A Next-Generation Discontinuous Galerkin Fluid Dynamics Solver with Application to High-Resolution Lung Airflow Simulations

Martin Kronbichler
kronbichler@lnm.mw.tum.de
Technical University of Munich
Garching, Germany
Uppsala University
Uppsala, Sweden

Niklas Fehn
niklas.fehn@lrz.de
Leibniz Supercomputing Centre
Garching, Germany

Peter Munch
munch@lnm.mw.tum.de
Technical University of Munich
Garching, Germany
Helmholtz-Zentrum hereon GmbH
Geesthacht, Germany

Maximilian Bergbauer
bergbauer@lnm.mw.tum.de
Technical University of Munich
Garching, Germany

Karl-Robert Wichmann
wichmann@lnm.mw.tum.de
Technical University of Munich
Garching, Germany
Ebenbuild GmbH
Garching, Germany

Carolin Geitner
geitner@lnm.mw.tum.de
Technical University of Munich
Garching, Germany

Momme Allalen
momme.allalen@lrz.de
Leibniz Supercomputing Centre
Garching, Germany

Martin Schulz
Wolfgang A. Wall
schulzm@in.tum.de
wall@lnm.mw.tum.de
Technical University of Munich
Garching, Germany

ABSTRACT

We present a novel, highly scalable and optimized solver for turbulent flows based on high-order discontinuous Galerkin discretizations of the incompressible Navier–Stokes equations aimed to minimize time-to-solution. The solver uses explicit-implicit time integration with variable step size. The central algorithmic component is the matrix-free evaluation of discretized finite element operators. The node-level performance is optimized by sum-factorization kernels for tensor-product elements with unique algorithmic choices that reduce the number of arithmetic operations, improve cache usage, and vectorize the arithmetic work across elements and faces. These ingredients are integrated into a framework scalable to the massive parallelism of supercomputers by the use of optimal-complexity linear solvers, such as mixed-precision, hybrid geometric-polynomial-algebraic multigrid solvers for the pressure Poisson problem. The application problem under consideration are fluid dynamical simulations of the human respiratory system under mechanical ventilation conditions, using unstructured/structured adaptively refined meshes for geometrically complex domains typical of biomedical engineering.

CCS CONCEPTS

• **Mathematics of computing** → **Mathematical software performance**; *Solvers*; • **Applied computing** → *Health informatics*.

KEYWORDS

high-order discontinuous Galerkin, matrix-free algorithms, multigrid, time-to-solution

ACM Reference Format:
Martin Kronbichler, Niklas Fehn, Peter Munch, Maximilian Bergbauer, Karl-Robert Wichmann, Carolin Geitner, Momme Allalen, Martin Schulz, and Wolfgang A. Wall. 2021. A Next-Generation Discontinuous Galerkin Fluid Dynamics Solver with Application to High-Resolution Lung Airflow Simulations. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21), November 14–19, 2021, St. Louis, MO, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3458817.3476171

SC '21, November 14–19, 2021, St. Louis, MO, USA

ACM ISBN 978-1-4503-8442-1/21/11…$15.00
https://doi.org/10.1145/3458817.3476171

1 OVERVIEW OF THE PROBLEM

The complex structure of a human lung extending across various scales and its well balanced functioning make the respiratory system a fascinating organ of the human body: from the upper, purely conducting airways, the air passages continuously branch into deeper regions (see Figure 1) and transition into the respiratory zone consisting of a spongy arrangement of alveoli, which finally are involved in the exchange of oxygen. Considering this structural

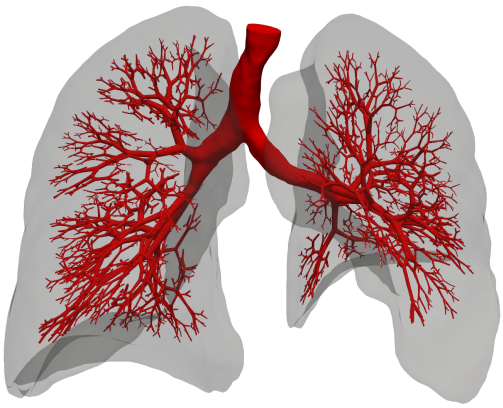

Figure 1: Visualization of an airway tree of the first 12 generations (red) continuously branching into both lungs (grey). The larger airways and the lungs are segmented from medical image data, airways below CT image resolution are generated by a morphologically based tree growing algorithm.

delicacy and the direct exposure to the external environment, it is not surprising that pulmonary diseases are rated among the major causes of global morbidity and mortality [26], with increased attention nowadays due to the ongoing COVID-19 pandemic.

Improved insight into the respiratory system is crucial for developing new treatments or the personalized application of existing therapies for a large number of severe diseases, in particular those requiring mechanical ventilation. Although a life-preserving measure for some diseases, mechanical ventilation places additional stress on already damaged lungs, potentially resulting in so-called ventilator induced lung injury (VILI), and the patient's chances of survival drastically decrease with the duration of use [17]. Since even small alterations in ventilator settings can greatly reduce mortality [10], a better understanding of the lung (patho-)physiology is crucial.

A reoccurring problem is that the variety of complex and interacting phenomena, like fluid dynamical, tissue mechanical, and gas transport processes, are difficult to measure *in vivo* due to both ethical and technical reasons. The application of computational models for *in silico* investigations have great potential to deepen the general knowledge about fluid dynamics in the lung and to reveal certain phenomena which are, so far, inaccessible *in vivo* or *in vitro*. New evidence from simulations offers the chance to improve general medical treatment and to even individualize therapies by patient-specific modeling of ventilation, inhalation, or surfactant therapies.

Today, only very simplified models could realistically be used bedside in clinical settings. For a new generation of computation-assisted treatment guaranteeing a high level of safety, advanced high-fidelity models need to be investigated for a multitude of phenomena and parameters. Engineering practice typically requires a single simulation model to be realized in a matter of minutes to hours. However, conducting scale-resolving simulations of turbulent flows in wall-time limits of a few hours is one of the major challenges in computational fluid dynamics due to the large number of time steps (typically millions of time steps) required by such complex problems. Even on today's supercomputers, such simulations often require wall times of weeks to months. This situation is sometimes denoted as the LES (large-eddy simulation) crisis [47], based on extrapolations made about 3 decades ago regarding the availability of such simulations in the future, and recognizing that this goal has not been reached today.[1] Against this background, the need for computationally efficient methods minimizing time-to-solution in the strong-scaling limit is obvious.

