$ is given by $\boldsymbol{x} = \boldsymbol{\varphi}_t(\boldsymbol{X})$. The displacement field reads $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{X}$.

The tangent map is linear such that $\mathrm{d}\boldsymbol{x} = \boldsymbol{F} \cdot \mathrm{d}\boldsymbol{X}$, where $\boldsymbol{F} := \mathrm{Grad}\,\boldsymbol{\varphi} \equiv \boldsymbol{I} + \mathrm{Grad}\,\boldsymbol{u}$ is called the deformation gradient and $\boldsymbol{I}$ is the second-order unit tensor. The mapping has to be one-to-one and must exclude self-penetration. Consequently, the Jacobian $J = \det \boldsymbol{F} > 0$ has to be positive. We shall denote the push-forward transformation of rank-2 and rank-4 tensors $\mathbf{A}$ and $\mathcal{A}$ by a linear transformation χ defined as

$$\chi\,(\mathbf{A})_{ij} = F_{iA} A_{AB} F_{jB} \tag{1}$$

$$\chi\,(\mathcal{A})_{ijkl} = F_{iA} F_{jB} \mathcal{A}_{ABCD} F_{kC} F_{lD}. \tag{2}$$

Note that $\mathrm{d}\boldsymbol{x} = \chi(\mathrm{d}\boldsymbol{X})$.

In what follows, we parameterize the material behavior using the right Cauchy-Green tensor $\boldsymbol{C} := \boldsymbol{F}^T \cdot \boldsymbol{F}$. We will also use the Green-Lagrange strain tensor $\boldsymbol{E} := \frac{1}{2}\,[\boldsymbol{C} - \boldsymbol{I}]$ and the left Cauchy-Green tensor $\boldsymbol{b} := \boldsymbol{F} \cdot \boldsymbol{F}^T$. Clearly $J = \sqrt{\det \boldsymbol{C}}$ and $\partial(\bullet)/\partial \boldsymbol{E} = 2\,\partial(\bullet)/\partial \boldsymbol{C}$. For conservative systems without body forces, the total potential energy functional $\mathcal{E}$ is introduced as

$$\mathcal{E} = \int_{\mathcal{B}_0} \psi(\boldsymbol{F})\,\mathrm{d}V \;-\; \int_{\partial \mathcal{B}_0^N} \overline{\boldsymbol{T}} \cdot \boldsymbol{u}\,\mathrm{d}S\;, \tag{3}$$

where $\overline{\boldsymbol{T}}$ is the prescribed loading at the Neumann part of the boundary $\partial \mathcal{B}_0^N$ in the referential configuration, ψ denotes the strain-energy per unit reference volume and $\boldsymbol{u}$ should satisfy the prescribed Dirichlet boundary conditions $\boldsymbol{u} = \overline{\boldsymbol{u}}$ on $\partial \mathcal{B}_0^D := \partial \mathcal{B}_0 \setminus \partial \mathcal{B}_0^N \neq \emptyset$.

The principle of stationary potential energy at equilibrium requires that the directional derivative with respect to the displacement

$$\mathrm{D}_{\delta \boldsymbol{u}} \mathcal{E} := \frac{\mathrm{d}}{\mathrm{d}\epsilon} \mathcal{E}(\boldsymbol{u} + \epsilon \delta \boldsymbol{u})\Big|_{\epsilon=0} = 0 \qquad \forall \delta \boldsymbol{u}\,. \tag{4}$$

vanishes for all directions $\delta \boldsymbol{u}$ which satisfy homogeneous Dirichlet boundary conditions. This leads to the following scalar-valued non-linear equation

$$F(\boldsymbol{u}, \delta \boldsymbol{u}) = \int_{\mathcal{B}_0} \boldsymbol{P} : \mathrm{Grad}\,\delta \boldsymbol{u}\,\mathrm{d}V \;-\; \int_{\partial \mathcal{B}_0^N} \overline{\boldsymbol{T}} \cdot \delta \boldsymbol{u}\,\mathrm{d}S \;= 0\,, \tag{5}$$

where $\boldsymbol{P} := \partial \psi / \partial \boldsymbol{F}$ is the Piola stress tensor. The double contraction in the first term can be re-written in terms of the symmetric Kirchhoff stress tensor $\boldsymbol{\tau} := \boldsymbol{P} \cdot \boldsymbol{F}^T$ as

$$\boldsymbol{P} : \mathrm{Grad}\,\delta \boldsymbol{u} = \left[\boldsymbol{\tau} \cdot \boldsymbol{F}^{-T}\right] : \mathrm{Grad}\,\delta \boldsymbol{u} = \boldsymbol{\tau} : \left[\mathrm{Grad}\,\delta \boldsymbol{u} \cdot \boldsymbol{F}^{-1}\right] = \boldsymbol{\tau} : \mathrm{grad}\,\delta \boldsymbol{u}\,, \tag{6}$$

and therefore

$$F(\boldsymbol{u}, \delta\boldsymbol{u}) = \int_{\mathcal{B}_0} \boldsymbol{\tau} : \operatorname{grad}^s \delta\boldsymbol{u} \, \mathrm{d}V \; - \int_{\partial\mathcal{B}_0^N} \overline{\boldsymbol{T}} \cdot \delta\boldsymbol{u} \, \mathrm{d}S \; = 0 \,, \tag{7}$$

where $\operatorname{grad}^s \delta\boldsymbol{u} := \frac{1}{2}\left[\operatorname{grad}\delta\boldsymbol{u} + (\operatorname{grad}\delta\boldsymbol{u})^T\right]$ is involved due to the symmetry of $\boldsymbol{\tau}$. Note that $\boldsymbol{\tau}$ is the push-forward transformation of the Piola-Kirchhoff stress $\boldsymbol{S} := \partial\psi/\partial\boldsymbol{E} \equiv 2\partial\psi/\partial\boldsymbol{C}$, that is $\boldsymbol{\tau} = \chi(\boldsymbol{S}) = \boldsymbol{F}\cdot\boldsymbol{S}\cdot\boldsymbol{F}^T$.

2.2. Linearization

In order to solve (7) using Newton's method, a first order approximation around a given solution field $\overline{\boldsymbol{u}}$ is required such that

$$F(\overline{\boldsymbol{u}} + \Delta\boldsymbol{u}, \delta\boldsymbol{u}) \approx F(\overline{\boldsymbol{u}}, \delta\boldsymbol{u}) + \mathrm{D}_{\Delta\boldsymbol{u}} F(\overline{\boldsymbol{u}}, \delta\boldsymbol{u}), \tag{8}$$

where $\mathrm{D}_{\Delta\boldsymbol{u}}(\bullet)$ denotes the directional derivative in the direction $\Delta\boldsymbol{u}$. For conservative traction boundary conditions, the directional derivative is given by

$$\begin{aligned} \mathrm{D}_{\Delta\boldsymbol{u}} F(\overline{\boldsymbol{u}}, \delta\boldsymbol{u}) = & \int_{\mathcal{B}_0} \mathrm{D}_{\Delta\boldsymbol{u}} \left(\boldsymbol{F}\cdot\boldsymbol{S}\cdot\boldsymbol{F}^T\right) : \overline{\operatorname{grad}^s}\, \delta\boldsymbol{u} \, \mathrm{d}V \\ & + \int_{\mathcal{B}_0} \overline{\boldsymbol{\tau}} : \left[\operatorname{Grad}\delta\boldsymbol{u} \cdot \mathrm{D}_{\Delta\boldsymbol{u}}\boldsymbol{F}^{-1}\right] \mathrm{d}V. \end{aligned} \tag{9}$$

Following [12], this can be simplified to[2]

$$\begin{aligned} \mathrm{D}_{\Delta\boldsymbol{u}} F(\overline{\boldsymbol{u}}, \delta\boldsymbol{u}) = & \int_{\mathcal{B}_0} \overline{\operatorname{grad}^s}\Delta\boldsymbol{u} : J\boldsymbol{C} : \overline{\operatorname{grad}^s}\, \delta\boldsymbol{u} \, \mathrm{d}V \\ & + \int_{\mathcal{B}_0} \overline{\operatorname{grad}}\delta\boldsymbol{u} : \left[\overline{\operatorname{grad}}\Delta\boldsymbol{u} \cdot \overline{\boldsymbol{\tau}}\right] \mathrm{d}V. \end{aligned} \tag{10}$$

Here $\overline{(\bullet)}$ is used to denote quantities evaluated using the displacement field $\overline{\boldsymbol{u}}$, and the fourth-order material part of the spatial tangent stiffness tensor is the push forward of the material part of the referential tangent stiffness tensor $J\boldsymbol{C} = \chi\left(4\frac{d^2\psi(\mathbf{C})}{d\mathbf{C}\otimes d\mathbf{C}}\right)$. The first term in (10) is related to the linearization of the Piola-Kirchhoff stress $\boldsymbol{S}$ and is therefore called material part of the directional derivative. The second term in (10) is called the geometric part of the directional derivative since it originates from the linearization of $\boldsymbol{F}$, $\boldsymbol{F}^T$ and $\boldsymbol{F}^{-1}$.

[2] To that end, $\mathrm{D}_{\Delta\boldsymbol{u}}\boldsymbol{E} = \frac{1}{2}\left[\mathrm{D}_{\Delta\boldsymbol{u}}\boldsymbol{F}^T\cdot\boldsymbol{F} + \boldsymbol{F}^T\cdot\mathrm{D}_{\Delta\boldsymbol{u}}\boldsymbol{F}\right]$, $\mathrm{D}_{\Delta\boldsymbol{u}}\boldsymbol{F} = \operatorname{Grad}\Delta\boldsymbol{u}$ and $\mathrm{D}_{\Delta\boldsymbol{u}}\boldsymbol{F}^{-1} = -\boldsymbol{F}^{-1}\cdot\mathrm{D}_{\Delta\boldsymbol{u}}\boldsymbol{F}\cdot\boldsymbol{F}^{-1}$ are employed together with $\mathrm{D}_{\Delta\boldsymbol{u}}\boldsymbol{S} = 2\partial\boldsymbol{S}/\partial\boldsymbol{C} : \mathrm{D}_{\Delta\boldsymbol{u}}\boldsymbol{E}$.

