

# CODE GENERATION AND PERFORMANCE ENGINEERING FOR MATRIX-FREE FINITE ELEMENT METHODS ON HYBRID TETRAHEDRAL GRIDS

FABIAN BÖHM<sup>†\*</sup>, DANIEL BAUER<sup>†\*</sup>, NILS KOHL<sup>‡</sup>, CHRISTIE ALAPPAT<sup>\*</sup>,  
DOMINIK THÖNNES<sup>†</sup>, MARCUS MOHR<sup>‡</sup>, HARALD KÖSTLER<sup>†\*</sup>, AND ULRICH RÜDE<sup>†§</sup>

**Abstract.** This paper introduces a code generator designed for node-level optimized, extreme-scalable, matrix-free finite element operators on hybrid tetrahedral grids. It optimizes the local evaluation of bilinear forms through various techniques including tabulation, relocation of loop invariants, and inter-element vectorization - implemented as transformations of an abstract syntax tree. A key contribution is the development, analysis, and generation of efficient loop patterns that leverage the local structure of the underlying tetrahedral grid. These significantly enhance cache locality and arithmetic intensity, mitigating bandwidth-pressure associated with compute-sparse, low-order operators. The paper demonstrates the generator’s capabilities through a comprehensive educational cycle of performance analysis, bottleneck identification, and emission of dedicated optimizations. For three differential operators  $(-\Delta, -\nabla \cdot (k(\mathbf{x}) \nabla), \alpha(\mathbf{x}) \mathbf{curl} \mathbf{curl} + \beta(\mathbf{x}))$ , we determine the set of most effective optimizations. Applied by the generator, they result in speed-ups of up to  $58\times$  compared to reference implementations. Detailed node-level performance analysis yields matrix-free operators with a throughput of 1.3 to 2.1 GDoF/s, achieving up to 62 % peak performance on a 36-core Intel Ice Lake socket. Finally, the solution of the curl-curl problem with more than a trillion ( $10^{12}$ ) degrees of freedom on 21 504 processes in less than 50 seconds demonstrates the generated operators’ performance and extreme-scalability as part of a full multigrid solver.

**Key words.** matrix-free finite elements, code generation, performance engineering

**MSC codes.** 65F50, 65N30, 65N55, 65Y20, 65F10

**1. Introduction.** Matrix-free finite element methods [10, 28, 30, 36, 40] address two main limitations faced by conventional approaches that follow the assemble-solve cycle. Matrix-vector operations using standard sparse storage formats are typically bandwidth-limited on state-of-the-art architectures [32]. The characteristic machine balance, i.e., the ratio of memory bandwidth (B/s) to performance (FLOP/s) of current hardware favors on-the-fly evaluation that reduces bandwidth-pressure at the cost of additional arithmetic operations [25, 31, 32, 35]. Secondly, the available memory typically limits the spatial resolution if the entire matrix has to be assembled and stored. An example from the geosciences illustrates the latter issue. The simulation of convection in the Earth’s mantle with a global resolution of 1 km requires about a trillion ( $10^{12}$ ) elements [6]. Under the simplifying assumption that the underlying discretization yields one degree of freedom per element and an operator with a 7-point stencil, we end up with a memory requirement of

$$(1.1) \quad \underbrace{7}_{\text{non-zeros per row}} \cdot \underbrace{10^{12}}_{\text{matrix rows}} \cdot \underbrace{8\text{B}}_{\text{double precision}} = 56 \text{ TB}$$

for the system matrix. This estimate is extremely optimistic. The number of degrees of freedom and the stencil size are typically much larger and the overhead required

<sup>\*</sup>Erlangen National High Performance Computing Center (NHR@FAU), Erlangen, Germany

<sup>†</sup>Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany  
({fabian.boehm, daniel.j.bauer, christie.alappat, dominik.thoennes, harald.koestler,  
ulrich.ruede}@fau.de).

<sup>‡</sup>Dept. of Earth and Environmental Sciences, Ludwig-Maximilians-Universität München (LMU), Munich, Germany ({nils.kohl, marcus.mohr}@lmu.de).

<sup>§</sup>Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique (CERFACS), Toulouse, France.

to store the indexing data structure of the sparse matrix format is omitted here. Expecting a memory requirement one order of magnitude higher than (1.1), storing the system matrix becomes infeasible, even on the majority of the largest available supercomputers. Since most iterative linear solvers only require the results of matrix-vector operations but no explicit access to the matrix entries, *matrix-free* methods enable the solution of linear systems with trillions ( $10^{12}$ ) of unknowns [16, 28] that could not be realized with standard sparse assembly.

However, the implementation of efficient matrix-free methods is challenging since the matrix-free execution requires that the discretization and the linear solvers are coupled more tightly. Performing a matrix-free matrix-vector multiplication requires information about the underlying mesh, finite element spaces and the differential operators. At the same time, having this knowledge enables further domain-specific optimizations that cannot be exploited using standard sparse linear algebra. The extensive range of combinations of differential operators, finite element spaces, application-dependent optimizations and target platforms renders the manual implementation and optimization of compute kernels a daunting task. Not only because of the amount of code that needs to be developed, maintained and tested, but also because expertise from both numerical mathematics and performance engineering is crucial to leverage the full potential of the underlying hardware.

**1.1. Contribution.** This paper presents the HyTEG Operator Generator<sup>1</sup> (HOG) – a unified pipeline that realizes the automated generation of matrix-free compute kernels from a symbolic description of a differential operator and respective finite element spaces. The compute kernels are tailored to block-structured, hybrid tetrahedral grids that enable direct addressing of unknowns via implicit, analytical index mappings for fast, contiguous and predictable memory access. Specifically, they are integrated into the HyTEG finite element framework<sup>2</sup> [26, 29].

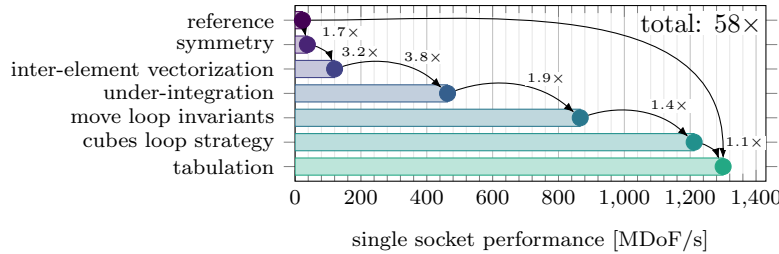
The main focus is put on the optimization of the matrix-free matrix-vector product  $y \leftarrow Ax$ , where  $A$  is the system matrix stemming from a finite element discretization of a partial differential equation (PDE). To demonstrate the flexibility of the code generator, we analyze bilinear forms that arise from the weak formulation of three different differential operators ( $-\Delta$ ,  $-\nabla \cdot (k(\mathbf{x}) \nabla)$ ,  $\alpha(\mathbf{x}) \mathbf{curl} \mathbf{curl} + \beta(\mathbf{x})$ ) for different types of finite element spaces (linear and quadratic Lagrange, first order Nédélec).

A central contribution of this paper is the in-depth, step-by-step performance analysis guiding the generation of optimized compute kernels through resource-based performance models [18]. Optimizations include identification of loop invariants in the abstract syntax tree (AST), inter-element vectorization, tabulation of factors of the weak form and several others. Tailored loop patterns that exploit the underlying structure of the grid to enhance cache locality are presented and analyzed. We evaluate the efficiency of the applied optimizations via the roofline performance model [46], layer condition analysis [20, 42], and analytical bounds for the memory traffic. For a range of weak forms, the set of most effective optimizations is identified from a larger pool of optimizations. Our analysis qualitatively and quantitatively demonstrates what limitations have to be overcome to achieve high node-level performance.

Concretely, the generated operators achieve a single-socket (36 cores) throughput of 1.3–2.1 GDoF/s on an Intel Xeon IceLake architecture. Thoroughly exploiting the machine, the operators reach up to 62 % of the machine’s peak performance and lie

<sup>1</sup><https://i10git.cs.fau.de/hyteg/hog>

<sup>2</sup><https://i10git.cs.fau.de/hyteg/hyteg>



**Figure 1** Performance of generated matrix-free matrix-vector multiplication kernels for a variable-coefficient diffusion operator  $-\nabla \cdot (k(\mathbf{x}) \nabla)$  discretized with quadratic conforming elements after application of individual performance optimizations. The final operator exhibits an accumulated speed-up of 58x. Detailed discussion in [Section 7](#).

close to the machine balance due to optimizations targeting the arithmetic intensity.

[Figure 1](#) summarizes the effect of the individual optimizations available in the code generator for a variable-coefficient diffusion operator discretized by quadratic, conforming finite elements. Compared to a reference implementation, we achieve an accumulated speed-up of 58x in this example. [Section 7](#) discusses more results and details of the analysis.

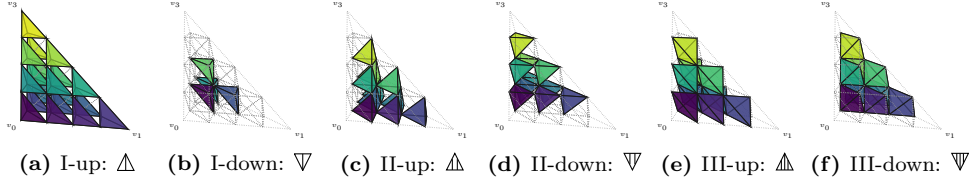
Finally, HYTEG’s full multigrid solver equipped with the generated matrix-free kernels demonstrates extreme scalability by solving a curl-curl problem with more than a trillion ( $> 10^{12}$ ) degrees of freedom (DoFs) on 21 504 processes in less than 50 seconds.

**1.2. Related Work.** Automated code generation has been successfully applied to accelerate the development of finite element (FE) based applications for several years and in many projects, most prominently via the FEniCS project [\[2\]](#). FEniCS generates code for fast global matrix assembly from a weak PDE definition in a dedicated, domain specific language (DSL) and using the FEniCS form compiler (FFC) [\[24\]](#). The resulting assembled linear system is solved with black-box solvers from the PETSc package [\[4\]](#). The present approach is similar but focuses on the generation of *matrix-free* FE methods, with a co-design of discretization, matrix-free operator and geometric multigrid solver. This is in contrast to the discretization-solver split implemented in FEniCS and offers the option to exploit the type of discretization in the solver. For this work, extreme scalability and high node-level performance are a first order design goal, which goes to the cost of having a narrower range of discretizations and PDEs than FEniCS.

A similar approach is realized through the Firedrake project [\[19\]](#). Like FEniCS, it uses a DSL to define the weak form and PETSc as a solver backend. Firedrake also supports matrix-free evaluation of matrix-vector products within Krylov solvers [\[25\]](#), with assembled matrices in the preconditioner.

The ExaStencils [\[34\]](#) framework offers code generation of whole programs. It implements a multi-layered language approach where each of the four layers provides a separate DSL that is tailored for domain experts from different communities. Problems can be generated from a textbook-like definition, where the discretization, solver and parallelization are generated by the framework.