The present work aims at realizing resolutions of the tracheobronchial tree that allow to fully resolve the resistance of the conducting airway tree by the 3D simulation model, i.e., the complete airway tree up to 11 generations of the Weibel model [60], thereby exceeding the resolution limits of state-of-the-art approaches (Section 2.1). For this purpose, we optimized ExaDG [21], a high-performance turbulent flow solver based on high-order discontinuous Galerkin discretizations of the incompressible Navier–Stokes equations. The solver is based on mixed explicit-implicit time integration with variable step sizes, enhanced with several optimizations enabling us to tackle its computational complexity. We make the following contributions:

- We provide a highly scalable simulation code for improving the predictive simulation capabilities of the human pulmonary system, which can aid in the treatment of many diseases, including COVID-19.
- We use unstructured hexahedral coarse meshes with structured refinement for hybrid patient-specific/idealized lung models.
- We introduce several algorithmic optimizations, including matrix-free operator evaluation, which enable us to better balance computation and memory accesses.
- We discuss a series of architectural optimizations to improve cache usage and vectorization, allowing us to fully exploit modern architectures.
- We present a multigrid framework on top of these node-level improvements to efficiently solve the most challenging part, the pressure Poisson equation, on large scale HPC systems.
- We demonstrate our work on use cases targeting realistic mechanical ventilation conditions.

Our experiments on the SuperMUC-NG system at the Leibniz Supercomputing Centre in Garching, Germany show that run times around or below 0.1 seconds per time step can be achieved for geometrically complex large-scale simulations.

2 BACKGROUND AND STATE OF THE ART

2.1 Comp. methods in respiratory mechanics

The complexity of the conducting airway tree of the human lung – growing exponentially with the number of splits, or generations g – requires some form of truncation when approached by computational techniques. Truncating the airway tree in three-dimensional

[1] It is important to realize that this problem can not be overcome by the use of a "larger" supercomputer.

flow simulations mainly has two reasons: (i) limitations in state-of-the-art imaging technologies (CT and MRI) that do not allow to resolve the geometry of smaller airways in higher generations of the human lung; and (ii) limitations regarding computational fluid dynamics software, including mesh generation tools and the computational efficiency of flow solvers. Regarding the first aspect, state-of-the-art image-based models appear to be limited to 6–8 airway generations for adults, see [13, 54], and somewhat lower values of 3–4 for neonates, see [55]. Regarding the second aspect, the highest number of terminal (or peripheral) airways resolved in 3D flow simulations is approximately 70 in [13], approximately 100–130 in [54, 55, 59], 451 in [35] for an ovine lung model, and 720 for the 17-generation Horsfield model studied in [29].[2] The present model resolves 1005 terminal airways for $g = 11$ generations.

Resolving the airway resistance of the tracheobronchial tree requires a 3D computational lung model to contain at least the first 10–12 generations, see [15, Figure 3]. Due to resolution limits of medical imaging, the patient-specific geometry of the first airway generations needs to be combined with morphologically correct, idealized cylindrical airway geometries for higher generations, leading to hybrid (patient-specific/idealized) models as used, e.g., in [55]. A resolution of all conducting airways up to 16–18 generations (according to Weibel's model [60]) currently appears to be infeasible. However, some studies aim at overcoming the exponential complexity by resolving only a few branches of the airway tree up to generation 16-18, while truncating the remaining airways at a lower generation number [29, 59].

Due to the complex geometry of the airway tree, meshes composed of tetrahedral elements is the de facto standard. Results published in the literature have typically been obtained on meshes composed of not more than one to ten million tetrahedral elements using low-order finite element approximations, see, e.g., [13, 29, 55].

While simplified computational studies truncate the lung after a certain number of generations with plain outflow boundary conditions, more sophisticated models [13, 55] use suitable boundary conditions at the terminal airways modeling the resistance and compliance of higher airway generations obtained by some physiologically realistic volume-filling 1D tree-growing algorithms.

2.2 Incompressible Navier–Stokes equations

The flow of air through the conducting airways of the human lung can be described by the mathematical model of the incompressible Navier–Stokes equations

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot (\boldsymbol{u} \otimes \boldsymbol{u}) - \nu \nabla^2 \boldsymbol{u} + \nabla p = \boldsymbol{f}\,,$$
$$\nabla \cdot \boldsymbol{u} = 0\,,$$

solved for velocity $\boldsymbol{u}$ and pressure p on a domain $\Omega \subset \mathbb{R}^3$ equipped with suitable initial and boundary conditions. Turbulent flows of high Reynolds number ($\mathrm{Re} = UL/\nu$) involve a wide range of spatial and temporal scales. Hence, an efficient numerical solution poses special requirements in terms of the resolution capabilities of discretization methods. Of high relevance is also the transport of oxygen and carbon dioxide. However, the main challenge lies in the solution of the incompressible Navier–Stokes equations both in terms of numerical discretization techniques and parallel scalability. For this reason, the present work focuses on the incompressible flow part. Note that developments and performance improvements enabling scale-resolving flow simulations are also a prerequisite for accurately predicting the transport of particles (air pollution, pharmaceuticals) in the respiratory system.

[2]We use the number of terminal airways as a measure of complexity of the computational lung model instead of the number of resolved generations, since the latter metric does not characterize complexity well when Weibel [60] and Horsfield [34] classifications are compared.

2.3 High-order discontinuous Galerkin methods

Given the $O(1)$ arithmetic intensity of typical CFD programs in relation to the $O(10)$ Flop-to-Byte ratio of current hardware, memory bandwidth and network latency emerge as the main hardware characteristics limiting performance in computational fluid dynamics. One direction towards optimal use of these limited resources are high-order discretization techniques. Due to their good dispersive and diffusive properties, fewer unknowns and fewer time steps are needed to reach a certain level of accuracy compared to low-order discretizations. From a computational perspective, the data streaming character of low-order CFD software is enriched by additional local computations on cacheable data. Algorithms of minimal computational complexity are crucial in this context to maintain an arithmetic cost small enough to stay within the Flops envelope of the hardware, enabling high-order methods to perform equally efficient (at best) as finite-difference stencils in terms of the number of unknowns processed per second.