2.3. Constitutive modelling

For the current study, we use the compressible Neo-Hookean model

$$\psi(\mathbf{C}) = \frac{\mu}{2}\left[\operatorname{tr}\mathbf{C} - \operatorname{tr}\mathbf{I} - 2\ln(J)\right] + \lambda \ln^2(J) \tag{11}$$

where μ and λ denote the shear modulus and Lamé parameter respectively. Derived using statistical thermodynamics applied to cross-linked polymers, the Neo-Hookean model [13, 14] is commonly used to model rubber-like materials in the finite-strain regime. It can be shown that the first and second derivatives of the strain energy function are given by

$$\frac{d\psi(\mathbf{C})}{d\mathbf{C}} = \frac{\mu}{2}\mathbf{I} - \frac{1}{2}\left[\mu - 2\lambda\ln(J)\right]\mathbf{C}^{-1} \tag{12}$$

$$\frac{d^2\psi(\mathbf{C})}{d\mathbf{C}\otimes d\mathbf{C}} = \frac{1}{2}\left[\mu - 2\lambda\ln(J)\right]\left[-\frac{d\mathbf{C}^{-1}}{d\mathbf{C}}\right] + \frac{\lambda}{2}\mathbf{C}^{-1}\otimes\mathbf{C}^{-1} \tag{13}$$

The Kirchhoff stress and its associated fourth-order material part of the spatial tangent tensor are

$$\boldsymbol{\tau} \equiv J\boldsymbol{\sigma} = \chi\left(2\frac{d\psi(\mathbf{C})}{d\mathbf{C}}\right) = \mu\mathbf{b} - \left[\mu - 2\lambda\ln(J)\right]\mathbf{I} \tag{14}$$

and

$$J\boldsymbol{C} = \chi\left(4\frac{d^2\psi(\mathbf{C})}{d\mathbf{C}\otimes d\mathbf{C}}\right) = 2\left[\mu - 2\lambda\ln(J)\right]\boldsymbol{\mathcal{S}} + 2\lambda\mathbf{I}\otimes\mathbf{I} \tag{15}$$

where $\boldsymbol{\mathcal{S}}$ is the fourth-order symmetric identity tensor with $\mathcal{S}_{ijkl} = \frac{1}{2}\left[\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}\right]$. The action that $J\boldsymbol{C}$ performs when contracted with an arbitrary rank-2 symmetric tensor is therefore

$$J\boldsymbol{C} : (\bullet) = 2\left[\mu - 2\lambda\ln(J)\right](\bullet) + 2\lambda\operatorname{tr}(\bullet)\mathbf{I}. \tag{16}$$

3. Finite Element discretization

We now introduce a FE triangulation $\mathcal{B}_0^h$ of $\mathcal{B}_0$ and the associated FE space of continuous piecewise elements with polynomial space of fixed degree. The displacement fields are given in a vector space spanned by standard vector–valued FE basis functions $\boldsymbol{N}_i(\boldsymbol{x})$ (e.g. polynomials with local support on a patch of elements):

$$\boldsymbol{u}^h =: \sum_{i\in\mathcal{I}} u_i\boldsymbol{N}_i(\boldsymbol{X}) \qquad \delta\boldsymbol{u}^h =: \sum_{i\in\mathcal{I}} \delta u_i\boldsymbol{N}_i(\boldsymbol{X}), \tag{17}$$

where the superscript h denotes that this representation is related to the FE mesh with size function $h(\boldsymbol{X})$ and $\mathcal{I}$ is the set of unknown degrees of freedom (DoF).

The Newton-Raphson solution approach is adopted for the nonlinear problem considered here. Given the current trial solution field $\overline{\boldsymbol{u}^h}$, the correction $\Delta\boldsymbol{u}^h$ field is obtained as the solution to the following system of equations (see (7) and (10)):

$$\sum_{j\in\mathcal{I}} A_{ij}\Delta u_j = -F_i \tag{18}$$

$$A_{ij} \equiv a(\boldsymbol{N}_i, \boldsymbol{N}_j) = \int_{\mathcal{B}_0} \underbrace{\overline{\text{grad}^s}\boldsymbol{N}_i : \boldsymbol{JC} : \overline{\text{grad}^s}\,\boldsymbol{N}_j + \overline{\text{grad}}\boldsymbol{N}_i : \left[\overline{\text{grad}}\boldsymbol{N}_j \cdot \overline{\boldsymbol{\tau}}\right]}_{\tilde{a}(\boldsymbol{N}_i, \boldsymbol{N}_j)} \mathrm{d}V \tag{19}$$

$$F_i = \int_{\mathcal{B}_0} \overline{\boldsymbol{\tau}} : \text{grad}^s\,\boldsymbol{N}_i\,\mathrm{d}V - \int_{\partial\mathcal{B}_0^N} \overline{\boldsymbol{T}} \cdot \boldsymbol{N}_i\,\mathrm{d}S. \tag{20}$$

Here, $\boldsymbol{A}$ is the discrete tangent operator, $\boldsymbol{F}$ is the discrete gradient of the potential energy and $\tilde{a}(\boldsymbol{N}_i, \boldsymbol{N}_j)$ is a representation for the integrand in the bilinear form $a(\boldsymbol{N}_i, \boldsymbol{N}_j)$ for the trial solution field $\overline{\boldsymbol{u}^h}$. Note that $\overline{\text{grad}^s}\boldsymbol{N}_i$ and $\overline{\text{grad}}\boldsymbol{N}_i$ are gradients of shape functions in the spatial configuration, whereas the integration is done in the referential configuration.

4. Description of the utilized numerical framework

The discretized system of linear equations formulated in Section 3, in conjunction with the constitutive model described in Section 2.3 were implemented using `deal.II` version 9.0 [11], an open-source finite element library. This library offers a number of paradigms by which to parallelize a finite-element code, and as a member of the xSDK (Extreme-scale Scientific Software Development Kit) ecosystem [15] aims to support exascale computing. In this work we employ `deal.II`'s interface to `p4est` [16] library to perform a standard domain decomposition, whereafter each MPI process "owns" only a subset of elements and the mesh is distributed across all MPI processes. Figure 1 illustrates a distributed MPI decomposition on the fine mesh level for the 2D problem that will be introduced later in Section 7.

In practice, this implies that parallelization of the assembly operation (for the matrix-based approach) and the linear solver (as well as various pre- and post-processing steps), both of which are the focal points of investigation in this manuscript, has been achieved using MPI. For the linear solver, `deal.II`'s implementation of the preconditioned conjugate gradient (CG) solver for symmetric

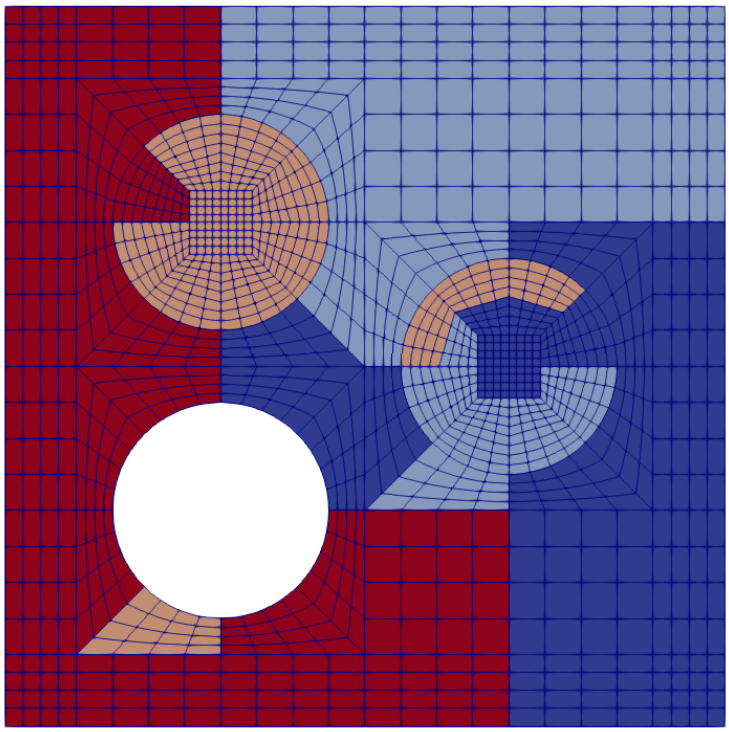


Figure 1: 2D mesh after two global refinements distributed into three MPI processes. The color indicates the process owning the element.

positive-definite systems is consistently used, leaving the choice for method of assembly and linear system preconditioning remaining.

The discrete tangent operator described by (19) can be implemented in one of two ways, namely through a classical matrix-based approach or, alternatively, a matrix-free approach. For the matrix-based approach adopted in this study, the Trilinos [17] library was leveraged; the system tangent matrix is stored in a distributed *Epetra_FECrsMatrix* from the Epetra package [18] sparse data structure and the linear solver was preconditioned with an algebraic multigrid (AMG) preconditioner from the ML package [19]. This is the most highly performant matrix-based approach using the Trilinos framework that, to date, is possible in `deal.II`.

The second implementation, which is discussed in detail in the following Sections, utilized a matrix-free approach in conjunction with a geometric-multigrid preconditioner. As will be detailed later, the pre-existing matrix-free framework that was used offers further opportunity for low-level parallelism that we exploit.

5. Matrix-free operator evaluation

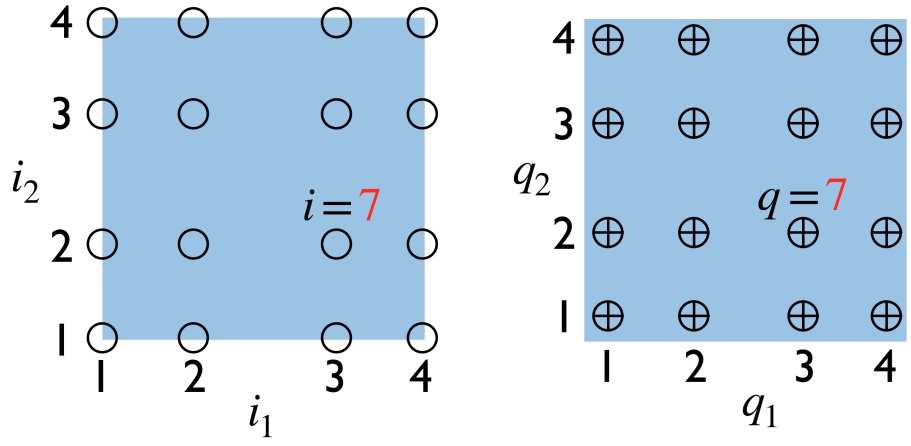
Classically, in order to solve (18) the matrix $\boldsymbol{A}$ and the residual force vector $\boldsymbol{F}$ corresponding to the discretization are assembled (that is, the non-zero matrix elements A_{ij} are individually computed and stored) and a direct or iterative solver is used to solve the linear system. In this case, the matrix-vector product $\boldsymbol{Ax}$ results from the product of the individual matrix elements with the elements in the vector, and is usually implemented with the aid of specialized linear algebra packages. On modern computer architectures however, loading sparse matrix data into the CPU registers is significantly slower than performing the arithmetic operations when solving. For this reason, recent implementations often focus on so-called matrix-free approaches where the matrix is not assembled but rather the result of its operation on a vector is directly calculated. The idea is to perform the operations within the solver on-the-fly rather than loading matrix elements from memory. For iterative solvers, this is possible since it is sufficient to compute matrix-vector products alone. The matrix-vector product $\boldsymbol{Ax}$ (i.e. the action of operator $\boldsymbol{A}$ on a vector $\boldsymbol{x}$) can be expressed by

$$\begin{aligned}(\boldsymbol{Ax})_i &= \sum_j a(\boldsymbol{N}_i, \boldsymbol{N}_j) x_j \\ &\approx \sum_K \sum_q \sum_j \tilde{a}(\boldsymbol{N}_i, \boldsymbol{N}_j)(\boldsymbol{\xi}_q) x_j w_q J_q^K\end{aligned} \tag{21}$$

where $\tilde{a}(\boldsymbol{N}_i, \boldsymbol{N}_j)(\boldsymbol{\xi}_q)$ is a representation for the evaluation of the bilinear form in one quadrature point $\boldsymbol{\xi}_q$, J_q^K is the Jacobian of the mapping from the isoparametric element to the element K and w_q is the quadrature weight. For the sake of demonstration, let us assume that $\boldsymbol{A}$ represents a mass operator with scalar valued shape functions $\{N_i(\boldsymbol{x})\}$. In this case, it holds that

$$\tilde{a}(N_i, N_j)(\boldsymbol{\xi}_q) = N_i(\boldsymbol{\xi}_q) N_j(\boldsymbol{\xi}_q).$$

Further assuming that the number of quadrature points n in one-dimension matches the degree of the ansatz space plus one, the evaluation of all of the shape functions (n^d) in all quadrature points (n^d) has complexity $O(n^{2d})$ in d space dimensions. For the total matrix-vector product we get $O(n^{2d})$, and this is the same as for a regular matrix-vector product where the entries are already precomputed. Assuming tensor product ansatz spaces and a tensor product quadrature rule, evaluation of this operations can be performed more efficiently using a technique called "sum factorization".