**1.3. Structure.** The remaining article is structured as follows. The domain partitioning follows the concept of hybrid tetrahedral grids and is introduced in [Section 2](#). [Section 3](#) describes the architecture of the code generator. [Section 4](#) defines the finite element setting and the matrix-free application of an FE operator. [Section 5](#) presents loop strategies and discusses cache locality. [Section 6](#) summarizes the imple-



**Figure 2** Six types of micro-elements with different orientations in space, illustrated on refinement level  $\ell = 2$ . Each group is denoted by a unique symbol to streamline the presentation of algorithms. Combined, they constitute the complete macro tetrahedron. Micros of a certain orientation are, as visible, translation invariant. Depending on the orientation, the macro-tetrahedron fits a differing number of micro-elements, e.g., four  $\Delta$ -elements against only two  $\nabla$ -elements in the longest row. See [26] for details.

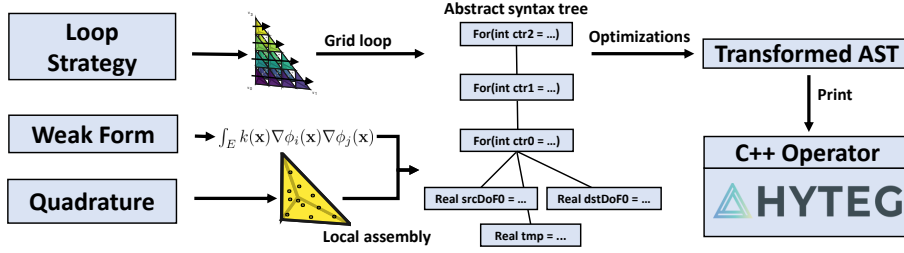
mented performance optimizations. Section 7 proceeds with the performance analysis and optimization of the generated kernels for different differential operators. Finally, Section 8 demonstrates the scalability of the generated operators in a matrix-free multigrid solver applied to the curl-curl problem.

**2. Hybrid Tetrahedral Grids.** The optimizations presented in this paper are tailored towards a block-structured domain partitioning. Specifically, we embed the generated compute kernels into the HYTEG framework for matrix-free, large-scale FE simulations [29]. HYTEG is based on the concept of hierarchical hybrid grids (HHG) [7]. The domain is approximated by an unstructured tetrahedral coarse grid that is uniformly refined according to [8, 26]. We refer to the elements of the unstructured coarse grid as macro-tetrahedra and to the tetrahedra that emerge from the refinement as micro-tetrahedra.

The resulting grid is fully structured within each macro-element, and each new tetrahedron is identical up to translation to one of six reference tetrahedra. See Figure 2 for an illustration of two times refined macro-tetrahedra and the categorization of the arising micro-tetrahedra. This local structure is heavily exploited by the compute kernels through implicit indexing of the FE data structures without indirections or additional bookkeeping of connectivities. Consecutive, direct memory access is crucial for the high performance compute kernels presented in this paper. The article [26] provides a detailed description of the refinement procedure and the resulting indexing schemes.

An extreme-scalable data structure is constructed through the distribution of the macro-primitives among parallel processes. The uniform refinement enables the construction of matrix-free geometric multigrid solvers by design, and thus the construction of asymptotically optimal matrix-free full multigrid solvers that are essential to solve PDEs at the extreme scale [28]. The performance and scalability of this approach was demonstrated in a series of articles, see also [6, 17, 26, 27, 44]. Specifically, the solution of saddle point systems with more than ten trillion ( $10^{13}$ ) unknowns on hundreds of thousands of processes could be realized due to the matrix-free implementation [16].

**3. Code Generation for FEM on Hybrid Tetrahedral Grids.** HOG implements a unified pipeline to generate efficient *matrix-free* FE compute kernels on block-structured tetrahedral grids. Similar to the FEniCS-approach, it automatically generates kernels from the symbolic description of a weak form and several other parameters, such as the quadrature rule, FE spaces and optimizations selected. Specifically, the HOG takes a weak form as input in the shape of a SYMPY [37] symbolic expression. The quadrature points and weights are supplied by the QUADPY library



**Figure 3** Code generation pipeline of the HOG. A scalable, matrix-free operator for hybrid tetrahedral grids is generated and optimized from simple input parameters. The resulting kernel is embedded into the HyTEG FE framework [29, 26].

[41]. An abstract syntax tree (AST) is constructed from the inputs using AST node classes from PYSTENCILS [5], a library for the generation of stencil codes. Optimizations are applied to the AST, before C++ code is printed that will be called by the HyTEG backend. Figure 3 visualizes the code generation pipeline.

The HOG generates kernels that execute an operation on a single refined macro-primitive. Different operations are supported, including matrix-free matrix-vector multiplication and assembly of matrix diagonals, and new operations can easily be added. The generated code includes not only assembly of the element matrices on a single element, but also encompasses the loop over the local subdomain. The regular structure within a macro tetrahedron enables generation of specially adapted loop strategies, which significantly improve memory access patterns and, thus, the operator’s performance. Inter-element vectorization [43], i.e., computing multiple elements in parallel using vectorized instructions, is a second central optimization.

The HOG applies a wide range of efficient, state-of-the-art quadrature rules for simplices, like the Xiao-Gimbutas rules [47] and can be configured to under-integrate, when applicable. If reasonable, quadrature-free kernels can also be generated. Additionally, the HOG applies established optimization techniques such as loop fusion, tree- or polynomial-based common subexpression elimination (CSE) [22], exploitation of symmetry and tabulation [24], all adapted to hybrid tetrahedral grids.

The generated operator uses HyTEG’s MPI communication routines, providing the desired scalability. HyTEG’s geometric multigrid and Krylov solvers then employ the generated operators in smoothers, residual computations and coarse grid solves.

**4. Matrix-Free Finite Elements.** We consider the solution of linear systems of the form  $A\mathbf{v} = \mathbf{f}$  arising from the discretization of linear elliptic PDEs with the finite element method [9, 14] subject to a triangulation  $\mathcal{T}(\Omega)$  of the domain  $\Omega$  and finite dimensional trial and test spaces  $\mathcal{V} = \langle \phi^i \rangle_{i \in \mathcal{I}^V}$  and  $\mathcal{W} = \langle \psi^j \rangle_{j \in \mathcal{I}^W}$  with associated basis functions  $\phi^i$  and  $\psi^j$ .

Most iterative linear solvers such as multigrid and Krylov methods do not require access to the entries of  $A$ . It is sufficient to provide a method to compute the *result* of the application of  $A$  to a vector, which can be implemented via *matrix-free* kernels, enabling a solution of the linear system without the need to explicitly form  $A$ .

With an integrand  $G$  that originates from the weak formulation of the PDE, we

---

**Algorithm 1** Apply the local operator  $A_T$  on element  $T$ .

---

```

1: function LOCALAPPLY( $T, \mathbf{v}, \mathbf{w}$ )
2:    $A_T \leftarrow \left[ \sum_q |\det J_F| w_q \hat{G}(\hat{\mathbf{x}}_q, \hat{\phi}^i, \hat{\psi}^j) \right]_{j \in \mathcal{I}_T^\mathcal{W}, i \in \mathcal{I}_T^\mathcal{V}}$  ▷ assemble local operator
3:    $\mathbf{w} \leftarrow \mathbf{w} + P_T^\mathcal{W} A_T R_T^\mathcal{V} \mathbf{v}$  ▷ apply

```

---

have that

$$(4.1) \quad A = \left[ \int_{\mathcal{T}(\Omega)} G(\mathbf{x}, \phi^i, \psi^j) \right]_{j \in \mathcal{I}^\mathcal{W}, i \in \mathcal{I}^\mathcal{V}} = \sum_{T \in \mathcal{T}(\Omega)} P_T^\mathcal{W} \underbrace{\left[ \int_T G(\mathbf{x}, \phi^i, \psi^j) \right]_{j \in \mathcal{I}_T^\mathcal{W}, i \in \mathcal{I}_T^\mathcal{V}}}_{=: A_T} R_T^\mathcal{V}.$$

It is sufficient to compute the integral over  $T$  for pairings  $i, j$  with overlapping support on  $T$ .  $R_T^\mathcal{V} : \mathbb{R}^{|\mathcal{I}^\mathcal{V}|} \rightarrow \mathbb{R}^{|\mathcal{I}_T^\mathcal{V}|}$  and  $P_T^\mathcal{W} : \mathbb{R}^{|\mathcal{I}_T^\mathcal{W}|} \rightarrow \mathbb{R}^{|\mathcal{I}^\mathcal{W}|}$  denote the selection of DoFs with support on  $T$  and the corresponding mapping back to the set of global DoFs, respectively.

**Algorithm 1** implements the local matrix-vector multiplication on an arbitrary element  $T$ . In practice, the integral over  $T$  is transformed to a reference element  $\hat{T}$  and evaluated by a quadrature rule with points  $\hat{\mathbf{x}}_q$  on  $\hat{T}$  and weights  $w_q$ . The map  $F : \hat{T} \rightarrow T$  facilitates the transformation of  $G, \phi$  and  $\psi$  to their reference versions  $\hat{G}, \hat{\phi}$  and  $\hat{\psi}$ . In this paper we only consider the case that the map is affine, in which case its Jacobian  $J_F$  is constant on each structured subdomain, i.e.,

macro-tetrahedron. **Algorithm 1** is applied inside a loop over all grid elements which is subject of [Section 5](#).

**REMARK 1.** *The local matrix-vector multiplication is not necessarily split into the assembly of the local matrix  $A_T$  and the subsequent matrix-vector multiplication. The local quadrature approach described in [30] fuses the two operations, such that each entry of the local result vector  $A_T R_T^\mathcal{V} \mathbf{v}$  is evaluated using only a single integral. The HOG does this implicitly by unrolling quadrature loops and fusing the assembly and local matrix-vector multiplication during a CSE.*

**5. Optimizing Cache-Locality: Loop Strategies.** On hybrid tetrahedral grids, the sum over all elements  $T$  of the grid  $\mathcal{T}(\Omega)$  in (4.1) results in an outer loop over the (unstructured) macro-elements  $T_M$  of the macro (coarsest) grid  $\mathcal{T}_M(\Omega)$  and an inner loop over the (structured) micro-elements  $T_m$  which arise from each macro due to uniform refinement. We refer to the specific order in which the micro-elements of a single macro-element are traversed as *loop strategy*.

**5.1. Sawtooth Loop Strategy.** Using the *sawtooth* loop strategy, the micro-elements  $T_m$  of the current macro-element  $T_M$  are iterated over in six separate loops, one for each orientation of micro elements. Each of these *element-loops* consists of a triple-nested spatial loop. **Algorithm 2** implements the grid loop with the sawtooth loop strategy. Selecting a micro element of a specific orientation  $\blacktriangle$  and position  $x, y, z$  in space within a macro element  $T_M$  is denoted by  $T_M(\blacktriangle, x, y, z)$ .  $B(\blacktriangle, \cdot)$  determines the loop bound, which depends on the spatial orientation of the micro-element. Specifically, we have  $B(\blacktriangle, n) = n$ ,  $B(\blacktriangledown, n) = n - 1$  for  $\blacktriangle \in \{\blacktriangle, \blacktriangledown, \blacktriangleleft, \blacktriangleright\}$ , and  $B(\blacktriangledown, n) = n - 2$  [26].