The present work relies on discontinuous Galerkin (DG) methods of high (formal) approximation order. DG methods inherit the geometric flexibility and mathematical foundation of finite element methods, i.e., the geometry is approximated by a mesh of (potentially non-conforming) elements of characteristic size h, $\Omega_h = \bigcup_{e=1}^{N} \Omega_e$, and the solution is approximated by a polynomial expansion of degree k within the element. The parametric mapping of a unit element in reference space to a deformed element in physical space is important not only in terms of geometric flexibility, but also in resolving the geometry with high-order of accuracy and in enabling boundary-fitted and boundary-refined meshes particularly relevant in CFD.

Between the elements of a mesh, the approximate solution exhibits discontinuities. Adapting the concept of numerical fluxes from finite volume methods is important in order to treat transport-dominated problems in a stable and accurate manner. These concepts enable theoretically high-order of accuracy, yet preserving the compactness of the stencil when it comes to evaluating discretized differential operators by a sweep through all elements of the mesh, requiring only exchange of data with the nearest neighbors.

This work relies on a DG discretization scheme for the incompressible Navier–Stokes equations developed recently in [25], which is compactly written as

$$\mathcal{M}\left(\frac{\partial \mathsf{U}}{\partial t}\right) + \mathcal{C}\,(\mathsf{U}) + \mathcal{V}\,(\mathsf{U}) + \mathcal{A}_{\mathrm{pen}}\,(\mathsf{U}) + \mathcal{G}\,(\mathsf{P}) = \mathcal{F}\,,$$
$$\mathcal{D}\,(\mathsf{U}) = 0\,.$$

The discrete differential operators act on the solution vectors $\mathsf{U} = U_i, i = 1, \ldots, Nd(k+1)^3$ and $\mathsf{P} = P_i, i = 1, \ldots, Nk^3$ containing all degrees of freedom of the discrete velocity solution of degree k

and pressure solution of degree $k-1$ (inf–sup stability). Section 3.1 details the algorithms used to evaluate discretized PDE operators such as $\mathcal{V}(\mathsf{U})$ by means of matrix-free operator evaluation. This discretization uses the local Lax–Friedrichs flux for the convective term C, the interior penalty method for the viscous term $\mathcal{V}$, and central fluxes for the velocity divergence term $\mathcal{D}$ and pressure gradient term $\mathcal{G}$. The mass operator $\mathcal{M}$ and right-hand side operator $\mathcal{F}$ are local to each element and do not involve numerical fluxes. The additional penalty operator $\mathcal{A}_{\text{pen}}$ weakly enforces the divergence-free constraint in a point-wise manner and continuity of the velocity normal to the interface between elements. This stabilization approach exhibits a close analogy to H^{div}-conforming discretizations [20], i.e., it aims to equip simple L^2-conforming finite element spaces with the robustness of H^{div}-conforming spaces and – at the same time – exploit the fast inversion of the mass operator of L^2-conforming DG methods. The discretization approach described above has been validated in the literature [19, 20, 23, 25].

2.4 Splitting solver for incompressible flows

Our solver makes use of operator-splitting techniques to efficiently advance the incompressible Navier–Stokes equations in time. This leads to a sequence of symmetric positive definite sub-problems to be solved within time step n integrating the equations from time t_n to $t_{n+1} = t_n + \Delta t_n$. We use the high-order dual splitting scheme [36] of temporal order $J = 2$,

$$\frac{\gamma_0^n \hat{\mathsf{U}} - \sum_{i=0}^{J-1} \alpha_i^n \mathsf{U}^{n-i}}{\Delta t_n} = \mathcal{M}^{-1}\left(-\sum_{i=0}^{J-1} \beta_i^n C\left(\mathsf{U}^{n-i}\right) + \mathcal{F}(t_{n+1})\right), \quad (1)$$

$$\mathcal{L}\left(\mathsf{P}^{n+1}\right) = -\frac{\gamma_0^n}{\Delta t_n}\mathcal{D}\left(\hat{\mathsf{U}}\right), \quad (2)$$

$$\hat{\hat{\mathsf{U}}} = \hat{\mathsf{U}} - \frac{\Delta t_n}{\gamma_0^n}\mathcal{M}^{-1}\left(\mathcal{G}\left(\mathsf{P}^{n+1}\right)\right), \quad (3)$$

$$\left(\frac{\gamma_0^n}{\Delta t_n}\mathcal{M} + \mathcal{V}\right)\hat{\hat{\mathsf{U}}} = \frac{\gamma_0^n}{\Delta t_n}\mathcal{M}\left(\hat{\mathsf{U}}\right), \quad (4)$$

$$\left(\mathcal{M} + \mathcal{A}_{\text{pen}}\Delta t_n\right)\mathsf{U}^{n+1} = \mathcal{M}\left(\hat{\hat{\mathsf{U}}}\right). \quad (5)$$

The individual steps are the explicit convective step (1), the pressure step (2), the explicit projection step (3), the viscous step (4), and the penalty step (5). The imposition of boundary conditions is described in detail in [19, 23]. The incompressible nature of the equations is reflected in the Poisson problem (2) for the pressure, i.e., the pressure adapts instantaneously to changes in boundary conditions. From a computer science perspective, the pressure Poisson equation typically necessitates more complex communication patterns with global communication to reflect this behavior, in contrast with the nearest neighbor communication involved in explicit steps as well as simpler conjugate gradient solvers for the viscous and penalty steps, as discussed in more detail in Section 3.