(a) Nodes of cubic Lagrange element (b) Gaussian quadrature points

Figure 2: Illustration of tensorial indexing for nodes and quadrature points. In this sketch a node/quadrature point with the number 7 can be indexed with a multi-index $\{3, 2\}$.

Let us illustrate this in 2D. Tensor-product quadrature points can be expressed as a combination of one dimensional quadrature points

$$\boldsymbol{\xi}_q = (\widetilde{\xi}_{q_1}, \widetilde{\xi}_{q_2}), \tag{22}$$

where for each quadrature points q we can associate a multi-index (q_1, q_2). Weights associated with the quadrature formula w_q can be expressed as a product of one-dimensional weights

$$w_q = \widetilde{w}_{q_1} \widetilde{w}_{q_2} \,. \tag{23}$$

Shape functions can also be expressed as product of one dimensional basis functions

$$N_i(\boldsymbol{\xi}_q) = \widetilde{N}_{i_1}(\widetilde{\xi}_{q_1}) \widetilde{N}_{i_2}(\widetilde{\xi}_{q_2}) \tag{24}$$

where for each DoF index i we can associate a multi-index (i_1, i_2). See Figure 2 for an illustration. Therefore, the result of the application of the scalar-valued mass operator, represented as a vector y_i, on a single element K can be written as

$$\begin{aligned} y_i \equiv Y_{i_1\, i_2} &= \sum_{q_1} \sum_{q_2} \sum_{j_1} \sum_{j_2} \widetilde{N}_{i_1}(\widetilde{\xi}_{q_1}) \widetilde{N}_{i_2}(\widetilde{\xi}_{q_2}) \widetilde{N}_{j_1}(\widetilde{\xi}_{q_1}) \widetilde{N}_{j_2}(\widetilde{\xi}_{q_2}) X_{j_1\, j_2} \widetilde{w}_{q_1} \widetilde{w}_{q_2} J^K_{q_1\, q_2} \\ &= \sum_{q_1} \widetilde{N}_{i_1}(\widetilde{\xi}_{q_1}) \widetilde{w}_{q_1} \sum_{q_2} \widetilde{N}_{i_2}(\widetilde{\xi}_{q_2}) \widetilde{w}_{q_2} J^K_{q_1\, q_2} \left[\sum_{j_1} \widetilde{N}_{j_1}(\widetilde{\xi}_{q_1}) \sum_{j_2} \widetilde{N}_{j_2}(\widetilde{\xi}_{q_2}) X_{j_1\, j_2} \right] \forall i_1\, i_2 \end{aligned}$$

Note that $x_j \equiv X_{j_1\, j_2}$; the former accesses the element source vector using a linear index j, whereas the latter uses the associated multi-index accesses $\{j_1\ j_2\}$. These four loops all have the same structure. We either keep the shape function fixed and iterate over the quadrature points in one spatial direction (in the first two sums) or keep the quadrature point fixed and iterate over all the shape functions in one spatial direction (in the last two sums). Since each of these $2 \times d$ loops has a complexity of n and we perform them for n^d values in quadrature points or values in degrees of freedoms simultaneously, this results in an algorithm with an arithmetic complexity of $O(n^{d+1})$. Hence, we can expect that such a matrix-free approach is faster than a regular matrix-based approach, especially for the 3D case, and where the discretization and evaluation employs high polynomial degree basis functions with the corresponding quadrature rule. More information on matrix-free techniques can be found in [1, 20].

For the performance of the matrix-free operator, it is crucial to find a good intermediate point between precomputing everything (caching) and evaluating all the quantities on-the-fly. In order to obtain gradients with respect to the spatial configuration, that appear in the tangent operator (19) of the finite-strain elasticity problem derived with a Lagrangian formulation, we used the *MappingQEulerian* class from the `deal.II` [11] library. In addition to the standard mapping from the reference (isoparametric) element to the element in real space, this class computes the mapping to the spatial configuration given a displacement field. Due to the specifics of matrix-free operator evaluation implemented in `deal.II`, the integration is eventually performed over the spatial configuration. Therefore, we have to additionally divide by the Jacobian J of the deformation map at a given quadrature point to express the integral in the referential configuration.

Below, we propose three algorithms to implement the tangent operator of finite-strain elasticity (see (19)) using the matrix-free approach, each of which have a different level of abstraction, algorithmic complexity, and memory requirements. They are introduced in order of lowest memory footprint to largest:

Algorithm 1 caches a single scalar which involves the logarithm of the Jacobian – a computationally demanding operation compared to additions and multiplications. As evident from Section 2, we need to re-evaluate the Kirchhoff stress at each quadrature point in order to apply the tangent operator. This in turn requires evaluating the deformation gradient $\boldsymbol{F}$ and therefore gradients with respect to the referential configuration.

In order to avoid recalculation of the Kirchhoff stress, Algorithm 2 caches its value for each element and quadrature point. In this case, we also avoid the need to evaluate gradients of the displacement field with respect to the referential

Algorithm 1: Matrix-free tangent operator: cache scalar quantities only

Given : Source FE vector $\boldsymbol{x}$, current FE solution $\overline{\boldsymbol{u}^h}$, cached $c_1 := \mu - 2\lambda \log(J)$ for each cell and quadrature point

Return: action of the FE tangent operator (19) on $\boldsymbol{x}$

foreach *element* $K \in \Omega^h$ **do**
- extract local vector values on this element $\overline{\boldsymbol{u}^h}_K$, $\boldsymbol{x}_K$;
- evaluate the following tensors at each quadrature point using sum factorization :
  - Grad $\overline{\boldsymbol{u}^h}_K$; `// 2nd order`
  - $\boldsymbol{g}_s := \overline{\text{grad}^s \boldsymbol{x}}_K$; `// 2nd order symmetric`
  - $\boldsymbol{g} := \overline{\text{grad}\boldsymbol{x}}_K$; `// 2nd order`
  - **foreach** *quadrature point q on K* **do**
    - evaluate $\boldsymbol{F} = \boldsymbol{I} + \text{Grad}\,\overline{\boldsymbol{u}^h}$; `// 2nd order`
    - evaluate $J = \det(\text{F})$; `// scalar`
    - evaluate $\boldsymbol{b} = \boldsymbol{F} \cdot \boldsymbol{F}^T$; `// 2nd order symmetric`
    - evaluate $\boldsymbol{\tau} = \mu \boldsymbol{b} - c_1 \boldsymbol{I}$; `// 2nd order symmetric`
    - evaluate $\boldsymbol{\mathcal{G}g} := \boldsymbol{g} \cdot \boldsymbol{\tau}/J$; `// 2nd order`
    - queue $\boldsymbol{\mathcal{G}g}$ for contraction $\overline{\text{grad}\boldsymbol{N}_i} : \boldsymbol{\mathcal{G}g}$;
    - evaluate $\boldsymbol{Cg}_s := [2c_1 \boldsymbol{g}_s + 2\lambda\,\text{tr}(\text{g}_\text{s})\text{I}]\,/J$; `// 2nd order symmetric`
    - queue $\boldsymbol{Cg}_s$ for contraction $\overline{\text{grad}^s \boldsymbol{N}_i} : \boldsymbol{Cg}_s$;
  - **end**
  - evaluate queued contractions using sum factorization ;
  - distribute results to the destination vector

end

configuration. This algorithm also utilizes the chosen constitutive relationship in that the operation of $\boldsymbol{JC}$ remains directly expressed using (16). Therefore, the action of the material part of the fourth-order spatial tangent stiffness tensor $\boldsymbol{JC}$ on the second-order symmetric tensor $\overline{\text{grad}^s \boldsymbol{x}}$ is cheap to evaluate. To that end, we cache two scalars that depend on the Jacobian J.

Finally, Algorithm 3 represents the most general case which does not assume any form of the material part of the fourth-order spatial tangent stiffness tensor. In this case, we have to cache the fourth-order symmetric tensor $\boldsymbol{C}$ and the Kirchhoff stress $\boldsymbol{\tau}$ for each element and quadrature point, and perform the double contraction

Algorithm 2: Matrix-free tangent operator: cache second-order Kirchhoff stress $\boldsymbol{\tau}$ and thereby avoid the need to evaluate referential quantities like $\boldsymbol{F}$ at when executed.

Given : Source FE vector $\boldsymbol{x}$, cached $c_1 := 2\left[\mu - 2\lambda\log(J)\right]/J$, $c_2 := 2\lambda/J$ and $\boldsymbol{\tau}/J$ for each cell and quadrature point, evaluated based on $\overline{\boldsymbol{u}^h}$

Return: action of the FE tangent operator (19) on $\boldsymbol{x}$

foreach *element* $K \in \Omega^h$ **do**
- extract local vector values on this element $\boldsymbol{x}_K$;
- evaluate the following tensors at each quadrature point using sum factorization :
  - $\boldsymbol{g}_s := \overline{\mathrm{grad}^s}\boldsymbol{x}_K$; `// 2nd order symmetric`
  - $\boldsymbol{g} := \overline{\mathrm{grad}}\boldsymbol{x}_K$; `// 2nd order`
- **foreach** *quadrature point q on K* **do**
  - evaluate $\boldsymbol{\mathcal{G}g} := \boldsymbol{g}\cdot[\boldsymbol{\tau}/J]$; `// 2nd order`
  - queue $\boldsymbol{\mathcal{G}g}$ for contraction $\overline{\mathrm{grad}}\boldsymbol{N}_i : \boldsymbol{\mathcal{G}g}$;
  - evaluate $\boldsymbol{C}\boldsymbol{g}_s := c_1\boldsymbol{g}_s + c_2\,\mathrm{tr}(\mathrm{g_s})\mathrm{I}$; `// 2nd order symmetric`
  - queue $\boldsymbol{C}\boldsymbol{g}_s$ contraction $\overline{\mathrm{grad}^s}\boldsymbol{N}_i : \boldsymbol{C}\boldsymbol{g}_s$;
- **end**
- evaluate queued contractions using sum factorization ;
- distribute results to the destination vector

end

with the second-order symmetric tensor $\overline{\mathrm{grad}^s}\boldsymbol{x}$ on-the-fly.