**Algorithm 2** is straightforward to implement but has suboptimal memory properties for conforming discretizations. DoFs are shared between micro-elements, there-

**Algorithm 2** Sawtooth loop strategy.

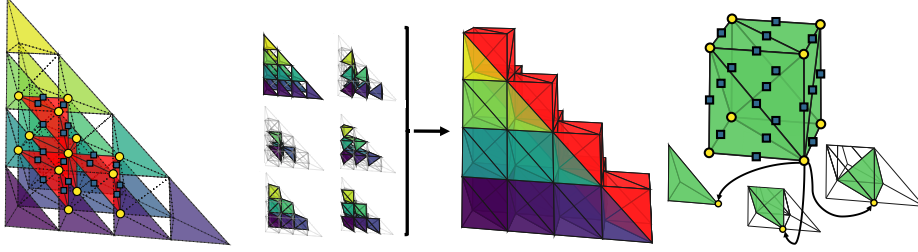
---

```

1: function ELEMENTWISEAPPLYSAWTOOTH( $\mathbf{v}$ ,  $\mathbf{w}$ ,  $\ell$ )
2:   for each  $T_M \in \mathcal{T}_M(\Omega)$  do ▷ loop macro elements
3:     for each  $\Delta \in \{\Delta, \nabla, \Delta, \nabla, \Delta, \nabla\}$  do ▷ loop orientations
4:       for  $z = 0 \dots B(\Delta, 2^\ell)$  do ▷ element loop
5:         for  $y = 0 \dots B(\Delta, 2^\ell - z)$  do
6:           for  $x = 0 \dots B(\Delta, 2^\ell - z - y)$  do
7:             LOCALAPPLY( $T_M(\Delta, x, y, z)$ ,  $\mathbf{v}$ ,  $\mathbf{w}$ )

```

---



**Figure 4** *Left:* Micro-elements of the  $\Delta$ - (purple to yellow gradient) and  $\nabla$ -type (in red) share vertex and edge DoFs. The DoFs are loaded from main memory during the  $\Delta$ -element loop, which then iterates the remaining macro-tetrahedron. The overlapping DoFs are evicted from cache in the process for practically relevant refinement levels. The following  $\nabla$ -element-loop iterates over the red micro-elements and has to reload the overlapping DoFs from main memory. *Middle:* The 6 element loops of the sawtooth loop strategy are fused together. Red micro elements lie outside the macro-tetrahedron and must be omitted through conditionals and by splitting the iteration space into complete and incomplete iterations. *Right:* The vertex-DoF in the lower front-right corner is part of multiple micro tetrahedra and will be loaded only once from main memory with the cubes loop strategy.

fore, DoFs are accessed repeatedly in multiple successive element-loops of [Algorithm 2](#). After a certain element-loop loaded a specific DoF from main memory, it will iterate the whole remaining macro-tetrahedron, such that the DoF is likely evicted from cache. When the following element-loop accesses the same DoF, it has to load it from main memory again. This leads to more main memory traffic than necessary. [Figure 4](#) (left) depicts a concrete example.

**5.2. Cubes Loop Strategy.** The disadvantageous memory properties of the sawtooth loop strategy can be alleviated by fusing the 6 element-loops. The single, fused loop computes the operator application on 6 micro-elements per iteration, which compose a cube. If a DoF is part of multiple micros within the same cube, as exemplified in [Figure 4](#) (right), all accesses to this DoF from these micros are within a single iteration of the cubes loop. Due to this temporal locality, the DoF is kept in cache and must not be reloaded from main memory. Thus, the cubes loop strategy can be considered a form of spatial blocking on tetrahedral grids. The cubes loop strategy is depicted in [Figure 4](#) (middle) and implemented in [Algorithm 3](#).

Naively, cubes-iterations at the diagonally oriented plane of the macro tetrahedron include micro-elements that are not part of the macro tetrahedron (marked red in [Figure 4](#) (middle)). These micros could be excluded from the iteration using conditionals. However, to avoid conditionals in the innermost loop, the  $x$ -loop is cut into three parts instead. In all but the last two iterations, all elements of a cube lie within the macro-element. Thus, we call these iterations *complete*. On the other hand, the remaining two iterations are *incomplete*. Neither includes the  $\nabla$ -tetrahedron, and in the last cube, only the  $\Delta$ -tetrahedron is traversed.

**Algorithm 3** Cubes loop strategy.

---

```

function ELEMENTWISEAPPLYCUBES(v, w,  $\ell$ )
  for each  $T_M \in \mathcal{T}_M(\Omega)$  do                                 $\triangleright$  loop macro elements
    for  $z = 0 \dots 2^\ell$  do                                     $\triangleright$  triple nested spatial loop
      for  $y = 0 \dots 2^\ell - z$  do
        for  $x = 0 \dots 2^\ell - z - y - 2$  do                     $\triangleright$  complete iterations
          for each  $\Delta \in \{\Delta, \nabla, \triangle, \nabla, \triangle, \nabla\}$  do
            LOCALAPPLY( $T_M(\Delta, x, y, z)$ , v, w)
          for each  $\Delta \in \{\Delta, \triangle, \nabla, \triangle, \nabla\}$  do     $\triangleright$  incomplete iterations
            LOCALAPPLY( $T_M(\Delta, 2^\ell - z - y - 1, y, z)$ , v, w)
          LOCALAPPLY( $T_M(\Delta, 2^\ell - z - y, y, z)$ , v, w)

```

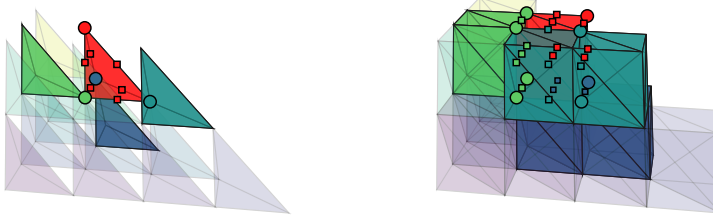
---

**5.3. Memory Volume and Layer Conditions.** In the following, we evaluate cache locality of both loop strategies and validate the assumptions made in the previous sections. To that end, we bound the main memory volume  $M$ , that is how much data the considered operators read from and write to main memory, by theoretical lower and upper bounds  $M_{\text{lower}} \leq M \leq M_{\text{upper}}$ . If  $M$  is close to  $M_{\text{lower}}$ , the loop strategy is efficient, but for a naive strategy  $M$  might be close to  $M_{\text{upper}}$ . Determining  $M_{\text{lower}}$  is straightforward: the operator has to read and write each DoF once from main memory in order to update its content by a matrix-vector multiplication.  $M_{\text{upper}}$  represents the worst case: the operator reloads all associated DoFs on each micro-element from main memory, even if they have already been accessed on previous elements.

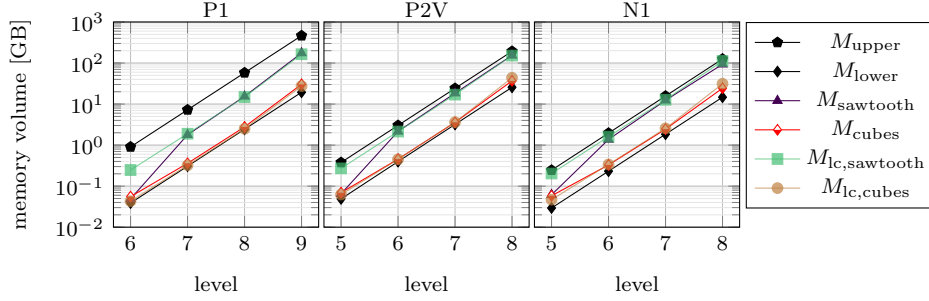
The memory volume can be estimated more precisely through layer conditions [42, 20]. We translate the concept of layer conditions, which was originally developed for stencil codes, to conforming, elementwise finite element operators. To that end, we assume that a DoF can be read from cache if and only if *all* DoFs of at least one neighboring micro-element still reside in cache.

Figure 5 (left) illustrates this for the  $\Delta$ -part of the sawtooth loop-strategy for vertex and edge DoFs of a  $\mathcal{P}_2$  discretization. The DoFs of the current micro-tetrahedron (red) are subdivided into different groups: The light green DoFs will always be in cache because they have been accessed in the previous iteration. The red DoFs will never be in cache because they lie ahead in the iteration order (assuming a single operator application). For teal and blue DoFs it depends on layer conditions, i.e., whether the neighbor element with the corresponding color has been evicted. Put differently, whether the *tail* of the iteration containing those elements is still in cache. In the cubes loop Figure 5 (right), a whole front of DoFs can be read from cache from the last cube iteration (light green). They make up  $1/3$  of all DoF accesses. With the sawtooth loop strategy, this fraction is just  $1/10$ .

In practice, we compute an estimate for the main memory volume based on layer conditions  $M_{\text{lc},s}$ ,  $s \in \{\text{sawtooth}, \text{cubes}\}$  as follows. We know the memory volumes  $M_s^{\text{red}}$ ,  $M_s^{\text{teal}}$  and  $M_s^{\text{blue}}$  associated to the respective DoF accesses per iteration of a loop strategy  $s$ . Furthermore, we can compute the size of the tails  $M_{i,s}^{\text{tail, teal}}$  and  $M_{i,s}^{\text{tail, blue}}$  between the current element in iteration  $i$  and the elements that have to reside in cache to provide an overlapping DoF (teal or blue colored). Then, we compare the computed tail size with the actual size of the cache on the used machine, e.g. the L3 cache. Doing this for each micro-element on each macro-element, we obtain an estimate for the memory volume an operator requires during its matrix-free application:



**Figure 5** Categorization of vertex and edge DoFs on the current iteration (red) of a sawtooth (left) and cubes loop (right): light green DoFs are always cached from the previous iteration (light green elements), red DoFs are never cached because they lie ahead of the iteration order, teal and blue DoFs are cached depending on the presence of data from the corresponding neighbor elements in cache.



**Figure 6** Corridor plot of main memory volume against refinement levels, comparing the measurements with the analytical bounds and layer condition estimates.

$$(5.1) \quad M_{lc,s} = n_{\text{macros}} \cdot \sum_i M_s^{\text{red}} + \begin{cases} M_s^{\text{teal}} & \text{if } M_{i,s}^{\text{tail, teal}} \geq M_{L3} \\ M_s^{\text{blue}} + M_s^{\text{teal}} & \text{if } M_{i,s}^{\text{tail, blue}} \geq M_{L3} \end{cases}$$

with  $n_{\text{macros}}$  being the number of macro tetrahedra on the MPI process. If the tail for a certain neighboring element, e.g. the teal colored ones in Figure 5, is larger than the cache size, it must have been evicted from cache and reloaded from main memory on access. This is a simplified model that neglects certain aspects of the caching behavior, but yields good results in practice.