The explicit formulation of the convective term avoids the solution of a non-linear algebraic system of equations, at the price of a restriction of the time step size according to the CFL condition

$$\Delta t = \min_{e=1,\ldots,N} \frac{\text{CFL}}{k^{1.5}} \left.\frac{h}{\|\boldsymbol{u}_h\|}\right|_e. \quad (6)$$

In our code, the ratio $\frac{h}{\|\boldsymbol{u}_h\|}$ is evaluated locally within each element whereas CFL and the polynomial degree k are kept constant throughout the mesh. The time step size is adjusted from one time step to the next in order to minimize the overall number of time steps by adapting Δt to the instantaneous local velocity fields in the most critical elements. This condition renders the time step size a dependent variable, reducing the parameter space $\{h, k, \Delta t\}$ of the discretization to the spatial parameters $\{h, k\}$.

3 INNOVATIONS REALIZED

3.1 Matrix-free operator evaluation

One of our main contributions is the development of algorithms that both minimize and balance the number of arithmetic operations and the memory access, in order to achieve optimal node-level performance on modern multi-core node architectures, as they are common on today's HPC systems with high Flop-to-Byte ratios.

The action of the discretized operators on vectors is implemented in a matrix-free fashion by mimicking the data-access behavior of finite difference stencils, generalized to geometrically flexible high-order discretization techniques. The element and surface integrals underlying the discontinuous Galerkin discretization are computed on the fly by Gaussian quadrature exploiting sum factorization techniques. This concept has originally been established by the spectral element community with relatively high degrees [16, 52]. Developments in the last decade, e.g., [11, 42, 43], combined with the more rapid increase of arithmetic performance compared to memory bandwidth, have established these matrix-free schemes as the fastest known method to evaluate the operator action for degrees $k \geq 2$ [45]. The matrix-free evaluation of, e.g., the symmetric interior penalty discretization of the Laplace operator $\mathcal{L}(\mathsf{P})$ on the left-hand side of Equation (2), is performed by a sum of contributions from all the cells and faces in the mesh,

$$\mathcal{L}(\mathsf{P}) = \sum_{e=1}^{N_{\text{cells}}} G_e^{\mathsf{T}} I_e^{\mathsf{T}} D_e I_e G_e \mathsf{P} + \sum_{f=1}^{N_{\text{faces}}} G_f^{\mathsf{T}} I_f^{\mathsf{T}} D_f I_f G_f \mathsf{P}. \quad (7)$$

In Equation (7), the gather operator G_e extracts the degrees-of-freedom local to element e from the global vector P and handles possible hanging-node constraints. Operator I_e computes the reference-coordinate gradient $\hat{\nabla} p_h$ of the interpolated solution associated with P at all $(k+1)^3$ quadrature points of element e. The effect of the element deformation is factored out from the gradient by the chain rule and contained in the operator D_e, and similarly for the metric terms of the test function as well as the quadrature weight and the determinant of the Jacobian. The gradient of the test functions and summation over the cell's quadrature points is represented by I_e^{T}, before the local results are scattered into the result vector by G_e^{T}. Likewise, the face integrals can be written as a sequence of a gather operator G_f (from two neighboring elements), interpolation of values and reference-coordinate gradients to quadrature points (I_f), a differential operator D_f acting independently at each quadrature point, integration with I_f^{T} and a subsequent scatter operator G_f^{T} adding local results into the global vector. Figure 2 visualizes the algorithmic steps of the sum factorization operations underlying the operators I_e and I_e^{T} as well as the operations at quadrature points D_e. While the former combine information from

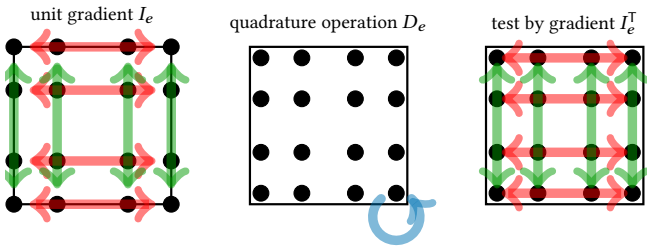

Figure 2: Visualization of interpolation, operation at quadrature points, and integration in the matrix-free evaluation (7) **for** $d = 2$ **and** $k = 3$**.**

all nodal values and quadrature points, the latter processes each point independently.

The interpolation operators I_e and I_f, which couple between all the unknowns within an element, are the same on each element, allowing us to keep a single instance of I_e or I_f in fast cache or register memory. Hence, only the solution vectors and the block-diagonal matrices D_e and D_f, with a block size of 3×3 collecting all components of the derivative in 3D, need to be loaded from slower main memory. Furthermore, we exploit the tensor-product structure within I_e and I_f in terms of the basis functions and quadrature points, in order to hide arithmetic operations behind the memory transfer. By contrast, non-tensor product elements such as tetrahedra involve an exceedingly high arithmetic cost [50] with reduced performance regarding the application-relevant throughput metric (Table 1).

Our highly-tuned kernels use Flop-minimizing optimizations in the sum factorization algorithms, such as change of basis and even-odd decomposition [43]. These optimizations yield a speedup of 1.5×–2× compared to previous results such as [38]. Cell and face integrals are computed alternately, in order to limit the access in the source vector P to a single access from main memory and serve the other accesses primarily from caches. The algorithms by [43] are used through the `deal.II` finite element library [3, 5], master branch from August 2021 (commit `b865ae5`).

3.2 MPI and SIMD parallelization

The matrix-free algorithm (7) is parallelized with pure MPI by partitioning the computational domain in the usual finite element setting [7, 12], running the loops only on the elements and faces associated with the elements owned by the local MPI process. The data exchange is between nearest neighbors in the mesh only using non-blocking communication operations. Communication is efficiently overlapped with computation to minimize communication overhead.