Note that the SIMD vectorization in `deal.II` [11] is applied at the finite element level, i.e. the matrix-free operator is applied simultaneously on several elements (called "blocks"). The main reason is that, apart from the point-wise computation of the stresses and elastic tangents, the operations are typically the same on all elements[3]. For a given quadrature point, we simultaneously perform

[3] In general, not all quadrature points in each element group are endowed with identical, non-dissipative constitutive laws that follow the same code path during evaluation or the kinetic quantities and tangents. This motivates the caching strategy employed in Algorithm 3, as the pre-caching of the chosen data ensures a context in which SIMD parallelization of subsequent operations is guaranteed. Algorithms 1 and 2 sacrifice some of this generality for decreased memory requirements.

Algorithm 3: Matrix-free tangent operator: cache material part of the fourth-order spatial tangent stiffness tensor $\boldsymbol{C}$ and Kirchhoff stress $\boldsymbol{\tau}$.

Given : Source FE vector $\boldsymbol{x}$, cached $\boldsymbol{\tau}/J$ and $\boldsymbol{C}$ for each cell and quadrature point, evaluated based on $\overline{\boldsymbol{u}^h}$

Return: action of the FE tangent operator (19) on $\boldsymbol{x}$

foreach *element* $K \in \Omega^h$ **do**
- extract local vector values on this element $\boldsymbol{x}_K$;
- evaluate the following tensors at each quadrature point using sum factorization :
  - $\boldsymbol{g}_s := \overline{\text{grad}^s \boldsymbol{x}_K}$; `// 2nd order symmetric`
  - $\boldsymbol{g} := \overline{\text{grad} \boldsymbol{x}_K}$; `// 2nd order`
- **foreach** *quadrature point q on K* **do**
  - evaluate $\boldsymbol{\mathcal{G}g} := \boldsymbol{g} \cdot [\boldsymbol{\tau}/J]$; `// 2nd order`
  - queue $\boldsymbol{\mathcal{G}g}$ for contraction $\overline{\text{grad} \boldsymbol{N}_i} : \boldsymbol{\mathcal{G}g}$;
  - evaluate $\boldsymbol{Cg}_s := \boldsymbol{C} : \boldsymbol{g}_s$; `// 2nd order symmetric`
  - queue $\boldsymbol{Cg}_s$ for contraction $\overline{\text{grad}^s \boldsymbol{N}_i} : \boldsymbol{Cg}_s$;
- **end**
- evaluate queued contractions using sum factorization ;
- distribute results to the destination vector

end

tensor evaluations and contractions on all elements[4]. For example, the deformation gradient $\boldsymbol{F} = \boldsymbol{I} + \text{Grad}\,\overline{\boldsymbol{u}^h}$ (second-order tensor) is evaluated at the specific quadrature point of all elements within the "block". This is achieved using templated number types within the *Tensor* class of the `deal.II` library. In particular, `deal.II` provides the *VectorizedArray* templated generic class that defines a unified interface to a vectorized data type and overloads basic arithmetic operations based on compiler intrinsics. The core of the sum factorization algorithms is provided by the *FEEvaluation* class, that already supports all the necessary contraction (such as $\overline{\text{grad}^s \boldsymbol{N}_i} : \boldsymbol{Cg}_s$) to implement the three algorithms proposed above. Thanks to the object-oriented design of the `deal.II` library, the core of those

[4]For the heterogeneous materials considered here, the elements are first grouped according to the material type (matrix or inclusion) and then the SIMD vectorization is applied for each group of elements. This functionality is provided by the `deal.II` library in the *MatrixFree* class.

matrix-free algorithms can be implemented in only a handful of lines. Distributed memory MPI parallelization of the matrix-free operators is done using the standard domain decomposition technique, where each MPI process is responsible for applying the operator only on the so-called locally owned elements. For more information about the implementation details and data structures for sum factorization approaches in `deal.II` (including hanging constraints, storage of the inverse Jacobian of the transformation from unit to real cell, MPI and distributed memory parallelization as well SIMD vectorization) and performance tests, see [1].

6. Geometric multigrid preconditioning

Efficient and scalable matrix-vector products are not enough to obtain a method suitable for large computations. In particular, linear scaling with respect to the number N of unknown DoFs is desirable. The efficiency of standard iterative solvers and preconditioners deteriorates for large systems of linear equations due to increasing iteration counts under mesh refinement. In contrast to other preconditioners, geometric multigrid (GMG) preconditioners [2, 21–23] can be proven to result in iteration counts that are independent of the number of mesh refinements by smoothing the residual on a hierarchy of meshes. For further information about geometric multigrid methods we refer the reader to [22, 24, 25]. Implementational details of the MPI-parallel multigrid method in `deal.II` are discussed in [8].

For this work, we adopt the matrix-free level operators within the GMG. In general, given a heterogeneous non-linear material, the definition of transfer operators[5] (for restriction and prolongation) and level operators is not trivial. One approach is to start from an arbitrary heterogeneous material at the fine scale and employ homogenization theory [26–29] to design suitable transfer operators [30][6]. In this approach, the fine-scale displacement is decomposed into long-wave and short-wave contributions by an additive split, where the former is associated to the homogeneous response of a patch of elements and the latter represents fluctuations due to the heterogeneous structure of the patch. However, for a patch of elements consisting of the same material, the transfer operators reduce to the standard geometric transfer. Based on this observation we adopt the following approach: It is assumed that the mesh at the coarsest level can accurately describe the hetero-

[5] Finite dimensional linear operators from a vector space associated with the fine-scale mesh to the coarse-scale mesh, and in the opposite direction.

[6] We note that this approach is limited to the case where for each quadrature point within an element the fourth-order tangent stiffness tensor is the same.

geneous finite-strain elastic material. In this case, the restriction and prolongation operations for patches of elements are always performed for the same material and thus admits usage of standard geometric transfer operators. We define the level operators as follows: Recall that the fine scale tangent operator (19) is obtained by linearization around the current displacement field $\boldsymbol{u}^h$. We then restrict this displacement field to all multigrid levels $\{l\}$ and evaluate tangent operators using the matrix-free algorithms from Section 5. Essentially, level operators correspond to the linearization around the smoothed representation of displacement field[7].

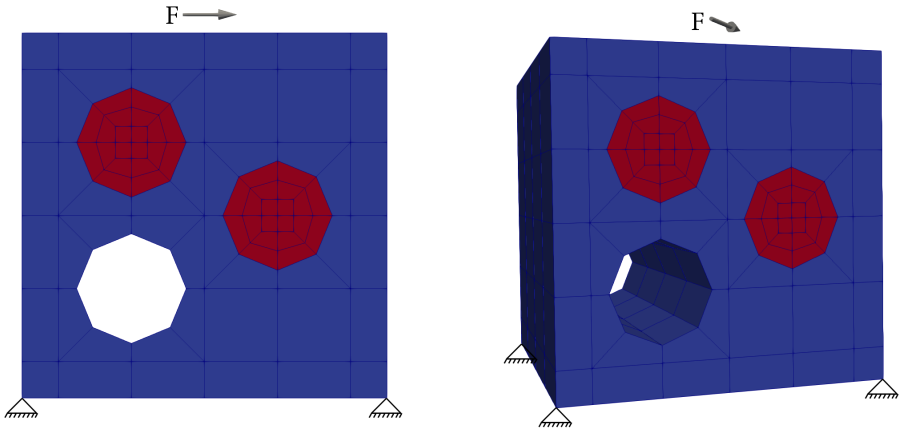
To summarize, in this section we propose a method by which to construct level tangent operators for finite-strain elasticity to be employed within the GMG preconditioner. The other parts of the GMG preconditioner such as level smoothers, matrix-free interlevel transfer operators and the coarse level iterative solver are provided by the `deal.II` library. The efficiency of the preconditioner with the considered level operators will be assessed by numerical examples in the next section. We will, however, refrain from studying the machine performance aspects of the multigrid preconditioner as a whole as the focus of this paper is the matrix-free implementation of tangent operators of finite-strain elasticity both at the fine mesh level as well as the coarser multigrid levels. Finally, we note that at the time of writing, a $p-$ version of the GMG was not available within the `deal.II` library and therefore not considered here.

For examples of other approaches that combine homogenization and multigrid methods, see [31, 32] for applications using small strain elastic materials and [33] with applications in fracture.

7. Numerical examples

In this section, we apply the proposed matrix-free operator algorithms for finite-strain hyperelastic materials as well as the geometric multigrid preconditioner to a benchmark problem from [30]. The coarse meshes for the 2D and 3D problems are illustrated in Figure 3. The 2D material consists of a square (depicted in blue), a hole and two spherical inclusions (depicted in red). The size of the square domain is 10^{-3}mm. The matrix material is taken to have Poisson's ratio 0.3 and shear modulus $\mu = 0.4225 \times 10^6\,\mathrm{N/mm^2}$. The inclusion is taken to

[7]Note that this differs from the classical approach where the (tangent) operator $\boldsymbol{A}^{l+1}$ on level $l+1$ is directly related to the (tangent) operator $\boldsymbol{A}^l$ on level l via $\boldsymbol{A}^{l+1} = \boldsymbol{I}_l^{l+1}\boldsymbol{A}^l\boldsymbol{I}_{l+1}^l$, where $\boldsymbol{I}_{l+1}^l$ and $\boldsymbol{I}_l^{l+1}$ are global prolongation (coarse-to-fine) and restriction (fine-to-coarse) operators, respectively.



(a) 2D coarse mesh (b) 3D coarse mesh

Figure 3: Discretization of the heterogeneous material at the coarsest mesh level and the prescribed boundary conditions.

be 100 times stiffer. The 3D material is obtained by extrusion of the 2D geometry into the third dimension. Note that although a relatively coarse mesh is used at the coarsest multigrid level, the geometry of the inclusions is captured accurately by manifold descriptions of the boundaries and interfaces. Both domains are fully fixed along the bottom surface and a distributed load is applied at the top in the $(1, 0)$ or $(1, 1, 0)$ direction for the 2D and the 3D problems, respectively. The force density $12.5 \times 10^3 \mathrm{N/mm}^2$ or $12.5\sqrt{2} \times 10^3 \mathrm{N/mm}^3$ is applied in 5 steps for the 2D and the 3D problems, respectively. With respect to the nonlinear solver, the displacement tolerance of the ℓ_2 norm for the Newton update is taken to be 10^{-5} whereas the relative and absolute tolerances for the residual forces is 10^{-8}. The relative convergence criteria for the linear solver is 10^{-6}. Figure 4 shows deformed meshes at the final loading step with quadratic elements and two global mesh refinements. The same distributed approach to mesh decomposition (as most evident on the finest level) is adopted on the multigrid levels. Figure 5 illustrates the hierarchy of multigrid meshes for the 2D problem after two global refinements for the mesh introduced in figure 1.