**5.4. Memory Study.** Figure 6 compares the measured main memory volume for the two loop strategies  $M_{\text{sawtooth}}$ ,  $M_{\text{cubes}}$ , obtained by LIKWID's [45] hardware performance counters, with the bounds  $M_{\text{upper}}$ ,  $M_{\text{lower}}$  and layer condition estimates  $M_{lc, \text{sawtooth}}$ ,  $M_{lc, \text{cubes}}$  based on (5.1). We measure (P1), a constant diffusion operator  $(-\Delta)$  discretized by linear Lagrangian elements, (P2V), a variable-coefficient diffusion operator  $(-\nabla \cdot (k(\mathbf{x}) \nabla))$  discretized by quadratic Lagrangian elements and (N1), a curl-curl operator  $(\alpha(\mathbf{x}) \mathbf{curl curl} + \beta(\mathbf{x}))$  discretized by Nédélec elements (more details on the operators in Section 7).

The upper and lower bounds define a corridor in which all measurements lie. The cubes loop strategy consistently comes close to the lower bound, showing high cache locality and only few, necessary accesses to main memory.

In terms of cache locality, it clearly outperforms the sawtooth loop strategy which consistently places close to the upper bound. On low refinement levels both strategies achieve very low memory volumes close to the lower bound. For these cases the data of a complete macro-tetrahedron fits into the L3-cache such that no special strategy is required to improve caching.

Furthermore, the estimates incorporating layer conditions closely match the measurements for both loop strategies. On level 9 for (P1) and 8 for (P2V) and (N1), the stronger (blue) layer condition breaks, and we see an upward kink in the measured memory volume for both loop strategies (less visible for sawtooth due the breaking of layer conditions only contributing a single DoF). Apparently, our estimates precisely capture the breaking of layer conditions.

Complementary to these pure memory focused observations, Section 7 analyses the impact of the cubes loop strategy on kernel performance.

**6. Optimizing Computations.** In the following, we present the optimizations the code generator applies to speed-up computation and reduce redundant calculations, and how they are realized as AST transformations during generation time. In Section 7 we will apply them to three different bilinear forms.

**6.1. Automatic Identification of Loop Invariants.** The evaluation of a bilinear form over elements of a structured grid may contain large fractions of redundant computation. For example, due to the translation invariance of micro-elements of the same orientation [26], their Jacobians are identical and therefore loop-invariant. It is common that in this and similar cases, the computation is moved outside of the loop, see [49] for more examples.

Such invariants can be identified automatically by the HOG via traversing the AST and checking statements in loop bodies for a dependency on loop counters. If there is no dependency, the statement is moved in front of the loop, eliminating the redundancy and reducing the number of floating point operations (FLOPs). The HOG thereby not only targets the local operator application by drawing invariants out of the loop over quadrature points, but also recognizes computation that is spatially-invariant.

**6.2. Inter-element Vectorization.** Automatic vectorization of the grid loop of the matrix-free operator application is a significant challenge for the backend C++ compiler. This is due to the irregular iteration space on tetrahedra and a large loop body possibly containing another loop over quadrature points. On the other hand, handwriting vectorized versions of the assembly for different vector widths, instruction sets, architectures, weak forms and discrete function spaces is extremely tedious.

The HOG automatically vectorizes the matrix-free operator application during generation time. It cuts the  $x$ -direction loop into a vectorized and a remainder loop. All AST-nodes in the body of the former are replaced by vector instructions and the loop counter is modified to run over patches of multiple elements. The result is an operator that computes the local assembly on multiple micro-elements simultaneously, an inter-element vectorization similar to [43].

**6.3. Integration.** The type of quadrature rule used to compute the integrals in the local assembly significantly affects the number of FLOPs an operator executes. A quadrature rule that achieves exact integration with the fewest points is an obvious choice. However, according to [14], for second-order elliptic variational problems with polynomial degree  $q$  in the FE spaces, a quadrature rule exact up to polynomials of degree  $2q - 2$  is sufficient to achieve the expected convergence rate. This is called *under-integration* [24]. Alternatively, the HOG can integrate the local matrix analytically, providing the option of a quadrature-free kernel.

**6.4. Common Subexpression Elimination.** Two types of CSE are available in the HOG: a traditional, tree-based CSE and a polynomial CSE following [22]

which may be more effective for expressions that resemble polynomials. Examining their effectivity on matrix-free FE operators is out of scope of this paper.

**6.5. Quadrature Loops.** The generator can either unroll the loops over quadrature points in [Algorithm 1](#) or generate them in loop form. In the latter case, the quadrature loops for each entry of the local matrix are fused into a single loop to improve the CSE applied to the loop body. Unrolling the quadrature loops offers a wider space of expressions the CSE can eliminate on, which leads to more effective elimination and fewer FLOPs. However, it can bloat the number of statements in the kernel, leading to L2 cache problems. We investigate this in [Section 7.4.1](#).

**6.6. Tabulation of Factors of the Weak Form.** Another common technique orthogonal to precomputing whole local element matrices is to *tabulate*, that is to precompute factors of the weak form, e.g., the shape functions and their gradients, store them in table-like structures and access them in the kernel. This is e.g., done by the BASIX package within the FEniCS project [\[2\]](#).

Consider the entries of the local element matrix that arise from the discretization of the operator  $-\nabla \cdot (k(\mathbf{x}) \nabla)$ :

$$(6.1) \quad A_{T_m} = \left[ \sum_q k(\hat{\mathbf{x}}_q) \underbrace{w_q |\det J_F| (J_F^{-T} \hat{\nabla} \phi^i(\hat{\mathbf{x}}_q) \cdot J_F^{-T} \hat{\nabla} \phi^j(\hat{\mathbf{x}}_q))}_{\star} \right]_{j \in \mathcal{I}_{T_m}^{P_2}, i \in \mathcal{I}_{T_m}^{P_2}}.$$

The gradients in [\(6.1\)](#) are computed solely at quadrature points on the reference element  $\hat{T}$  and independent of the micro element  $T_m$ . They can be tabulated for each shape function  $\phi^i$  and quadrature point  $\hat{\mathbf{x}}_q$ . The table has  $10 \cdot n_q$  vector-valued entries for 10 shape functions in  $\mathcal{I}_{T_m}^{P_2}$  and  $n_q$  quadrature points. It is accessed in the loop over micro elements and the values used to compute the local assembly.

On hybrid tetrahedral grids, the important property that micro elements of a certain orientation  $\blacktriangle$  are translation-invariant [\[26\]](#) leads to the Jacobian of the affine mapping  $J_F$  being identical for all micro elements of that orientation. *Therefore, on hybrid tetrahedral grids, not only the gradient in (6.1) can be tabulated, but the complete factor  $\star$ .* Then, we tabulate for each pairing of shape function  $(\phi^i, \phi^j)$ , quadrature point and orientation of micro-elements, yielding a table with  $6 \cdot 100 \cdot n_q$  scalar entries, when neglecting symmetry.

**REMARK 2.** *In case of an additional curvilinear transformation occurring on certain meshes, another, spatially dependent Jacobian enters the weak form and we have to tabulate multiple factors.*

**6.7. Symmetry.** A symmetric variational form implies a symmetric local matrix. For such forms only half of the off-diagonal entries have to be computed, an optimization that reduces the FLOPs required for local assembly.

In moderately sized ASTs, the CSE is able to detect such symmetries itself without any additional effort. However, for large ASTs with many nodes, e.g., stemming from high degree quadrature rules or complicated integrands, the CSE sometimes fails to detect symmetric parts of the local matrix. Fortunately, the optimization is straightforward to implemented as an AST transformation: the entries of the local matrix are available as symbolic expressions and the symmetric entries can be replaced with accesses to their counterparts.

**6.8. Precomputation of Local Element Matrices.** A common technique to speed up the matrix-vector multiplication of an FE operator is to compute the

local element matrix for each element and store it a priori [12, 13]. Each time the operator is applied, the stored local matrix of the current element in the iteration is loaded from memory and applied to the local DoFs. This is especially advantageous for complicated variational forms and high-order quadrature rules, because all the implied computation can be shifted to a setup phase. However, the major drawback of the approach is the massively increased demand in main memory volume that makes it only feasible for relatively low refinement levels. *Furthermore, an operator using precomputation essentially only runs as fast as the local matrices can be loaded from memory and is thereby deeply memory-bound.* We include this here only for reference, as this approach is not matrix-free.

**7. Performance Analysis.** We evaluate the optimizations from Sections 5 and 6 implemented in the HOG by generating a range of operators for three different bilinear forms and different finite element spaces. Particularly, we generate the matrix-free matrix-vector multiplication optimized for maximum throughput measured in updated DoF/s on a single node of the architecture presented in Section 7.1.

We will make use of the roofline performance model [46] which serves as a simple, yet effective tool to understand the measured performance, observe the impact of optimizations, identify bottlenecks and guide the optimization process. The peak performance, maximum memory bandwidth and other roofline-constants are measured using LIKWID [45]. The general approach is to explore the search space that is given by all combinations of optimizations and determine the fastest operator for each weak form, for which we lay out the optimization path taken to obtain it.

We emphasize three optimizations here, because the HOG applies them to all weak forms in this article and they can be considered as a standard approach to optimize matrix-free, elementwise FE operators (on hybrid tetrahedral grids):

1. Speed-up arithmetic by vectorizing across elements and executing arithmetic in parallel on multiple vector lanes. This shows as a performance boost and straight upward shift in the roofline model plot.
2. Eliminate redundant computation by identifying loop invariants and moving them ahead of the loop. This reduces the arithmetic intensity and shifts operators in direction of the memory-bound region of the roofline.
3. Alleviate the memory-boundedness induced during the previous step with the cubes loop strategy. Improving the cache-locality increases the arithmetic intensity and manifests as a shift in the direction of the compute-bound region of the roofline. A performance boost is caused by the reduction of the main memory volume in memory-bound operators.

Each partially optimized variant of an operator is labeled by a prefix for each of the test cases ((P1), (P2V), or (N1)) that are described at the start of the following sections. They are separated by an '\_' from a list of letters encoding the applied optimizations, which are given in Table 1. For instance, P1\_SV represents matrix-free operator for the diffusion form with optimizations symmetry (S) and vectorization (V).

**7.1. Test Case and Machine.** A single matrix-free application in double precision serves as the main test application. We do not consider the communication prior to and after the operator application, as this is out-of-scope for this work but will be considered in future publications.