In addition to the coarse-grained parallelism, the code is tailored to efficiently use SIMD vectorization across CPU-type architectures (Intel, AMD or Fujitsu A64FX). It exploits the available parallelism across cells and faces. However, the use of SIMD is complicated by differences between architectures, typically requiring significant porting and tuning efforts, and the limited capabilities of modern compilers to automatically detect and implement such outer-loop parallelism. In order to overcome these problems and to enable performance-portable vectorization, we employ a cross-platform abstraction via C++ wrapper classes. The design principle is to provide overloads of the basic arithmetic operations `+,-,*,/` as well as broadcast, gather, scatter or struct-of-array to array-of-struct conversions with the relevant intrinsics for each platform in a single place. This approach, similar to the `std::simd` class scheduled for the upcoming C++23 standard [33], allows to transparently cover many hundreds of use cases with full-width SIMD throughout a large application code. All other aspects of machine code generation beyond intrinsics are still handled by an optimizing C++ compiler, which results in excellent performance for this application domain [43]. Using this abstraction, the operations shown in Figure 2 for the summations in Equation (7) are run for N_{SIMD} items at once. As an example, on the Intel Skylake architecture with its AVX-512 instruction set, we can use this mechanism to configure the execution to use 8 items in double-precision or 16 items in single-precision, fully exploiting the 512-bit vector registers. Porting to other architectures can simply be achieved by reconfiguring the matching class configuration to exploit the matching SIMD width on the target platform. Apart from array-of-struct to struct-of-array conversions during the gather and scatter stages, this abstraction also covers all relevant operations inside cells and ensures that they are fully vectorized without cross-lane traffic. As a result, more than 97% of the arithmetic work is run in vector registers, independent of the specific target architecture and compiler.

Cross-element vectorization increases granularity. Hence, there are now two possible limits to parallel scalability of the matrix-free operator evaluation: (i) the impact of communication latency to be overlapped with computations, and (ii) the domain decomposition granularity by cross-element SIMD, which increases for higher degrees, see also Fischer et al. [27]. On present hardware, SIMD granularity affects parallel scalability for $k \gtrsim 8$ for scalar equations and $k \gtrsim 6$ for vector-valued ones. For the range of moderately high polynomial degrees $k = 2, \ldots, 5$ targeted by the present work, communication latency is the dominating limit. In the following, we refer to a batch of 8 physical cells as a SIMD cell (based on double-precision (DP) arithmetic for AVX-512 registers) in order to characterize the limit of parallel scalability in terms of the number of SIMD batches of cells per MPI process. Since our mixed-precision solver uses single-precision (SP) for the multigrid V-cycle (Section 3.4), SIMD granularity suggests that 2 DP SIMD batches of cells, i.e., one SP SIMD batch of cells, are the limit.

3.3 Mesh generation and mapping

Figure 3 visualizes our hybrid patient-specific/idealized lung model of the conducting airway tree for various numbers of generations. By the example of $g = 5$, Figure 4 details the mesh-generation process to obtain a hex-only mesh, which is created by an application-specific mesh generator explicitly targeting the efficient 3D representation of patient-specific lung airways and minimizing I/O. The large airways starting at the trachea reaching down to generation three as well as the contours of the single lung lobes were directly segmented from CT images based on the voxel gray scale value in the image. The remainder of the bronchial tree has been constructed from a recursive tree growing algorithm mimicking adult morphology [57]. The algorithmic tree growth generates an

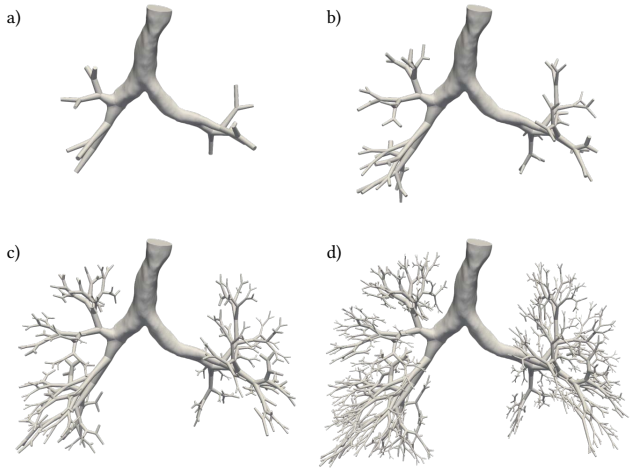

Figure 3: Visualization of lung model with different number of generations: a) $g = 5$, b) $g = 7$, c) $g = 9$, and d) $g = 11$ generations.

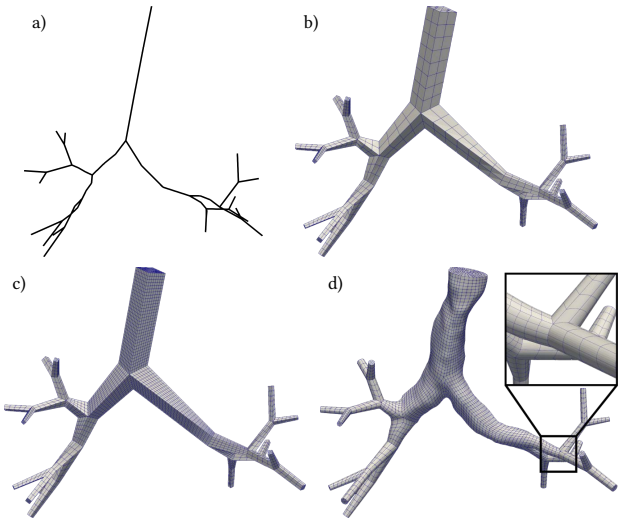

Figure 4: Visualization of lung mesh generation algorithm for $g = 5$ generations: (a) 1D tree, (b) 3D mesh consisting of square cylinders, (c) local mesh refinements of upper airways, (d) 3D deformed mesh according to mixed patient-specific/idealized geometry.

anatomically correct centerline starting at the third generation, the last fully segmentable airway generation. The next generation of daughter airways recursively bifurcate from their parents into the respective lung lobes regarding morphological length and diameter ratios reported for an adult (see [57] and references therein).