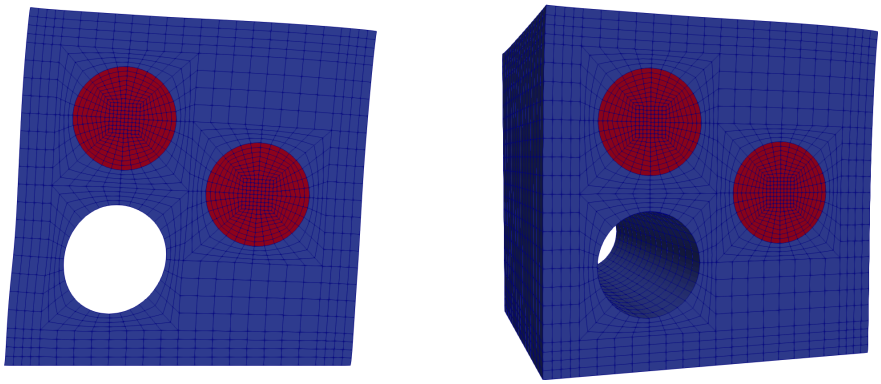
Aligned with the approach taken in similar previous investigations, compare e.g. [9], we limit ourselves to comparisons made on a node level in this study. Examinations performed in this manner should already indicate the differences between the matrix-free and matrix-based approaches, and give some preliminary understanding of the scalability potential of the numerical implementation without

p	q	N_{gref}	N_{el}	N_{DoF}
1	2	7	1441792	2887680
2	3	6	360448	2887680
3	4	5	90112	1625088
4	5	5	90112	2887680
5	6	5	90112	4510720
6	7	4	22528	1625088
7	8	4	22528	2211328
8	9	4	22528	2887680

Table 1: Parameters for the 2D benchmark: p is the polynomial degree, q is the number of quadrature points in 1D, N_{gref} is the number of global mesh refinements, N_{el} is the number of elements and N_{DoF} is the number of DoFs.

p	q	N_{gref}	N_{el}	N_{DoF}
1	2	4	1441792	4442880
2	3	3	180224	4442880
3	4	2	22528	1891008
4	5	2	22528	4442880

Table 2: Parameters for the 3D benchmark: p is the polynomial degree, q is the number of quadrature points in 1D, N_{gref} is the number of global mesh refinements, N_{el} is the number of elements and N_{DoF} is the number of DoFs.



(a) 2D deformed mesh (b) 3D deformed mesh

Figure 4: Deformed meshes at the final loading step with quadratic elements and two global mesh refinements.

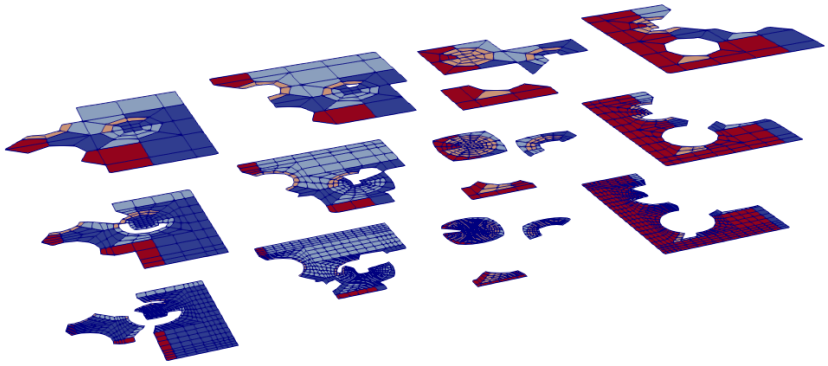


Figure 5: 2D multigrid mesh after two global refinements distributed into four MPI processes. The color indicates the process owning the element.

confounding factors related to data exchange through interconnects via MPI. The benchmark calculations are performed for 2D and 3D problems with various combinations of polynomial degrees and number of global refinements, as is stated in Tables 1 and 2. We chose a problem size around 10^6 DoFs for each test case as this was an upper limit due to the memory requirements of the matrix-based approach with a high polynomial order finite element. Even so, the problem size is large enough so that the sparse matrix (in excess of 1 Gb when using the lowest polynomial order finite element discretization) will not fit into the L3 cache (on the

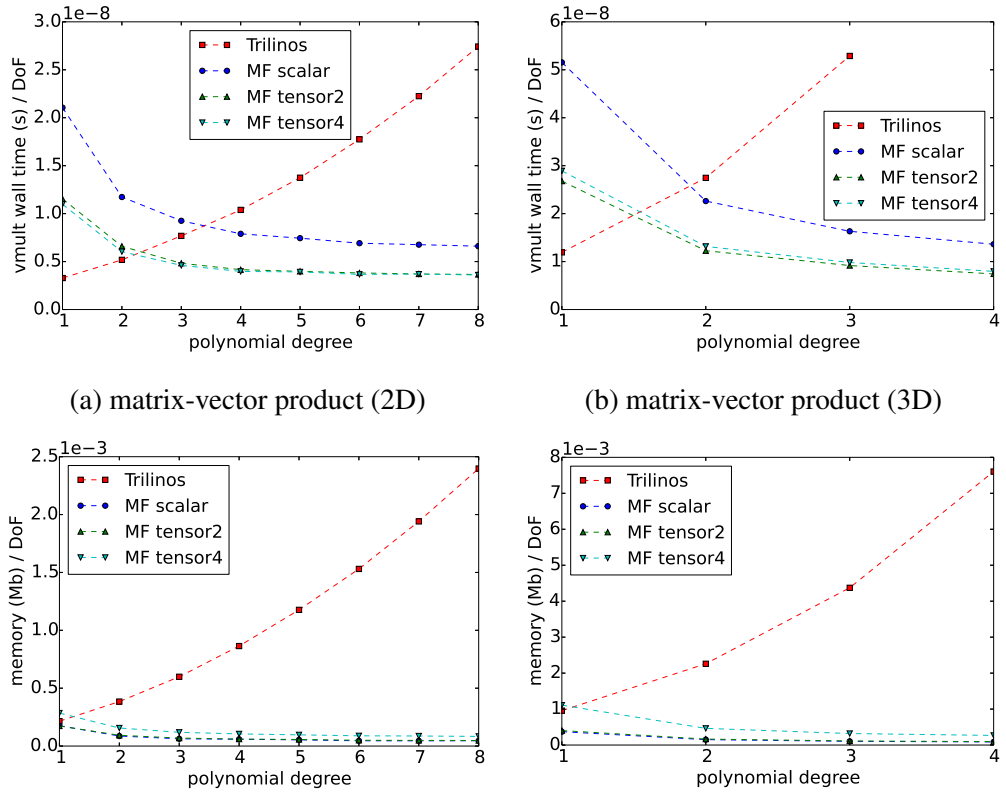
order of tens of Mb in size), and therefore we can observe that the matrix-based approach is memory bound.

We conduct two performance studies using the described discretization, the first being for only matrix-vector multiplication, and the second for the preconditioned iterative linear solver. The results of the matrix-free approach with a geometric multigrid preconditioner are compared to the matrix-based approach in conjunction with the algebraic multigrid (AMG) preconditioner using the package ML [19] of the Trilinos [17] library version 12.12.1. The aggregation threshold for the AMG preconditioner is taken as 10^{-4}. The geometric multigrid algorithm uses a V-cycle with two pre- and post-smoothing steps using Chebyshev polynomials [34] of fourth degree, provided by the `deal.II` library via the *PreconditionChebyshev* class. The largest eigenvalue $\lambda_{\max}$ of the level operators is estimated from 30 iterations of the CG solver and the smoother is set to target the range $[0.6\lambda_{\max}, 1.2\lambda_{\max}]$. On the coarsest level, we use the Chebyshev preconditioner as a solver [34] to reduce the residual by three orders of magnitude. Since this smoother only uses diagonal elements of the operator it is readily compatible with the matrix-free approach while most default smoothers rely on an explicit matrix form.

Computations were performed on a single node of two clusters: A node on the "Emmy" cluster at RRZE, FAU has two Xeon 2660v2 "Ivy Bridge" chips (10 cores per chip + SMT) running at 2.2 GHz with 25 MB shared cache per chip and 64 GB of RAM. Intel's compiler version 18.03 with flags "-O3 -march=native" and Intel MPI version 18.03 are used. The "IWR" results were obtained on a single machine with eight Xeon E7-8870 "Sandy Bridge" chips (10 cores per chip + SMT) running at 2.4 GHz with 30 MB shared cache per chip. For the simulation, GCC version 8.1.0 with flags "-O3 -march=native" and OpenMPI version 3.0.0 were used. Unless noted otherwise, the simulations are performed with 20 MPI processes, i.e. one fully occupied node.

7.1. Matrix-vector multiplication

Figures 6 and 7 show the results of the matrix-vector multiplication for the considered finite-strain hyperelastic benchmark problem as performed on the two clusters. We are interested in the wall-clock time (average over 10 runs per each linear solver step) per DoF as a first metric. As expected, the matrix-vector multiplication becomes very expensive for higher order discretization for sparse matrix-based approaches, as the matrix becomes more and more dense. The performance results for finite-strain elasticity operators considered in this study are qualitatively similar to those reported for the Laplace operator in [1]. Note that we



(a) matrix-vector product (2D)

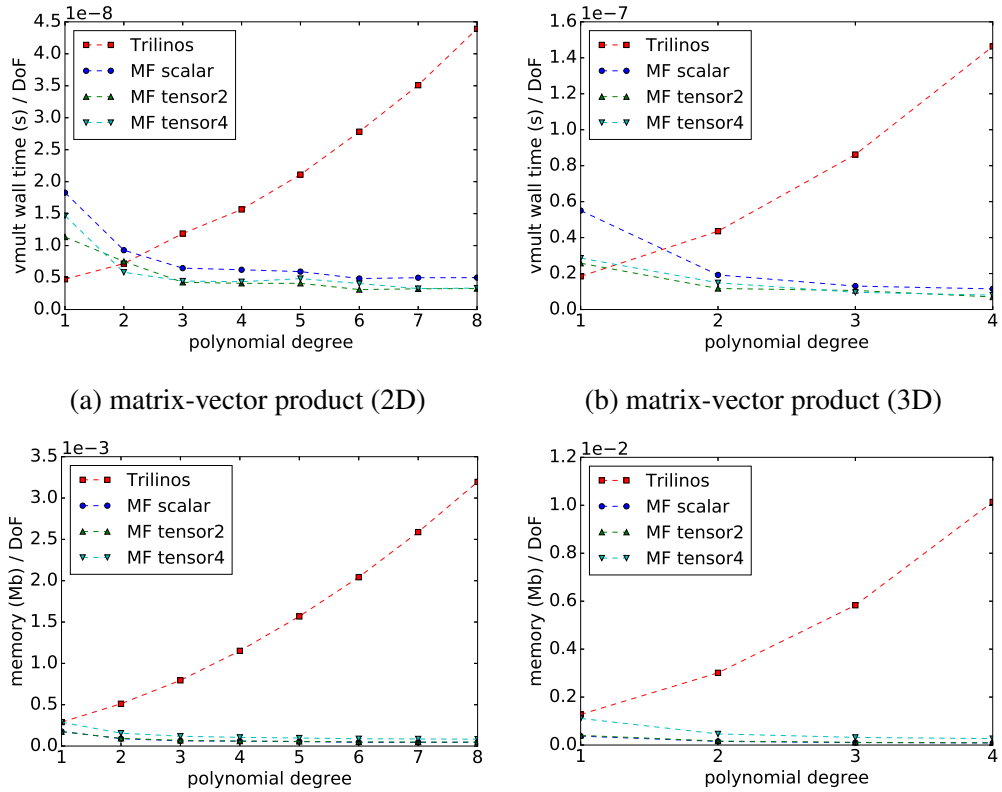
(b) matrix-vector product (3D)