The experiments are conducted on refinement level 7. Generally, we want to use as few macro-elements as possible while still accurately capturing the domain. This way, we can use many refinements, yielding largely ordered structure and potential for high

| optimization                | short |
|-----------------------------|-------|
| symmetry                    | S     |
| inter-element vectorization | V     |
| loop invariants             | I     |
| cubes loop strategy         | C     |
| under-integration           | U     |
| fused quadrature loops      | fQ    |
| tabulation                  | T     |
| precomputation              | P     |

**Table 1** Range of optimizations from [Section 6](#) and their abbreviations.

|                   |         |              |
|-------------------|---------|--------------|
| cores             | 36      | (per socket) |
| L1data cache size | 48 KB   | (per core)   |
| L1inst cache size | 32 KB   | (per core)   |
| L2 cache size     | 1.25 MB | (per core)   |
| L3 cache size     | 54 MB   | (shared)     |
| clock speed       | 2.6 GHz | (fixed)      |

**Table 2** Relevant technical details for the Intel(R) Xeon(R) Platinum 8360Y CPUs of type IceLake SP.

performance. Our previous applications indicate that the refinement level typically lies in the range from 5 to 8. In order to distribute at least one macro-primitive to each process, the coarse grid should also have at least as many macro-primitives as there are processes. For this paper, we choose a simple cuboid geometry made up of 36 macro-tetrahedra for the coarsest grid  $\mathcal{T}_M(\Omega)$ .

The macro-primitives are distributed to the 36 cores of a single socket of the Fritz supercomputer at the Erlangen National High Performance Computing Center NHR@FAU [39]. [Table 2](#) summarizes relevant information regarding the cache hierarchy, clock speed and number of cores of the CPU type built into the Fritz supercomputer. Measurements of roofline constants and the test application are conducted with likwid-bench and likwid-mpirun from the LIKWID performance-measurement tools [45]. The clock frequency is fixed to 2.6 GHz by the slurm environment on Fritz and processes are pinned to processors by LIKWID pinning masks. The applications are compiled with Intel ICX 2021.4 and the `-march=native -O3` flags.

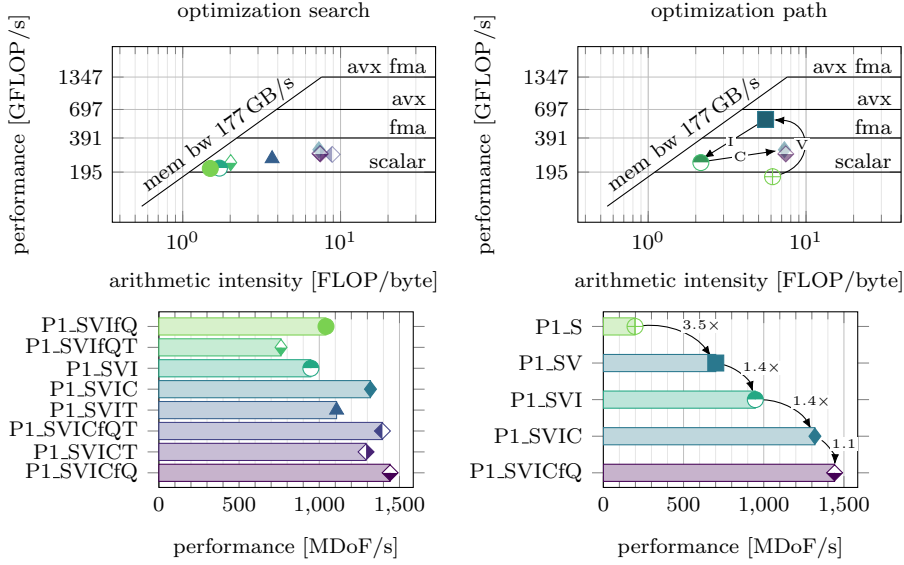
The reference implementations of the elementwise operators, on which the optimizations build, are generated by the HOG without optimizations and do not yet take the regular grid structure into account. Furthermore, apart from running SYMPY's CSE during generation of the local assembly, they fully rely on the backend C++ compiler for optimizations like vectorization.

**7.2. Constant Diffusion.** The simplest test case benchmarks a constant diffusion operator  $-\Delta$  discretized by linear continuous Lagrangian elements, i.e.,  $\mathcal{V} = \mathcal{W} = \mathcal{P}_1(\mathcal{T}_m(\Omega))$ . Despite its simplicity, its performance characteristics are representative for compute-sparse, low-order operators, such as weak divergence, gradient, or stabilization terms for the mixed  $\mathcal{P}_1 - \mathcal{P}_1$  approximation of the Stokes equation used in [23]. The local operator is given by:

$$(P1) \quad A_{T_m} = \left[ \int_{T_m} \nabla \phi^i \cdot \nabla \psi^j \right]_{j \in \mathcal{I}_{T_m}^{\mathcal{P}_1}, i \in \mathcal{I}_{T_m}^{\mathcal{P}_1}}$$

with  $\mathcal{I}_{T_m}^{\mathcal{P}_1}$  the index set of linear basis functions with support on micro element  $T_m$ . The linear spaces yield  $1.3 \times 10^8$  DoFs on the test cube at level 7.

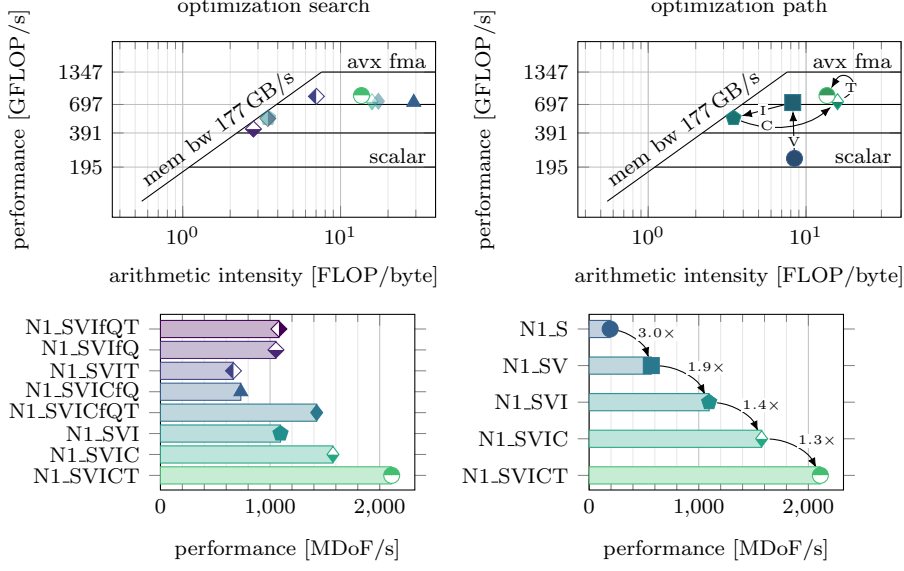
The operators using the most effective sets of optimizations with respect to MDoF/s are plotted on the left column of [Figure 7](#). Remarkably, the four fastest operators all utilize the cubes loop strategy (C). Given the inherent compute-sparsity associated with low polynomial degree, the enhancement of cache-locality becomes imperative to overcome the constraints imposed by memory bandwidth limitations. These operators achieve a throughput of up to 1.5 GDoF/s but reach only 23 % of the AVX FMA peak performance.



**Figure 7** *Left column:* Roofline (top) and MDoF/s (bottom) for the set of fastest operators of the form (P1). *Right column:* Roofline (top) and MDoF/s (bottom) with speed-ups for the optimization path to P1.SVICfQ  $\blacklozenge$ . The accumulated speed-up is 7.3 $\times$ .

The optimization path to obtain P1.SVICfQ  $\blacklozenge$ , the fastest operator on the current machine is presented on the right column of Figure 7. We start with P1.S  $\oplus$ , an operator restricted by the scalar instruction roof. Employing our standard optimization of inter-element AVX vectorization (P1.SV  $\blacksquare$ ), the code is no more limited to the hardware roof prescribed by scalar instructions, and the performance shoots vertically up, reaching close to the AVX roofs. To obtain P1.SVI  $\bullet$ , the HOG identifies the whole integrand, that is the constant gradients of test and trial basis and the Jacobian of the pull-back, as loop invariant and moves it ahead of the loop. This drastically decreases the overall computations while keeping the memory traffic almost constant. The result is a corresponding decrease in the arithmetic intensity, rendering P1.SVI  $\bullet$  memory-bound. The performance drops in terms of FLOP/s from close to 50 % to 23 % peak performance. However, the performance in DoF/s improves (and consequently runtime decreases) by a factor of 1.4 $\times$  due to a reduction in the overall FLOPs. Finally, we enhance the cache-locality of the memory-bound P1.SVI  $\bullet$  by the cubes loop strategy, recovering its arithmetic intensity and placing it close to the machine balance. The final operator P1.SVICfQ  $\blacklozenge$  achieves an accumulated speed-up of  $\sim 7.3\times$  compared to the scalar version.

**REMARK 3.** *Contrary to the upcoming, more compute-intensive (N1) and (P2V) operators, (P1) operators are still memory-bound or at the machine-balance even after applying the cubes loop strategy. Relocating loop invariants is exceptionally effective here, removing almost all computations from the loop and essentially reducing the operator to an elementwise copy-benchmark on tetrahedral grids. Thereby, the operators are prone to bottlenecks in the data path. Although the optimization leads to speed-up, it causes lower FLOP/s performance. This exemplifies the crucial difference between GDoF/s and FLOP/s: the latter is not always meaningful to measure performance improvements in presence of an optimization that changes the amount of FLOPs.*



**Figure 8** *Left column:* Roofline (top) and MDoF/s (bottom) of the best optimization combinations for the weak form (N1). *Right column:* Roofline (top) and MDoF/s (bottom) with speed-ups of the intermediate operators during the optimization process to obtain N1.SVICT. The accumulated speed-up is 11 $\times$ .

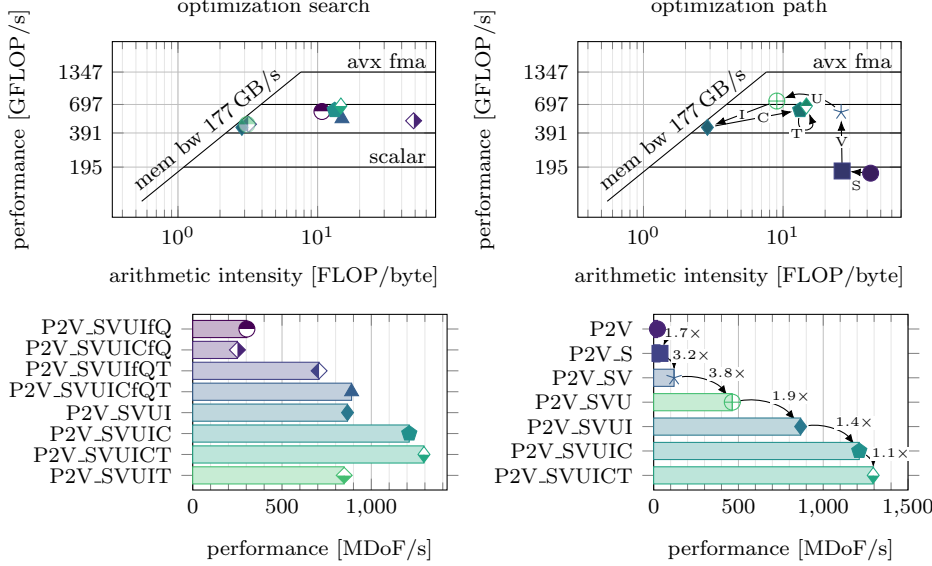
**7.3. Curl-Curl and Mass.** Next, we evaluate a linear combination of the curl-curl and mass operator [21]:  $\alpha(\mathbf{x}) \mathbf{curl} \mathbf{curl} + \beta(\mathbf{x})$ . Test and trial space are Nédélec elements of first order and first kind [38]. We assume that  $\alpha(\mathbf{x}), \beta(\mathbf{x}) \in \mathcal{K}$ , a discrete FE space used to approximate the coefficient function and that setting  $\mathcal{K} = \mathcal{P}_1(\mathcal{T}_m(\Omega))$  yields a sufficient approximation for smooth coefficients. In the Nédélec space  $\mathcal{ND}_1(\mathcal{T}_m(\Omega))$ , DoFs are associated with the edges of the element, yielding  $10^9$  DoFs on the test cube. The corresponding basis functions are vector-valued and ensure continuity across elements in tangential direction [38].