Our hex-only patient-specific mesh generator involves four steps: First, the centerlines of the segmented and of the recursively grown airways are changed into a single airway tree (see Figure 4 (a)) by introducing a centerline representation of the top generations, coinciding with the CT scan. Second, we transform centerline elements into cylinders with square cross section and anatomically correct dimensions (with 12 elements in the cross section) as well as with a suitable number of subdivisions in axial direction to ensure a high mesh quality with good cross-section to length ratios (see Figure 4 (b)). We merge the surface nodes of the independent cylinder meshes at special transition sections (see Figure 4 (b)). Third, we refine larger airways locally (see Figure 4 (c)) in order to balance the element sizes across the geometry. More importantly, the increased spatial resolution allows us to capture complex flow patterns characteristic for the upper airways during mechanical ventilation. To preserve high mesh quality, we use a non-conforming mesh with hanging nodes based on the forest-of-octree concept of the `p4est` library [7, 12]. Our algorithm also supports uniform refinement (of level l) in case higher resolution is needed everywhere. In terms of time step restrictions due to the CFL condition, we expect that low and intermediate generations are limiting, since the accumulated cross section area of all airways belonging to a certain generation increases significantly with the generation number. Fourth, we deform the top airway generations of the created 3D cylinder tree according to the CT scan (Figure 4 (d)). We use a ray-tracing algorithm by [49] to map the geometry onto a STL surface description directly segmented from CT scans, approximating the patient-specific geometry of each branch up to the third generation. Beyond this generation, an idealized cylindrical airway geometry is realized by a transfinite mapping [30] in radial direction. Note that the analytical geometry described by these concepts is approximated by a high-order polynomial description with auxiliary points in the interior of elements (along edges, surfaces, hexes), which we compute once during the simulation startup and then store for fast subsequent access as described by Heltai et al. [32]. The mesh is partitioned as implemented in the `deal.II` and `p4est` libraries [7, 12].

3.4 Scalable multigrid preconditioning

To solve the pressure Poisson equation, we use a *hybrid* multigrid solver as preconditioner for an iterative conjugate gradient solver. In our previous work [22], we have described a robust and efficient matrix-free multigrid realization for globally refined meshes and high-order discontinuous Galerkin methods. Algorithm 1 describes the main ingredients in a multigrid V-cycle, which is called recursively on coarser levels. The key aspect of this *hybrid* multigrid solver visualized in Figure 5 is that (i) a transfer to a continuous (auxiliary) space is performed to minimize iteration counts, and (ii) subsequent polynomial and geometrical coarsening is exploited to reduce the size of the coarse-grid problem to a minimum. In this setup, the two finest levels are the symmetric interior penalty DG discretization as well as the continuous FE discretization of the same degree and the same mesh. The throughput of these two ingredients in terms of memory transfer are the main drivers to the overall efficiency. The subsequent levels involve lower degrees (where we use a bisection from one level to the next) and coarser meshes, therefore contributing primarily to the communication latency and the strong-scaling properties.

For this work, we have extended the above-described algorithm to deal with locally refined meshes in a forest-of-octrees description. Specifically, new algorithms for the construction of geometric

Algorithm 1 MultigridVCycle(l, $\mathcal{A}^{(l)}$, $\mathsf{X}^{(l)}$, $\mathsf{B}^{(l)}$)

if $l = 0$ **then**
  $\mathsf{X}^{(0)} \leftarrow$ Coarse AMG solver ($\mathcal{A}^{(0)}$, $\mathsf{B}^{(0)}$)
else
  $\mathsf{X}^{(l)} \leftarrow$ Smooth (l, $\mathcal{A}^{(l)}$, $\mathsf{X}^{(l)}$, $\mathsf{B}^{(l)}$)
  $\mathsf{B}^{(l-1)} \leftarrow$ Restrict (l, $\mathsf{B}^{(l)} - \mathcal{A}^{(l)}\mathsf{X}^{(l)}$)
  $\mathsf{X}^{(l-1)} \leftarrow$ MultigridVCycle($l-1$, $\mathcal{A}^{(l-1)}$, 0, $\mathsf{B}^{(l-1)}$)
  $\mathsf{X}^{(l)} \leftarrow \mathsf{X}^{(l)}$+Prolongate($l$, $\mathsf{X}^{(l-1)}$)
  $\mathsf{X}^{(l)} \leftarrow$ Smooth (l, $\mathcal{A}^{(l)}$, $\mathsf{X}^{(l)}$, $\mathsf{B}^{(l)}$)
end if

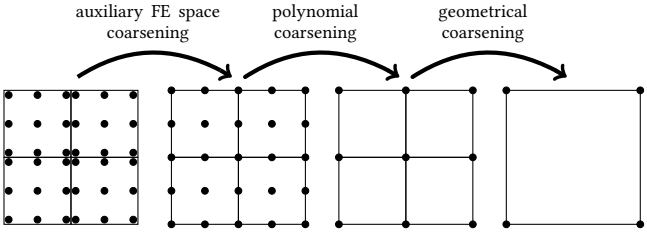

Figure 5: Visualization of transfer operations in our *hybrid* multigrid solver for the DG discretization.

levels via a procedure known as *global coarsening* [9, 56] have been added to the `deal.II` finite element library [4]: towards the next coarser level, each cell is coarsened if possible; the coarse grid is obtained once none of the remaining cells can be coarsened further. While this approach potentially involves more work on each level than local-smoothing algorithms [14], global-coarsening algorithms promise simpler load-balancing with better parallel efficiency and fewer iterations. A challenging task is to handle hanging-node constraints, which have to be considered during (i) the computation of the diagonal of the smoother, (ii) the setup of the transfer operators, and (iii) the application of the operators. However, the first two aspects are only relevant during the setup due to the static nature of our mesh. Regarding the third aspect, efficient implementations for the operator application on locally-refined meshes are available with at most 10% reduction in throughput due to the resolution of continuity requirements (continuous finite element levels) [42, 46] or sub-face interpolation (discontinuous Galerkin levels).

On each level, we implement the discrete operator (termed matrix-vector product), level transfer operations, and the smoother evaluations in a matrix-free fashion. In order to use fast matrix-vector products, we select a Chebyshev smoother with point Jacobi as preconditioner [2, 44, 45], using a polynomial degree of three with three matrix-vector products for pre- and postsmoothing. The V-cycle is run in single precision in order to improve the throughput of multigrid preconditioning (by doubling the number of cells per SIMD vector and per Byte of memory transferred from main memory). According to experiments in [44], this strategy does not affect multigrid convergence rates significantly. To increase the scalability and to limit the minimal work granularity on the coarser levels (at least 200 DoF/process), we distribute the coarser levels in the multigrid hierarchy only across subsets of processes and work on precomputed communicators similarly to Sundar et al. [56]. For the coarse-grid problem with several hundred thousands or more unknowns (depending on the number of lung generations), we use the algebraic multigrid solver `BoomerAMG` [18] on a linear continuous finite element space run in DP. All results are based on two V-cycles with a single sweep of symmetric Gauss–Seidel smoothing inside `BoomerAMG`, to comply with the smoother capability on the finer levels.