(c) memory consumption (2D)

(d) memory consumption (3D)

Figure 6: Emmy cluster, RRZE, matrix-vector multiplication.

consider only non-Cartesian meshes together with additional mappings (namely, manifold descriptions of the boundaries and interfaces [35] to capture the geometry as well as the linearization-related mapping required for evaluation of gradients with respect to the deformed configuration). Both factors increase the memory consumption and make element operations more expensive. Nonetheless, the matrix-free implementation is faster than the matrix-based counterpart already for bi-cubic Lagrangian basis in 2D and tri-quadratic in 3D. Figures 6(a), 6(b), 7(a) and 7(b) clearly show the influence of the different caching strategies, described in Algorithms 1, 2 and 3. Two conclusions can be drawn from these results: The scalar caching implementation is the most time consuming of the three, as at each step we additionally evaluate the gradient of the displacement field in the referential configuration that is required to evaluate the Kirchhoff stress $\boldsymbol{\tau}$. There is little



(a) matrix-vector product (2D)

(b) matrix-vector product (3D)

(c) memory consumption (2D)

(d) memory consumption (3D)

Figure 7: IWR cluster, matrix-vector multiplication

difference in terms of the wall-clock time per DoF between the approach where the material part of the fourth-order spatial tangent stiffness tensor is cached ("tensor4") and the one where we only cache the second-order Kirchhoff tensor and utilize the chosen hyperelastic constitutive equation to efficiently implement the action of the material part of the fourth-order spatial tangent stiffness tensor on the second-order symmetric tensor ("tensor2"). This indicates that the time savings we get from the latter approach are insignificant within the complete matrix-free operator implementation. Consequently, we can apply the matrix-free approach to any finite-strain material model by caching the material part of the fourth-order spatial tangent stiffness tensor in addition to the Kirchhoff stress and expect this to be faster than the matrix-based implementation. It is therefore feasible to define highly performant generic operators that are independent of any applied constitu-

tive laws, as long as the resulting tangent operator is expressed via (19).

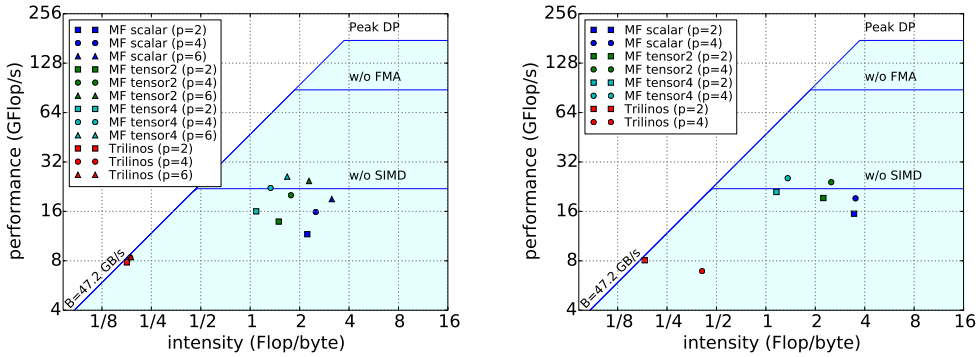
As outlined in the introduction, it is not only the performance of the matrix-free vector multiplication, but also the memory requirement to store a sparse matrix which is the driving force behind matrix-free methods. Figures 6(c), 6(d), 7(c) and 7(d) demonstrate that even with caching one fourth-order symmetric tensor and one second-order tensor at each quadrature point ("tensor4"), the matrix-free approach takes much less memory than its matrix-based counterpart. In those graphs, we report the overall memory needed to cache additional quantities required for the tangent operator here considered, as well as the memory required by the `deal.II` *MatrixFree* object, which caches the inverse of the Jacobian of the transformation from unit to real cell [1]. Note that for Algorithm 1 we need to evaluate gradients with respect to both deformed and undeformed configurations, and therefore two inverses will be cached via two different *MatrixFree* objects.

Finally, we analyze the node-level performance of the two approaches using the roofline model. Memory bandwidth (MBytes/s) and the number of floating point operations per second (MFLOP/s) are measured using the `MEM_DP` group of the LIKWID [36] tool, version 4.2.1. The measurements are done on a single socket of RRZE cluster by pinning all 10 MPI processes to this socket, whose memory bandwidth[8] and peak performance are $B = 47.2\,\mathrm{GB/s}$ and $P = 176\,\mathrm{GFlop/s}$, respectively. Turbo mode of the CPU was disabled.

Figure 8 shows results of the roofline performance model. As expected, the matrix-based approach is memory bound and (on average) achieves the performance of 8.21 GFlop/s (4.6% of the peak performance) in 2D. The matrix-free implementations of the fine-strain elastic tangent operator (on average) lead to a much better performance of 18.8 GFlop/s in 2D, as well as about an order of magnitude higher computational intensity, but still stays in the memory-bound regime. The achieved performance is about 10.6% of the peak arithmetic performance. This is much lower than the 70% reported for the Laplace operator on Cartesian meshes in [1], but similar to what was obtained in [9, Figure 14] when not optimizing for Cartesian meshes. Unfortunately, performance measurements for non-Cartesian meshes were not reported in [1].

When comparing the data for "tensor4" caching strategy (Algorithm 3) to that from the "tensor2" (Algorithm 2) caching strategy, we observe a marginal increase in computational intensity and almost unchanged performance. This coincides with our expectation that the latter algorithm requires transferring fewer bytes to

[8] As measured with the *load_avx* benchmark of *likwid-bench* [36].

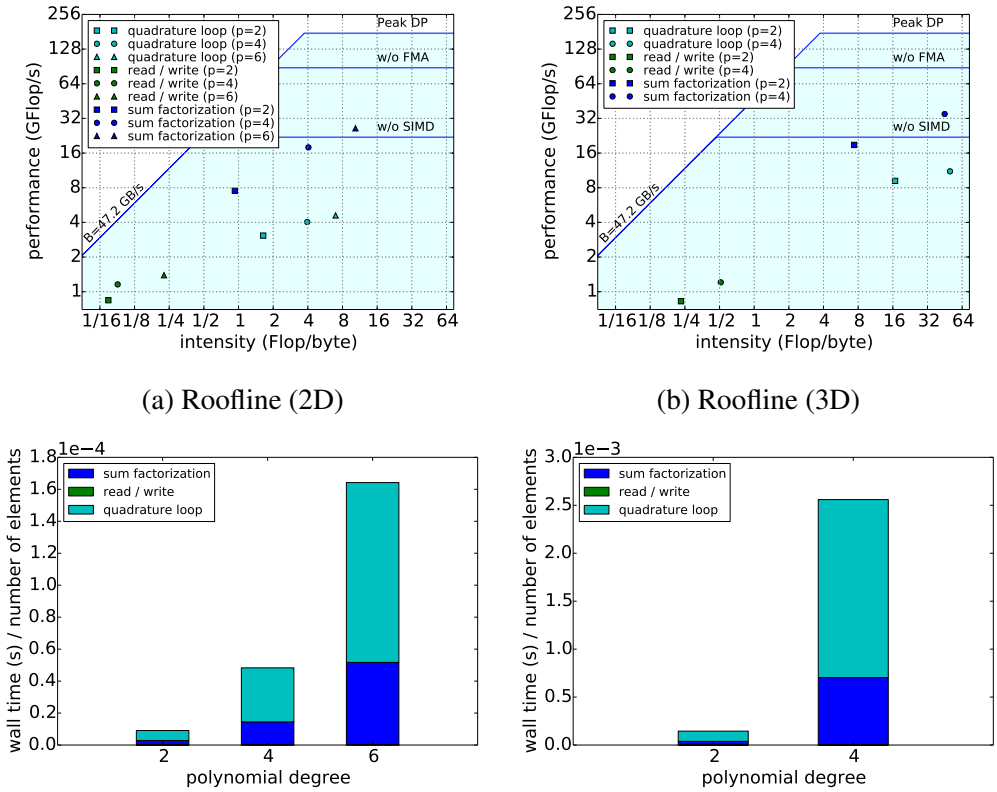


(a) 2D (b) 3D

Figure 8: Roofline model for a selection of polynomial degrees. 'Peak DP' denotes the peak double precision performance (P), 'w/o FMA' represents a ceiling without Fused Multiply-Add ($P/2$), finally 'w/o SIMD' is a ceiling where both FMA and SIMD vectorization are discarded ($P/8$).

perform operations on a single quadrature point (compare line 9 in Algorithm 3 that loads a symmetric rank-4 tensor to the line 9 in Algorithm 2 that evaluates the same contraction by loading two scalars c_1 and c_2). On the other hand, the "scalar" caching strategy (Algorithm 1) demonstrates a higher computational intensity. This is not surprising as at each quadrature point we need to re-evaluate the Kirchoff stress $\boldsymbol{\tau}$. Given that in this case we need to evaluate gradients with respect to both deformed and undeformed configurations, an additional second-order tensor per quadrature point (inverse of the Jacobian transformation) will be cached as compared to the Algorithm 2. As a result both algorithms will have comparable memory requirements (also confirmed by Figures 6(c), 6(d), 7(c) and 7(d)), however the "scalar" algorithm performs a large number of repetitive operations that are independent on the right hand side vector (steps 8–11 in Algorithm 1), which makes it the worst option among the three.