The associated local operator is defined as

$$(N1) \quad A_{T_m} = \left[ \int_{T_m} \alpha(\mathbf{x}) (\mathbf{curl} \phi^i \cdot \mathbf{curl} \psi^j) + \beta(\mathbf{x}) (\phi^i \cdot \psi^j) \right]_{j \in \mathcal{I}_{T_m}^{\mathcal{ND}_1}, i \in \mathcal{I}_{T_m}^{\mathcal{ND}_1}}.$$

(N1) operators exhibit vastly superior performance than those from (P1) with up to 62% of the AVX FMA peak performance and 2.1 GDoF/s (Figure 8, left column). The integrand is more compute-intensive, placing the best operators generally at higher arithmetic intensities.

The standard optimizations show a familiar pattern in terms of performance, mirroring what we observed for (P1), as visible in Figure 8, right column: a sharp performance spike from vectorization (V), followed by a shift towards memory boundness by drawing loop invariants (I) and then a return to the compute-bound region after applying the cubes loop strategy (C). These optimizations culminate in operator N1.SVIC. At this point, tabulation (T) substantially reduces the computational workload and enhances performance by nearly 500 MDoF/s (Figure 8, lower right). Two tables are assembled, one for the curl-curl and mass part, respectively. This is necessary due to the two spatially-varying coefficients in the weak form. Com-



**Figure 9** *Left column:* Roofline (top) and MDoF/s (bottom) of the best optimization combinations for the weak form (P2V). *Right column:* Roofline (top) and MDoF/s (bottom) with speed-ups of the intermediate operators during the optimization process to obtain P2V\_SVUICT. The accumulated speed-up is 58x.

binning unrolled quadrature loops and tabulation emerges as the most effective set of optimizations with an accumulated 11x speed-up in N1\_SVICT.

**7.4. Variable-Coefficient Diffusion.** A variable coefficient diffusion operator  $\nabla \cdot (k(\mathbf{x})\nabla)$  using  $\mathcal{V} = \mathcal{W} = \mathcal{K} = \mathcal{P}_2(\mathcal{T}_m(\Omega))$  represents more compute-intense operators. This discretization leads to  $8.9 \times 10^9$  DoFs on the test cube. A similar form arises from the Taylor-Hood ( $\mathcal{P}_2 - \mathcal{P}_1$ ) discretization of the Stokes equation in the case of a spatially-varying viscosity [6]. The local matrix is

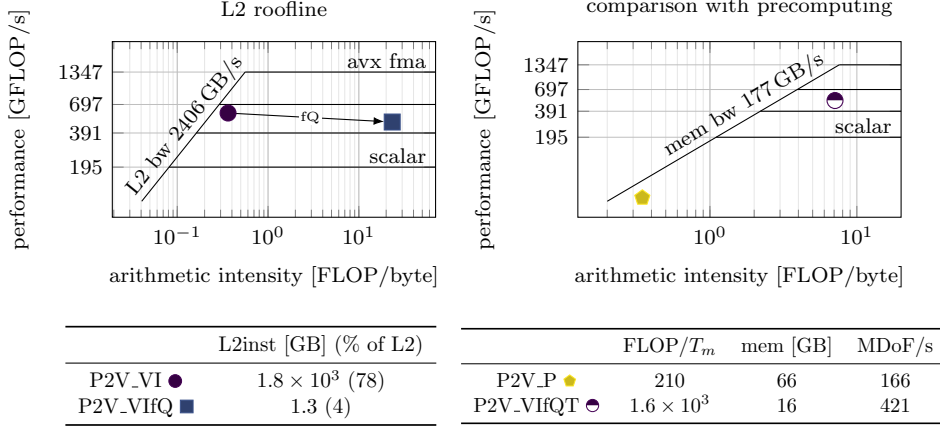
$$(P2V) \quad A_{T_m} = \left[ \int_{T_m} k(\mathbf{x}) (\nabla \phi^i \cdot \nabla \psi^j) \right]_{j \in \mathcal{I}_{T_m}^{P_2}, i \in \mathcal{I}_{T_m}^{P_2}}.$$

The left column of Figure 9 shows that a similar set of optimizations as for (N1), including the standard optimizations, unrolled quadrature loops and tabulation proves most effective also for (P2V). Operator P2V\_SVUICT reaches 1.3 GDoF/s and 50 % AVX FMA peak performance.

We observe the optimization path leading up to P2V\_SVUICT in Figure 9, right column, where additional optimizations are applied together with vectorization (V), moving loop invariants (I) and the cubes loop strategy (C). The latter cause the familiar upwards, left and right shifts in the roofline model.

The integrand of (P2V) is a polynomial of fourth degree due to the quadratic test, trial and coefficient space. Evaluating the integrand on 11 quadrature points defined by Xiao-Gimbutas rule [47] bloats the AST, preventing the CSE to detect symmetry. Explicitly exploiting symmetry in the HOG (S) eliminates 45 of the 100 entries in the local matrix (for 10 shape functions in test and trial space, respectively).

(P2V) offers the option to under-integrate (U) due to the difference in polynomial degree in the integrand and the shape functions. By applying the Xiao-Gimbutas rule of second order instead of fourth order, the number of quadrature points reduces from



**Figure 10** *Left column:* L2 roofline and L2 cache volume induced by loading instructions. The fraction of the L2 instruction volume with respect to the overall L2 cache volume (load and store) is annotated in parentheses. *Right column:* Roofline, FLOPS per element, main memory volume and performance compared against precomputation of local element matrices.

11 to 4, with a corresponding speed-up. We verified that the proper convergence rate is still achieved when solving analytical test cases on multiple refinement levels using P2V\_SVU⊕. Note that using a quadrature rule merely exact for linear polynomials destroys the convergence order.

By various optimizations reducing the arithmetic intensity, the HOG has transformed the compute-intense P2V●, placed deep in the compute-bound region, into the memory-bound P2V\_SVUI◆. The memory volume becomes the operator’s bottleneck, so reducing it by the cubes loop strategy provides another significant speed-up, accumulating to  $58\times$  in the final operator P2V\_SVUICT◆.

**7.4.1. Instruction Boundedness.** So far, unrolling the quadrature loops and applying CSE to the resulting statements has proved effective, entering the fastest operators for all weak forms. However, during experiments we observed that as compute-intensity of the operators increases, unrolling can cause a non-obvious bottleneck.

We generate P2V\_VI●, which unrolls the quadrature loop, does not tabulate and uses a quadrature rule exact for polynomials of fourth degree. To reveal the bottleneck, we require an L2-roofline where the memory roof and arithmetic intensity are computed using the L2-cache bandwidth. We do not under-integrate here for demonstration purpose, as it would obscure the analysis of the bottleneck.

The L2 roofline (Figure 10, upper left) shows that the operator P2V\_VI● lies very close to the L2-bandwidth roof. It suffers from a very large L2 cache volume, which does not stem from data transfers but instructions being loaded from L2 to L1 cache (Figure 10, lower left). P2V\_VI● is *instruction-bound*. Unrolling the quadrature loops with many points and a more involved weak form bloats the number of statements in the kernel body. Consequently, a large amount of distinct instructions has to be fetched, causing the high L2 cache volume. Simply not unrolling the quadrature loop reduces the number of statements in the kernel body and the instruction traffic to negligible amounts, alleviating the instruction-boundedness with P2V\_VIfQ■.

**7.4.2. Comparison with Precomputation of Local Element Matrices.** Given that sufficient memory is available, the local matrix for each element of the

grid can be precomputed, loaded and applied during the loop [13, 12]. We compare our generated operators against this precomputation technique for completeness and because it seems to be commonly applied to variable-coefficient problems in FE simulations. We establish a fair comparison against our implementation of precomputation by not exploiting symmetry and using the sawtooth loop strategy in both operators.

The right column of Figure 10 demonstrates the drawback of precomputing local matrices: although P2V\_P $\blacklozenge$  does relatively few FLOPs, it suffers from a large memory load from the local matrices, which renders it extremely memory-bound. For reference, the (P1) operator with the lowest arithmetic intensity, P1\_SVIfQ $\bullet$ , lies at 1.5 FLOP/byte, while P2V\_P $\blacklozenge$  places at just 0.3 FLOP/byte. P2V\_VIfQT $\bullet$  on the other hand assembles on the fly, does a magnitude more computation, but places well into the machine balance. Thereby, it runs  $2.5\times$  faster.

**8. Scaling Curl-Curl to a Trillion DoFs.** Finally, we demonstrate the weak scalability of the generated operators within the HYTeG framework.

The (homogeneous) curl-curl problem

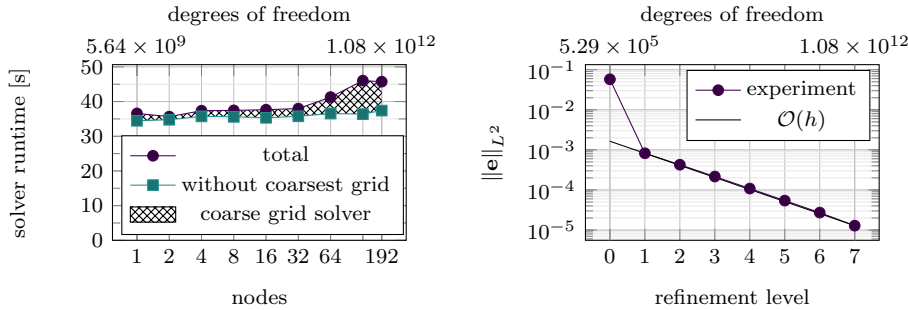
$$(8.1) \quad \begin{aligned} \alpha \mathbf{curl} \mathbf{curl} \mathbf{u} + \beta \mathbf{u} &= \mathbf{f} \quad \text{in } \Omega, \\ \mathbf{u} \times \mathbf{n} &= 0 \quad \text{on } \partial\Omega, \end{aligned}$$

in three dimensions, with given  $\mathbf{f} \in \mathbf{L}^2(\Omega)$ , and  $\alpha, \beta \in L^2(\Omega)$ , arises from Maxwell's equations in electromagnetic wave scattering problems [21]. We discretize the solution  $\mathbf{u}$  and right-hand side (RHS)  $\mathbf{f}$  with first order, first kind Nédélec elements ( $\mathcal{N}\mathcal{D}_1$ ). The corresponding matrix-free operator we use is N1\_SVIC $\blacklozenge$  from Section 7.3, implementing the weak form (N1).