In order to inject the patient-specific and cylindrical geometry to the coarser geometric levels essential for optimal multigrid convergence, we use a consistent interpolation between the geometric levels in terms of the multigrid transfer operations.

Table 1: Application-motivated performance metrics.

target	metric [unit]
node-level performance	throughput [DoF/s]
strong scalability	minimal wall-time per time step [s]
lung application	minimal wall-time per simulated breathing cycle [h/cycle], alternatively: min. wall-time per liter of tidal volume [h/l]

4 EVALUATION SETUP AND METRICS

In order to evaluate our approach, in particular how it addresses the challenges in computational fluid dynamics discussed above, we define a set of application-motivated performance metrics summarized in Table 1. The limiting hardware resources of memory bandwidth and latency are reflected in the metrics of node-level performance and strong scalability, respectively. The lung application of simulating respiratory fluid mechanics defines a threshold regarding the allowable wall-time per breathing cycle. The challenge is then to maximize the resolution of the discretization scheme described by $\{g, h, k\}$ as a means to achieve optimal accuracy while respecting this wall-time limit. In the strong-scaling limit, the wall-time essentially depends on the overall number of time steps ($\min t_{\text{wall}} \sim N_{\Delta t} \sim T/\Delta t$), which increases for increasing spatial resolution due to equation (6). It is interesting to realize that the number of time steps per breathing cycle depends on the tidal volume V_{T} rather than the period T of one breathing cycle

$$\min t_{\text{wall}} \sim N_{\Delta t} \sim \frac{T}{\Delta t} \overset{(6)}{\sim} \frac{T\,U}{D} \sim \frac{T}{D}\frac{V_{\text{T}}/D^2}{T} \sim \frac{V_{\text{T}}}{D^3}\,, \tag{8}$$

where the mesh size h in the CFL condition is expressed in terms of a characteristic airway diameter D and the characteristic velocity U in terms of tidal volume V_{T}, diameter D, and period T. This leads to an alternative application-oriented performance metric, the minimal wall-time per liter of tidal volume. It has the advantage to allow performance comparisons between lung simulations with different ventilation strategies such as "normal" ventilation and high-frequency oscillatory ventilation (HFOV), which are characterized by substantially different tidal volumes. We note that floating point performance and memory bandwidth utilization are identified as secondary metrics that are not a direct target of our performance optimizations.

All experiments were performed on the Intel Skylake architecture of the SuperMUC-NG system,[3] consisting of 6480 nodes with 2×24 cores of Intel Xeon Platinum 8174, running at 2.3 GHz for all experiments presented here. Based on a node-level performance comparison of the Intel-19.0 compiler and the GNU compiler gcc-9.2, the GNU compiler with options `-O3 -funroll-loops -march=skylake-avx512` was selected. The slightly better performance is due to beneficial code generation for the SIMD abstraction described in Section 3.2. For MPI, the Intel implementation, version 2019, was employed. All experiments are based on a series of 20 repetitions, taking the best-performing sample. It has been ensured that, apart from disturbances of other jobs on the network of the machine, the deviation of the mean to the minimum is less than a few percent, which makes the reported performance data realistic. Dynamic mesh balancing has been used to account for the 0.5–1% of nodes clocked down due to thermal throttling.

5 PERFORMANCE RESULTS

In this section, we assess the performance for simulations of realistic scenarios of the human lung and answer research questions of relevance to the application field. As explained above, the main challenge is to identify a scalable pressure Poisson solver based on multigrid, whose performance in turn depends on the throughput of the smoother and level transfer. Therefore, our analysis starts with an investigation of the matrix-free operator evaluation and multigrid smoother, before going on to the pressure Poisson solver and eventually the full application code.

5.1 Evaluation of matrix-free algorithms

The main benefit of the proposed matrix-free algorithms is their high computational throughput, measured as the number of unknowns (degrees of freedom, DoF) that can be processed on given computational resources. The left panel of Figure 6 reports the achieved throughput of a single operator evaluation on a node of SuperMUC-NG for the DG Laplacian with polynomial degrees $k = 1, \dots, 6$. The throughput is computed from an experiment measuring the run time of 100 matrix-vector products on a mesh of the lung with $g = 11$ generations and 2m elements for $k = 1$ and 350k elements for $k \in \{2, \dots, 6\}$, resulting in problem sizes between 10 and 100 MDoF. For degree $k = 3$, we measure a throughput of 1.4×10^9 DoF/s, which is 25× higher than a sparse matrix-vector product could reach for the same discretization, or 3× higher than the throughput of a sparse matrix-vector product with linear finite elements.

In order to represent the application-relevant kernels, the left panel of Figure 6 also lists the throughput of one iteration in the Chebyshev smoother in the saturated regime, i.e., the granularity of one matrix-vector product and the associated vector updates [2, 22]. Due to the use of single precision, the achieved throughput is around 30% higher than the double-precision matrix-vector product alone. The gap to the theoretical 2× advantage of single precision shows the cost of vector updates and the high level of optimization in the matrix-free operator evaluation. Figure 6 (left) also highlights that both the discontinuous Galerkin operation on the finest multigrid level L as well as the continuous finite element discretization on the

[3] https://top500.org/system/179566/

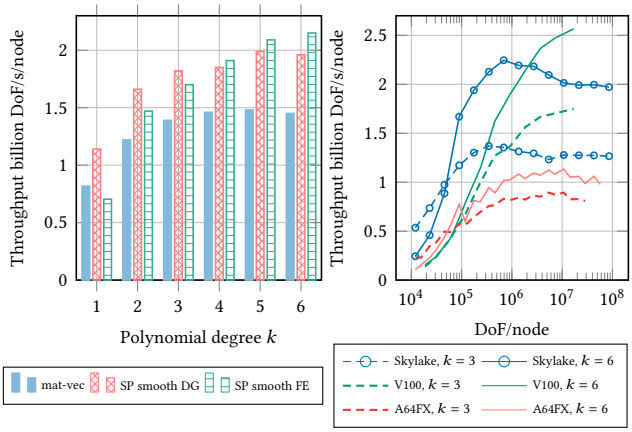

Figure 6: Left: Throughput of mat-vec product of the DG Laplacian on one Skylake node run in double precision (DP) and of one iteration of the Chebyshev smoother run in single precision (SP). The smoother is evaluated on the finest level with a symmetric interior penalty DG discretization and on the next coarser level with a continuous FE discretization. Right: Comparison of throughput per CG iteration of the CEED benchmark BP3 [27] on one Skylake node versus one V100 GPU on Summit [39] versus one A64FX node.

second-finest level $L - 1$ have been highly optimized with similar throughput. Since level $L-1$ runs with around 2.5× fewer unknowns for degree $k = 3$ in 3D, the run time in level $L - 1$ is actually more than 2× lower.