In order to have a better understanding of the behavior of the most promising "tensor4" implementation, we collect its performance data and computing times for various stages inside the element loop (see Algorithm 3): (i) 'read / write' denotes reading and writing from/to a global vector (lines 2 and 13); (ii) 'sum factorization' denotes application of sum factorization techniques (lines 4, 5 and 12); (iii) 'quadrature loop' denotes operations performed on each quadrature (lines 6 – 11). Given that the measurements are now down within the cell loop, we collected results on smaller meshes by adopting only one global refinement in 2D and no global mesh refinement in 3D. It is clear from Figure 9(c) and 9(d)



(a) Roofline (2D)

(b) Roofline (3D)

(c) Computing times (2D)

(d) Computing times (3D)

Figure 9: Breakdown of computing times and roofline analysis of various steps of Algorithm 3.

that the 'read / write' steps take a negligible amount of time on each element. For higher order bases, the majority of time is spent within the quadrature loop. When analyzing the roofline data (Figure 9(a) and 9(b)), we observe that this part of the algorithm demonstrates rather low performance, which in our opinion is the source of the suboptimal performance of the whole matrix-free matrix-vector product (see Figure 8). In this case the, quadrature loop consists of, essentially, a single contraction between two second order tensors (step 7 in Algorithm 3) and a double contraction between a symmetric second order tensor and a fourth order tensor (step 9 in Algorithm 3).

Table 3 and Table 4 show the speed-up obtained due to explicit SIMD vectorization intrinsics and due to MPI. For these rather small count of MPI processes, the results confirm that the performance gained is independent. Furthermore, the

	serial		MPI			SIMD			MPI+SIMD		
p	time	GFlop/s	time	GFlop/s	speedup	time	GFlop/s	speedup	time	GFlop/s	speedup
2	0.72	0.87	$7.43 \cdot 10^{-2}$	8.57	9.65	0.39	1.65	1.82	$4.14 \cdot 10^{-2}$	16.01	17.31
4	0.48	1.31	$5.11 \cdot 10^{-2}$	12.39	9.32	0.26	2.37	1.85	$2.79 \cdot 10^{-2}$	22.24	17.09
6	0.25	1.55	$2.64 \cdot 10^{-2}$	14.81	9.36	0.13	2.85	1.89	$1.49 \cdot 10^{-2}$	26.05	16.6

Table 3: Wall-clock time in seconds and performance in GFlops of Algorithm 3 in 2D for various combinations of polynomial degrees, vectorization and parallelization.

	serial		MPI			SIMD			MPI+SIMD		
p	time	GFlop/s	time	GFlop/s	speedup	time	GFlop/s	speedup	time	GFlop/s	speedup
2	1.83	1.44	0.19	13.76	9.54	1.25	2.24	1.47	0.13	21.02	13.65
4	1.15	1.75	0.12	16.62	9.5	0.75	2.75	1.53	$8.17 \cdot 10^{-2}$	25.49	14.08

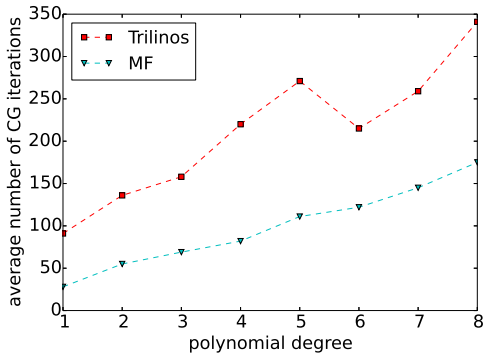
Table 4: Wall-clock time in seconds and performance in GFlops of Algorithm 3 in 3D for various combinations of polynomial degrees, vectorization and parallelization.

improvement due to explicit intrinsics for the data used in the quadrature loop, is comparable to the ones in [37, Figure 6] where the speedup factor is $1.2 - 1.8$ for a similar architecture.

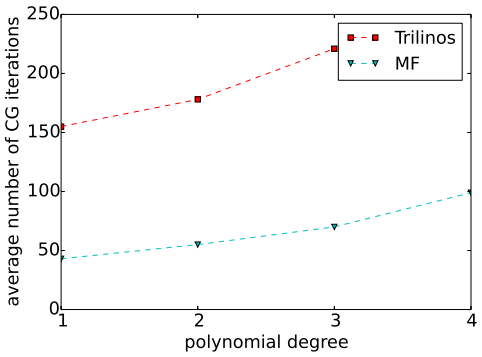
To summarize this section, we can attribute the speed-up of the matrix-free finite-strain tangent operators for higher-order elements to two factors. The first one is the higher algorithmic intensity as compared to the matrix-based implementation, which is clearly memory bound, see Figure 8. The second one is the reduction in the number of floating point operations per DoF thanks to the sum factorization technique, as discussed in Section 5 and [1, Section 2.4]. Among the three suggested implementations, the Algorithm 3 is the most flexible approach with relatively high computational intensity.

7.2. Preconditioned iterative solver

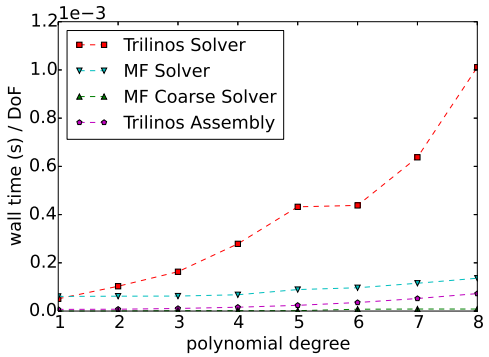
Next, we evaluate the performance of the proposed geometric multigrid preconditioner. Our main goal is to examine whether or not the adopted level operators lead to an efficient preconditioner from the linear algebra perspective. To that end, we consider the average number of CG iterations throughout the entire simulation (i.e. each Newton-Raphson iteration and each loading step). Figures 10(a), 10(b), 11(a) and 11(b) show a noticeable increase in the average number of CG iterations for different polynomial degrees for the GMG preconditioner. Although not reported in this section, we identified that the coarse level solver setup will greatly affect the performance of the preconditioner. In particular, we can improve the number of iterations by choosing a more accurate coarse level solver, however this leads to an overall more expensive linear solver. However, the settings used in this study are optimized for the considered problem in terms of the



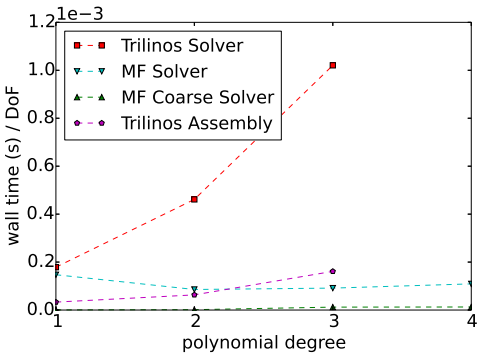
(a) CG iterations (2D)



(b) CG iterations (3D)



(c) CG solution time (2D)

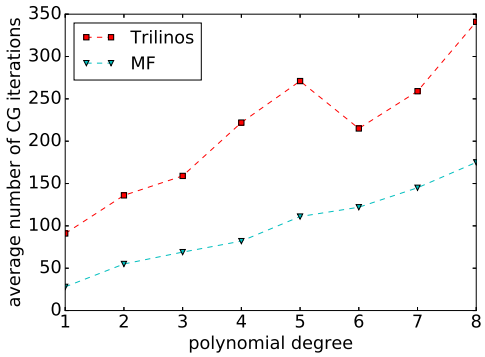


(d) CG solution time (3D)

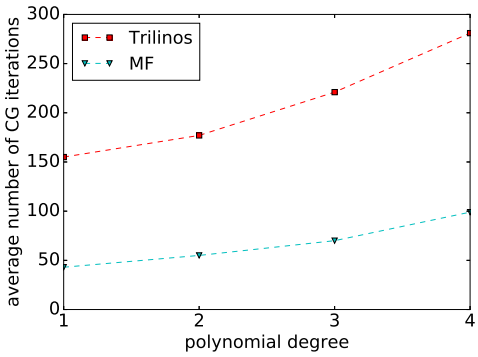
Figure 10: Emmy cluster, RRZE, iterative solver.

wall-time spent in the linear solver. The black-box AMG preconditioner demonstrates the same trend in terms of the number of CG iterations, but requires many more iterations for convergence. Consequently, we can conclude that for the here considered finite strain elasticity problem (i) the selected smoother is suitable, and that (ii) although the adopted level operators are not built using the triple product $\boldsymbol{A}^{l+1} = \boldsymbol{I}_l^{l+1}\boldsymbol{A}^l\boldsymbol{I}_{l+1}^l$, they still result in a multigrid preconditioner which requires much fewer CG iterations as compared to the algebraic multigrid.

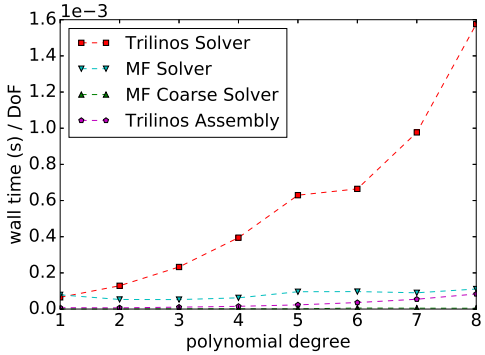
Additionally, we compare solution times of the preconditioned iterative solvers. Our goal is to determine whether the matrix-free approach with a GMG preconditioner can be faster than the matrix-based approach with a well-established implementation of an AMG preconditioner. We note specifically that the comparison is restricted to the choice of level operators used with the pre-existing GMG frame-



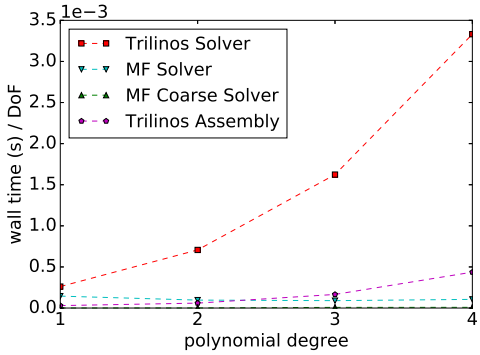
(a) CG iterations (2D)



(b) CG iterations (3D)



(c) CG solution time (2D)



(d) CG solution time (3D)

Figure 11: IWR cluster.

work, and not the framework itself. Figures 10(c), 10(d), 11(c) and 11(d) confirm that the efficiency of the proposed GMG preconditioner translates into faster solution times for all polynomial degrees. Most importantly, the wall time per DoF for the matrix-free solver with GMG preconditioner stays almost constant both for 2D and 3D problems. In comparison, the wall time of the matrix-based iterative solver grows rapidly with increasing polynomial degree. Clearly this is related both to the performance of the preconditioner from the linear algebra perspective as well as the matrix-vector products, studied in the previous section. Based on these results, we conclude that the matrix-free approach with the adopted here geometric multigrid preconditioner can deliver a very competitive solution strategy for engineering problems with compressible hyperelastic finite strain material models.

cores	N_{DoF}	N_{gref}	vmult [s]	CG [s / iteration]
20	4,442,880	3	$6.75 \cdot 10^{-2}$	0.53
160	35,071,488	4	$7.05 \cdot 10^{-2}$	0.56
1280	278,694,912	5	$8.3 \cdot 10^{-2}$	0.69

Table 5: Weak scaling of Algorithm 3 in 3D for quadratic polynomial basis.