To assess the weak scalability, we solve (8.1) on the unit cube with an equally growing number of processes and coarse grid elements. In all runs there is a total of eight levels in the grid hierarchy. For simplicity, the coefficients  $\alpha(\mathbf{x})$  and  $\beta(\mathbf{x})$  equal 1 for all  $\mathbf{x}$ . Nonetheless, the operators treat them as if they are spatially varying.

The system is solved using matrix-free full multigrid (FMG) with a single V(1,1) cycle on each level. The hybrid smoother published in [21] is used due to the non-elliptic nature of the bilinear form. In short, smoothing is split into two sub-steps. While a standard smoother in  $\mathcal{N}\mathcal{D}_1$  reduces error components in  $\mathcal{N}(\mathbf{curl})^\perp$ , the nullspace is handled by relaxing on Poisson's equation in potential space, which is discretized using  $\mathcal{P}_1$  FEs. We choose Chebyshev smoothers of order 2 [1, 3] in both spaces. This means that one hybrid smoothing step requires in total three matrix-vector products in  $\mathcal{N}\mathcal{D}_1$ , two matrix-vector products in  $\mathcal{P}_1$ , and two transfer operations between the spaces. Note that two additional  $\mathcal{P}_1$  vectors must be allocated. The  $\mathcal{P}_1$  operator is generated with the same set of optimizations as the  $\mathcal{N}\mathcal{D}_1$  operator (SVIC). On the coarsest grid the matrix is assembled, and PETSc's [4] SOR preconditioned CG solver solves the system up to a relative residual reduction of  $10^{-3}$ .

The experiment is performed on the SuperMUC-NG Phase 2 cluster [33]. Each node comprises two Intel Xeon Platinum 8480+ (Sapphire Rapids) CPUs with 56 cores each and 512 GB DDR5 memory. Note that this architecture differs from the one used in the previous section. Weak scaling results from 1 to 192 nodes (112 to 21 504 cores) are summarized in Figure 11 (left). Near perfect scaling is seen up to 32 nodes. Starting with 64 nodes, the coarse grid solver starts to have a significant impact on the overall solve time. Presumably this is due to, first, the growing coarse grid size and suboptimal scaling of the CG-solver, and second, the low computational complexity compared to the high communication cost. It is expected that for larger problems the coarse grid solver will have a significant impact on the time to solution.



**Figure 11** *Left: Weak scaling of HyTEG’s FMG solver is almost perfect from 1 to 32 nodes. In larger runs the coarse grid solver (not optimized in this work) starts having significant impact. The remaining parts of the multigrid solver keep scaling well. Right:  $L^2$ -error grid convergence for the largest case (192 nodes, 21 504 parallel processes,  $1.08 \times 10^{12}$  DoFs on level 7).*

In previous work [11] it was shown how advanced sparse direct solvers can be employed to alleviate this problem. Alternatives include using parallel AMG methods, such as, e.g., hypre [15] as scalable coarse grid solvers. However, the systematic analysis of these alternatives is beyond the scope of the present article and will be studied in future work. On the other hand, the remaining part of the solver scales very well, even up to a trillion ( $10^{12}$ ) DoFs.

To assert that the solver finds the correct solution, the  $L^2$ -error against the known manufactured solution is computed. Figure 11 (right) shows that the error reduces at the expected linear rate [48].

**9. Conclusion.** The presented code generator is a powerful tool for the rapid development of matrix-free FE operators. A wide range of optimizations can be applied “at the press of a button” to any weak form and a wide array of matrix-free operations. Notable operations include matrix-vector products, the assembly of an inverse diagonal for Chebychev smoothers and diagonally lumped operators. Code generation is a vital tool to ensure sustainable high performance even for future applications and hardware architectures.

Using the HOG, high node-level performance can be obtained quickly even for complex problems on extreme scales, e.g., the simulation of whole-planet Earth mantle convection. In such simulations, the reduction in runtime resulting from HOG’s optimizations ultimately saves huge amounts of energy and money.

Through rigorous and systematic analysis surprisingly large speed-ups over existing production codes can be achieved. We found that traditional tools from performance engineering like the roofline model are essential for the identification of bottlenecks. Additionally, they make it possible to classify codes as “slow” or “fast”. In combination with code generation, a multitude of optimizations and combinations thereof can be evaluated with low effort to find an optimum.

Future extensions of the HOG include support for curvilinear boundaries, discontinuous Galerkin methods, vector function spaces, surrogate operators and multigrid grid-transfer operators. The abstract intermediate representation during generation time enables the development of additional backends for the automated generation of kernels for accelerators, particularly GPUs.

**Acknowledgments.** The authors gratefully acknowledge funding through the joint BMBF project CoMPS<sup>3</sup> (grant 16ME0647K). The authors would like to thank the

<sup>3</sup><https://gauss-allianz.de/en/project/title/CoMPS>

NHR-Verein e.V.<sup>4</sup> for supporting this work/project within the NHR Graduate School of National High Performance Computing (NHR). The authors gratefully acknowledge the scientific support and HPC resources provided by the Erlangen National High Performance Computing Center (NHR@FAU) of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU). NHR funding is provided by federal and Bavarian state authorities. NHR@FAU hardware is partially funded by the German Research Foundation (DFG) – 440719683. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V.<sup>5</sup> for funding this project by providing friendly user access during the pilot operation of the GCS Supercomputer SuperMUC-NG Phase 2 at Leibniz Supercomputing Centre<sup>6</sup>.

## REFERENCES

- [1] M. ADAMS, M. BREZINA, J. HU, AND R. TUMINARO, *Parallel multigrid smoothing: Polynomial versus Gauss–Seidel*, Journal of Computational Physics, 188 (2003), pp. 593–610, [https://doi.org/10.1016/S0021-9991\(03\)00194-3](https://doi.org/10.1016/S0021-9991(03)00194-3).
- [2] M. ALNÆS, J. BLECHTA, J. HAKE, A. JOHANSSON, B. KEHLET, A. LOGG, C. RICHARDSON, J. RING, M. ROGNES, AND G. WELLS, *The FEniCS project version 1.5*, Arch. Num. Soft., 3 (2015).
- [3] A. H. BAKER, R. D. FALGOUT, T. V. KOLEV, AND U. M. YANG, *Multigrid Smoothers for Ultraparallel Computing*, SIAM J. Sci. Comput., 33 (2011), pp. 2864–2887, <https://doi.org/10.1137/100798806>.
- [4] S. BALAY, W. D. GROPP, L. C. MCINNES, AND B. F. SMITH, *Efficient management of parallelism in object oriented numerical software libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhäuser Press, 1997, pp. 163–202, [https://doi.org/10.1007/978-1-4612-1986-6\\_8](https://doi.org/10.1007/978-1-4612-1986-6_8).
- [5] M. BAUER, J. HÖTZER, D. ERNST, J. HAMMER, M. SEIZ, H. HIERL, J. HÖNIG, H. KÖSTLER, G. WELLEIN, B. NESTLER, AND U. RÜDE, *Code generation for massively parallel phase-field simulations*, in SC '19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE Computer Society, 11 2019, pp. 1–32, <https://doi.org/10.1145/3295500.3356186>.
- [6] S. BAUER, H.-P. BUNGE, D. DRZISGA, S. GHELICHKHAN, M. HUBER, N. KOHL, M. MOHR, U. RÜDE, D. THÖNNES, AND B. WOHLMUTH, *TerraNeo — Mantle Convection Beyond a Trillion Degrees of Freedom*, in Softw. Exascale Comput. - SPPEXA 2016-2019, H.-J. Bungartz, S. Reiz, B. Uekermann, P. Neumann, and W. Nagel, eds., vol. 136 of Lecture Notes in Computational Science and Engineering, Springer, 2020, pp. 569–610, [https://doi.org/10.1007/978-3-030-47956-5\\_19](https://doi.org/10.1007/978-3-030-47956-5_19).
- [7] B. K. BERGEN AND F. HÜLSEMAN, *Hierarchical hybrid grids: Data structures and core algorithms for multigrid*, Numer. Linear Algebra Appl., 11 (2004), pp. 279–291, <https://doi.org/10.1002/nla.382>.
- [8] J. BEY, *Tetrahedral grid refinement*, Computing, 55 (1995), pp. 355–378, <https://doi.org/10.1007/BF02238487>.
- [9] S. C. BRENNER AND L. R. SCOTT, *The Mathematical Theory of Finite Element Methods*, Springer-Verlag, 2008, <https://doi.org/10.1007/978-0-387-75934-0>.
- [10] J. BROWN, *Efficient Nonlinear Solvers for Nodal High-Order Finite Elements in 3D*, Journal of Scientific Computing, 45 (2010), pp. 48–63, <https://doi.org/10.1007/s10915-010-9396-8>.
- [11] A. BUTTARI, M. HUBER, P. LELEUX, T. MARY, U. RÜDE, AND B. WOHLMUTH, *Block low-rank single precision coarse grid solvers for extreme scale multigrid methods*, Numerical Linear Algebra with Applications, 29 (2022), <https://doi.org/10.1002/nla.2407>.
- [12] G. F. CAREY, E. J. BARRAGY, R. T. MCLAY, AND M. SHARMA, *Element-by-element vector and parallel computations*, Communications in Applied Numerical Methods, 4 (1988), pp. 299–307, <https://api.semanticscholar.org/CorpusID:120879806>.
- [13] G. F. CAREY AND B.-N. JIANG, *Element-by-element linear and nonlinear solution schemes*, International Journal for Numerical Methods in Biomedical Engineering (Formerly: Com-