Extension of this study from the node level to the cluster level is beyond the scope of this work. However, as a step in this direction we have performed a preliminary study of the weak scalability of the implementation. Table 5 demonstrates a good weak scaling up to 1280 processes[9] of the wall time per matrix-vector product and per iteration in the CG solver for the 3D problem with a tri-quadratic basis. This confirms that the matrix-free and multigrid frameworks implemented in the `deal.II` library can provide a competitive solution for cluster-sized problems as well, also compare [3, Figure 7] for scaling results for up to 147,456 CPU cores and 34.4 billion degrees of freedom for an incompressible flow problem.

8. Summary and Conclusions

In this contribution, we proposed and numerically investigated three different matrix-free implementations of tangent operators for finite-strain elasticity with heterogeneous materials. To the best of our knowledge, this is the first work that applies matrix-free sum factorization techniques to partial differential equations arising in this case. Among the three examined algorithms, the implementation that caches the material part of the fourth-order spatial tangent stiffness tensor together with the second-order Kirchhoff stress was shown to be faster than the matrix-based approach for polynomial degrees higher than one in 3D and polynomial degrees higher than two in 2D. Given that we did not utilize the specific form of the material part of the stiffness tensor in this case, we conclude that the matrix-free implementation of the tangent operator can be applied to any constitutive model which operates with the material part of the fourth-order spatial tangent stiffness tensor on the quadrature point level.

The roofline model indicates that the matrix-free finite-strain elasticity tangent operators have an order magnitude higher algorithmic intensity. We identified that

[9] This corresponds to 64 nodes, which is the maximum we could request on the RRZE Emmy HPC cluster.

one of the bottlenecks is related to the quadrature loop on each element where double contractions of fourth-order and second-order symmetric tensor as well as single contraction of two second-order tensors is performed.

We also propose a method by which to construct level tangent operators and employ them to define a geometric multigrid preconditioner with standard geometric transfer operations between each level. The GMG was applied to heterogeneous material assuming that the coarsest level can provide an adequate discretization of the heterogeneity. The numerical studies indicate that the proposed preconditioner leads to much fewer iterations of the iterative solver as compared to the algebraic multigrid preconditioner. The multigrid matrix-free preconditioner also leads to a solution approach that is faster than the matrix-based AMG for all polynomial degrees studied here. Most importantly, the wall time per degree of freedom for solving the problem is close to being constant. On the other hand, the matrix-based implementation with AMG becomes prohibitively expensive for higher-order bases. We conclude that the matrix-free implementations of tangent operators for finite-strain elasticity together with the geometric multigrid preconditioner is a very competitive solution strategy, as compared to more traditional matrix-based approach. Our future work will be focused on adopting the proposed matrix-free multigrid solution approach to FE^2 homogenization as well studying its behavior on the cluster level.

Acknowledgements

D. Davydov acknowledges the financial support of the German Research Foundation (DFG), grant DA 1664/2-1.

D. Davydov, D. Arndt and J-P. Pelteret are grateful to Martin Kronbichler (TU Munich), Jed Brown (CU Boulder) and Veselin Dobrev (Lawrence Livermore National Laboratory) for fruitful discussions on matrix-free operator evaluation approaches.

D. Arndt was supported by the German Research Foundation (DFG) under the project "High-order discontinuous Galerkin for the exa-scale" (ExaDG) within the priority program "Software for Exascale Computing" (SPPEXA).

P. Steinmann acknowledges the support of the Cluster of Excellence Engineering of Advanced Materials (EAM) which made this collaboration possible, as well as funding by the EPSRC Strategic Support Package "Engineering of Active Materials by Multiscale/Multiphysics Computational Mechanics".

References

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

[2] D. A. May, J. Brown, L. Le Pourhiet, A scalable, matrix-free multigrid preconditioner for finite element discretizations of heterogeneous Stokes flow, Computer methods in applied mechanics and engineering 290 (2015) 496–523.

[3] B. Krank, N. Fehn, W. A. Wall, M. Kronbichler, A high-order semi-explicit discontinuous Galerkin solver for 3D incompressible flow with application to DNS and LES of turbulent channel flow, Journal of Computational Physics 348 (2017) 634–659.

[4] J. Brown, Efficient nonlinear solvers for nodal high-order finite elements in 3D, Journal of Scientific Computing 45 (1-3) (2010) 48–63.

[5] B. Gmeiner, M. Huber, L. John, U. Rüde, B. Wohlmuth, A quantitative performance study for Stokes solvers at the extreme scale, Journal of Computational Science 17 (2016) 509–521.

[6] A. Abdelfattah, M. Baboulin, V. Dobrev, J. Dongarra, C. Earl, J. Falcou, A. Haidar, I. Karlin, T. Kolev, I. Masliah, et al., High-performance tensor contractions for GPUs, Procedia Computer Science 80 (2016) 108–118.

[7] K. Ljungkvist, M. Kronbichler, Multigrid for matrix-free finite element computations on graphics processors, Tech. rep., Technical Report 2017-006, Department of Information Technology, Uppsala University (2017).

[8] T. C. Clevenger, T. Heister, G. Kanschat, M. Kronbichler, A Flexible, Parallel, Adaptive Geometric Multigrid method for FEM, in preparation.

[9] M. Kronbichler, K. Kormann, Fast matrix-free evaluation of discontinuous galerkin finite element operators, arXiv preprint arXiv:1711.03590.

[10] S. Müthing, M. Piatkowski, P. Bastian, High-performance implementation of matrix-free high-order discontinuous galerkin methods, arXiv preprint arXiv:1711.10885.

[11] G. Alzetta, D. Arndt, W. Bangerth, V. Boddu, B. Brands, D. Davydov, R. Gassmoeller, T. Heister, L. Heltai, K. Kormann, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, D. Wells, The `deal.II` Library, Version 9.0, Journal of Numerical Mathematics.

[12] P. Wriggers, Nonlinear finite element methods, Springer Science & Business Media, 2008.

[13] L. R. G. Treloar, The Physics of Rubber Elasticity, Oxford University Press, USA, 1975.

[14] L. R. G. Treloar, H. G. Hopkins, R. S. Rivlin, J. M. Ball, The mechanics of rubber elasticity, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 351 (1666) (1976) 301–330. doi:10.1098/rspa.1976.0144.

[15] R. Bartlett, I. Demeshko, T. Gamblin, G. Hammond, M. A. Heroux, J. Johnson, A. Klinvex, X. Li, L. C. McInnes, J. D. Moulton, D. Osei-Kuffuor, J. Sarich, B. Smith, J. Willenbring, U. M. Yang, xSDK foundations: Toward an extreme-scale scientific software development kit, Supercomputing Frontiers and Innovations 4 (1). doi:10.14529/jsfi170104.

[16] C. Burstedde, L. C. Wilcox, O. Ghattas, `p4est`: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees, SIAM Journal on Scientific Computing 33 (3) (2011) 1103–1133. doi:10.1137/100791634.

[17] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, K. S. Stanley, An overview of the Trilinos project, ACM Trans. Math. Softw. 31 (3) (2005) 397–423. doi:http://doi.acm.org/10.1145/1089014.1089021.

[18] M. A. Heroux, Epetra performance optimization guide, Tech. Rep. SAND2005-1668, Sandia National Laboratories (2005).

[19] M. W. Gee, C. M. Siefert, J. J. Hu, R. S. Tuminaro, M. G. Sala, ML 5.0 smoothed aggregation user's guide, Tech. Rep. SAND2006-2649, Sandia National Laboratories (2006).

[20] P. E. Vos, S. J. Sherwin, R. M. Kirby, From h to p efficiently: Implementing finite and spectral/hp element methods to achieve optimal performance

for low-and high-order discretisations, Journal of Computational Physics 229 (13) (2010) 5161–5181.

[21] J. H. Bramble, J. E. Pasciak, J. Xu, Parallel multilevel preconditioners, Mathematics of Computation 55 (191) (1990) 1–22.

[22] W. L. Briggs, V. E. Henson, S. F. McCormick, A Multigrid Tutorial: Second Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[23] B. Janssen, G. Kanschat, Adaptive Multilevel Methods with Local Smoothing for H^1- and H^{curl}-Conforming High Order Finite Element Methods, SIAM Journal on Scientific Computing 33 (4) (2011) 2095–2114. doi:10.1137/090778523.

[24] W. Hackbusch, Multi-grid methods and applications, Springer Series in Computational Mathematics, Springer-Verlag Berlin Heidelberg, 1985. doi:10.1007/978-3-662-02427-0.

[25] P. Wesseling, An introduction to multigrid methods, Pure and applied mathematics, John Wiley & Sons, 1992.

[26] P. Suquet, Elements of homogenization for inelastic solid mechanics, in: E. Sanchez-Palencia, A. Zaoui (Eds.), Homogenization techniques for composite media, Vol. 272 of Lecture Notes in Physics, Springer-Verlag Berlin Heidelberg, 1987, pp. 193–278.

[27] R. Hill, On constitutive macro-variables for heterogeneous solids at finite strain, Proc. R. Soc. Lond. A 326 (1565) (1972) 131–147.

[28] Z. Hashin, Analysis of composite materials–a survey, Journal of Applied Mechanics 50 (3) (1983) 481–505.

[29] P. P. Castaneda, P. Suquet, Nonlinear composites, in: Advances in applied mechanics, Vol. 34, Elsevier, 1997, pp. 171–302.

[30] C. Miehe, C. Bayreuther, On multiscale FE analyses of heterogeneous structures: from homogenization to multigrid solvers, International Journal for Numerical Methods in Engineering 71 (10) (2007) 1135–1180.

[31] C. G. Bayreuther, C. Miehe, Coupling of homogenization techniques with multigrid solvers for unstructured meshes, in: Analysis and Simulation of Multifield Problems, Springer, 2003, pp. 67–72.

[32] J. Fish, V. Belsky, Multi-grid method for periodic heterogeneous media part 2: Multiscale modeling and quality control in multidimensional case, Computer Methods in Applied Mechanics and Engineering 126 (1) (1995) 17–38.

[33] L. Kaczmarczyk, C. Pearce, N. Bicanic, E. de Souza Neto, Numerical multiscale solution strategy for fracturing heterogeneous materials, Computer Methods in Applied Mechanics and Engineering 199 (17-20) (2010) 1100–1113. doi:10.1016/j.cma.2009.11.018.
URL `http://eprints.gla.ac.uk/38423/`

[34] R. S. Varga, Matrix iterative analysis, 2nd Edition, Springer, Berlin, 2009.

[35] D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, D. Wells, The `deal.II` library, version 8.5, Journal of Numerical Mathematics 25 (3). doi:10.1515/jnma-2017-0058.

[36] J. Treibig, G. Hager, G. Wellein, LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments, in: Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures, San Diego CA, 2010.

[37] M. Kronbichler, K. Kormann, I. Pasichnyk, M. Allalen, Fast matrix-free discontinuous galerkin kernels on modern computer architectures, in: International Supercomputing Conference, Springer, 2017, pp. 237–255.