<sup>4</sup><https://www.nhr-verein.de>

<sup>5</sup><https://www.gauss-centre.eu>

<sup>6</sup><https://www.lrz.de>

- munications in Numerical Methods in Engineering; Communications in Applied Numerical Methods), 2 (1986), pp. 145–153, <https://doi.org/10.1002/cnm.1630020205>.
- [14] P. G. CIARLET, *The Finite Element Method for Elliptic Problems*, Society for Industrial and Applied Mathematics, 2002, <https://doi.org/10.1137/1.9780898719208>.
  - [15] R. D. FALGOUT AND U. M. YANG, *hypre: A library of high performance preconditioners*, in Computational Science — ICCS 2002, P. M. A. Sloot, A. G. Hoekstra, C. J. K. Tan, and J. J. Dongarra, eds., 2002, pp. 632–641, [https://doi.org/10.1007/3-540-47789-6\\_66](https://doi.org/10.1007/3-540-47789-6_66).
  - [16] B. GMEINER, M. HUBER, L. JOHN, U. RÜDE, AND B. WOHLMUTH, *A quantitative performance study for Stokes solvers at the extreme scale*, Journal of Computational Science, 17 (2016), pp. 509–521, <https://doi.org/10.1016/j.jocs.2016.06.006>.
  - [17] B. GMEINER, U. RÜDE, H. STENGEL, C. WALUGA, AND B. WOHLMUTH, *Performance and Scalability of Hierarchical Hybrid Multigrid Solvers for Stokes Systems*, SIAM J. Sci. Comput., 37 (2015), pp. C143–C168, <https://doi.org/10.1137/130941353>.
  - [18] G. HAGER AND G. WELLEIN, *Introduction to High Performance Computing for Scientists and Engineers*, CRC Press, 1 ed., July 2010, <https://doi.org/10.1201/EBK1439811924>.
  - [19] D. A. HAM, P. H. J. KELLY, L. MITCHELL, C. J. COTTER, R. C. KIRBY, K. SAGIYAMA, N. BOUZIANI, S. VORDERWUELBECKE, T. J. GREGORY, J. BETTERIDGE, D. R. SHAPERO, R. W. NIXON-HILL, C. J. WARD, P. E. FARRELL, P. D. BRUBECK, I. MARSDEN, T. H. GIBSON, M. HOMOLYA, T. SUN, A. T. T. MCRAE, F. LUPORINI, A. GREGORY, M. LANGE, S. W. FUNKE, F. RATHGEER, G.-T. BERCEA, AND G. R. MARKALL, *Firedrake User Manual*, Imperial College London and University of Oxford and Baylor University and University of Washington, first edition ed., 5 2023, <https://doi.org/10.25561/104839>.
  - [20] J. HAMMER, J. EITZINGER, G. HAGER, AND G. WELLEIN, *Kerncraft: A Tool for Analytic Performance Modeling of Loop Kernels*, Springer International Publishing, 2017, pp. 1–22, [https://doi.org/10.1007/978-3-319-56702-0\\_1](https://doi.org/10.1007/978-3-319-56702-0_1).
  - [21] R. HIPTMAIR, *Multigrid Method for Maxwell’s Equations*, SIAM J. Numer. Anal., 36 (1998), pp. 204–225, <https://doi.org/10.1137/S0036142997326203>.
  - [22] A. HOSANGADI, F. FALLAH, AND R. KASTNER, *Optimizing polynomial expressions by algebraic factorization and common subexpression elimination*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 25 (2006), pp. 2012–2021, <https://doi.org/10.1109/TCAD.2006.875712>.
  - [23] T. J. HUGHES, L. P. FRANCA, AND M. BALESTRA, *A new finite element formulation for computational fluid dynamics: V. circumventing the Babuška-Brezzi condition: a stable Petrov-Galerkin formulation of the stokes problem accommodating equal-order interpolations*, Computer Methods in Applied Mechanics and Engineering, 59 (1986), pp. 85–99, [https://doi.org/10.1016/0045-7825\(86\)90025-3](https://doi.org/10.1016/0045-7825(86)90025-3).
  - [24] R. C. KIRBY AND A. LOGG, *A compiler for variational forms*, ACM Transactions on Mathematical Software, 32 (2006), <https://doi.org/10.1145/1163641.1163644>.
  - [25] R. C. KIRBY AND L. MITCHELL, *Solver composition across the pde/linear algebra barrier*, SIAM Journal on Scientific Computing, 40 (2018), p. C76–C98, <https://doi.org/10.1137/17m1133208>.
  - [26] N. KOHL, D. BAUER, F. BÖHM, AND U. RÜDE, *Fundamental data structures for matrix-free finite elements on hybrid tetrahedral grids*, Int. J. Parallel Emergent Distrib. Syst., 39 (2024), pp. 51–74, <https://doi.org/10.1080/17445760.2023.2266875>.
  - [27] N. KOHL, M. MOHR, S. EIBL, AND U. RÜDE, *A Massively Parallel Eulerian-Lagrangian Method for Advection-Dominated Transport in Viscous Fluids*, SIAM J. Sci. Comp., 44 (2022), pp. C260–C285, <https://doi.org/10.1137/21M1402510>.
  - [28] N. KOHL AND U. RÜDE, *Textbook Efficiency: Massively Parallel Matrix-Free Multigrid for the Stokes System*, SIAM J. Sci. Comput., 44 (2022), pp. C124–C155, <https://doi.org/10.1137/20M1376005>.
  - [29] N. KOHL, D. THÖNNES, D. DRZISGA, D. BARTUSCHAT, AND U. RÜDE, *The HyTeG finite-element software framework for scalable multigrid solvers*, International Journal of Parallel, Emergent and Distributed Systems, 34 (2019), pp. 477–496, <https://doi.org/10.1080/17445760.2018.1506453>.
  - [30] M. KRONBICHLER AND K. KORMANN, *A generic interface for parallel cell-based finite element operator application*, Computers and Fluids, 63 (2012), pp. 135–147, <https://doi.org/10.1016/j.compfluid.2012.04.012>.
  - [31] M. KRONBICHLER AND K. KORMANN, *Fast Matrix-Free Evaluation of Discontinuous Galerkin Finite Element Operators*, ACM Trans. Math. Softw., 45 (2019), pp. 1–40, <https://doi.org/10.1145/3325864>.
  - [32] M. KRONBICHLER AND W. A. WALL, *A Performance Comparison of Continuous and Discontinuous Galerkin Methods with Fast Multigrid Solvers*, SIAM J. Sci. Comput., 40 (2018),

- pp. A3423–A3448, <https://doi.org/10.1137/16M110455X>.
- [33] LEIBNIZ-RECHENZENTRUM (LRZ), *Pilot operation SuperMUC-NG Phase 2*, <https://doku.lrz.de/pilot-operation-supermuc-ng-phase-2-403079197.html> (accessed 2024-03-02).
  - [34] C. LENGAUER, S. APEL, M. BOLTEN, S. CHIBA, U. RÜDE, J. TEICH, A. GRÖSSLINGER, F. HANNIG, H. KÖSTLER, L. CLAUS, ET AL., *Exastencils: Advanced multigrid solver generation*, in Software for Exascale Computing - SPPEXA 2016-2019, Cham, 2020, Springer International Publishing, pp. 405–452, [https://doi.org/10.1007/978-3-030-47956-5\\_14](https://doi.org/10.1007/978-3-030-47956-5_14).
  - [35] D. MAY, J. BROWN, AND L. LE POURHIET, *PTatin3D: High-Performance Methods for Long-Term Lithospheric Dynamics*, in SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 11 2014, <https://doi.org/10.1109/SC.2014.28>.
  - [36] D. MAY, J. BROWN, AND 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), pp. 496–523, <https://doi.org/10.1016/j.cma.2015.03.014>.
  - [37] A. MEURER, C. P. SMITH, M. PAPROCKI, O. ČERTÍK, S. B. KIRPICHEV, M. ROCKLIN, A. KUMAR, S. IVANOV, J. K. MOORE, S. SINGH, T. RATHNAYAKE, S. VIG, B. E. GRANGER, R. P. MULLER, F. BONAZZI, H. GUPTA, S. VATS, F. JOHANSSON, F. PEDREGOSA, M. J. CURRY, A. R. TERREL, V. ROUČKA, A. SABOO, I. FERNANDO, S. KULAL, R. CIMRMAN, AND A. SCOPATZ, *Sympy: symbolic computing in python*, PeerJ Computer Science, 3 (2017), p. e103, <https://doi.org/10.7717/peerj-cs.103>.
  - [38] J. C. NEDELEC, *Mixed finite elements in  $\mathbb{R}^3$* , Numer. Math., 35 (1980), pp. 315–341, <https://doi.org/10.1007/bf01396415>.
  - [39] NHR@FAU, *Fritz Parallel Cluster*. <https://hpc.fau.de/systems-services/documentation-instructions/clusters/fritz-cluster>. Accessed: 2024/01/12.
  - [40] J. RUDI, A. C. I. MALOSI, T. ISAAC, G. STADLER, M. GURNIS, P. W. J. STAAR, Y. INEICHEN, C. BEKAS, A. CURIONI, AND O. GHATTAS, *An extreme-scale implicit solver for complex pdes: highly heterogeneous flow in earth's mantle*, in SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2015, pp. 1–12, <https://doi.org/10.1145/2807591.2807675>.
  - [41] N. SCHLÖMER, N. PAPIOR, D. ARNOLD, J. BLECHTA, AND R. ZETTER, *nshloe/quadpy: None (v0.16.10)*. Zenodo, <https://doi.org/10.5281/zenodo.5541216>.
  - [42] H. STENGEL, J. TREIBIG, G. HAGER, AND G. WELLEIN, *Quantifying performance bottlenecks of stencil computations using the execution-cache-memory model*, in ICS '15: Proceedings of the 29th ACM on International Conference on Supercomputing, 2015, p. 207–216, <https://doi.org/10.1145/2751205.2751240>.
  - [43] T. SUN, L. MITCHELL, K. KULKARNI, A. KLÖCKNER, D. A. HAM, AND P. H. J. KELLY, *A study of vectorization for matrix-free finite element methods*, Int. J. of High Perf. Comp. App., (2019), <https://doi.org/10.1177/1094342020945005>.
  - [44] D. THÖNNES AND U. RÜDE, *Model-based performance analysis of the HyTeG finite element framework*, in Proceedings of the Platform for Advanced Scientific Computing Conference, PASC '23, New York, NY, USA, 2023, Association for Computing Machinery, <https://doi.org/10.1145/3592979.3593422>.
  - [45] J. TREIBIG, G. HAGER, AND 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, <https://doi.org/10.1109/ICPPW.2010.38>.
  - [46] S. WILLIAMS, A. WATERMAN, AND D. PATTERSON, *Roofline: An insightful visual performance model for multicore architectures*, Commun. ACM, 52 (2009), pp. 65–76, <https://doi.org/10.1145/1498765.1498785>.
  - [47] H. XIAO AND Z. GIMBUTAS, *A numerical algorithm for the construction of efficient quadrature rules in two and higher dimensions*, Computers and Mathematics with Applications, 59 (2010), pp. 663–676, <https://doi.org/10.1016/j.camwa.2009.10.027>.
  - [48] L. ZHONG, S. SHU, G. WITTUM, AND J. XU, *Optimal error estimates for Nédélec edge elements for time-harmonic Maxwell's equations*, J. Comp. Math., 27 (2009), pp. 563–572, <https://doi.org/10.4208/jcm.2009.27.5.011>.
  - [49] K. B. ØLGAARD AND G. N. WELLS, *Optimizations for quadrature representations of finite element tensors through automated code generation*, ACM Transactions on Mathematical Software, 37 (2010), <https://doi.org/10.1145/1644001.1644009>.