The right panel of Figure 6 lists the throughput of one conjugate gradient (CG) iteration for the CEED benchmark problem BP3 [27] with continuous finite elements of degrees $k = 3, 6$ for the present implementation based on the `deal.II` framework on the Intel Skylake architecture with 2×24 cores. The result is compared to state-of-the-art results from CEED in 2020 [39] on one Nvidia V100 GPU of Summit. Furthermore, the graph presents results of running our code on one node of the Fujitsu A64FX processor. Note that the throughput reported by CEED [40, Fig. 18] for a Fugaku node is slightly lower. Despite the lowest memory bandwidth of the three architectures (256 GByte/s nominal vs. 900 GByte/s for the other two architectures), the throughput on Skylake is highly competitive, a result of additional arithmetic and data access optimizations in our code [6]. More importantly, the SuperMUC-NG hardware reaches a much higher throughput for problem sizes of 10^4–10^6 than the competing systems (Number 1 and 2 of Top500 list). Even though the numbers are only reported for a single node here due to the availability of the other results, we can observe in practice that the faster performance for small sizes also improves the strong scaling limit for more nodes, a key performance metric in this work (see Table 1).

Figure 7 shows the achieved arithmetic throughput in the saturated regime (10–100 MDoF/node) for polynomial degrees $k = 1, \dots, 6$ as a function of the arithmetic intensity in a roofline plot [62]. Data is presented both for an idealized memory transfer model as well as the actually measured transfer from hardware performance

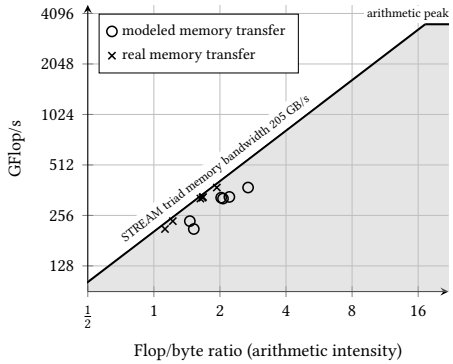

Figure 7: Node-level performance in terms of roofline model considering the evaluation of the DG Laplacian with polynomial degrees $k = 1, 2, \ldots, 6$ on the deformed lung geometry with $g = 11$ generations. Both the case of an ideal memory transfer (perfect caching, no transfer between neighboring MPI processes) and the actually measured memory transfer are shown.

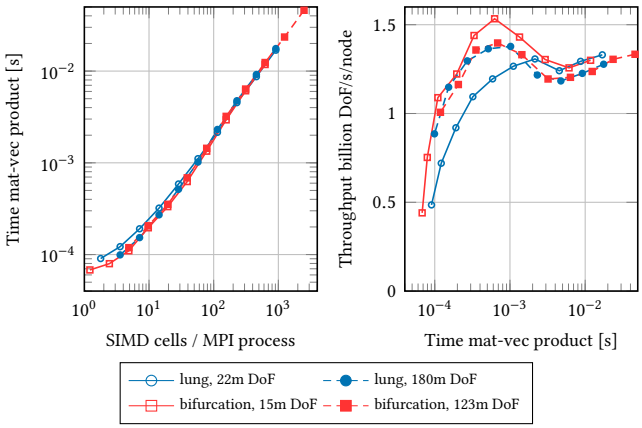

Figure 8: Throughput and latency analysis of matrix-free evaluation of the DG Laplacian with degree $k = 3$. For each of the four data sets shown, the number of compute nodes is increased from right to left, using up to 2048 nodes for the two bigger cases and 512 nodes for the two smaller cases. The lung test case uses adaptively refined meshes with hanging nodes and complicated transitions, whereas the bifurcation uses a uniformly refined mesh.

counters, evaluated using the LIKWID [58] tool. The number of arithmetic operations follows a slight modification of the data in Table 4 of [43] due to a different polynomial basis, and is confirmed to be accurate within a few percent by hardware performance counters. The roofline results show that the memory bandwidth is the performance limit in the regime of large problem sizes considered here. Data points of higher polynomial degrees appear further to the right in the plot due to a higher arithmetic intensity, but none of the interesting polynomial degrees is limited by the arithmetic throughput.

The underlying theoretical memory transfer model assumes a single transfer of each entry in the solution vectors from RAM memory, as well as access to the metric terms within D_e and D_f of Equation (7) plus the meta-data of a few integers per element to store the element-neighbor connectivity in terms of the vector indices and unknown numbering, following previous analysis [43]. All additional accesses are assumed to be served from fast cache memory, including the MPI data exchange between neighboring subdomains. The actual memory transfer is 20–30% higher, as in fact both the MPI communication and part of the neighbor access exceed caches for the present problem sizes. In Figure 7, this effect is visible by a lower arithmetic intensity of the measured memory transfer for the same GFlop/s value.

In the setting of the CFD simulations with many time steps, the ability to reach as small run times per time step as possible is reflected in the ability of strong scaling of the matrix-vector product. Figure 8 presents the scaling of both a complicated geometry in terms of the lung airway tree with 11 generations from Figure 3 as well as a simple bifurcation consisting of three cylinders. As opposed to classical strong scaling plots using the number of MPI processes on the abscissa, the left panel of the figure plots the run time over the work to be done per MPI process for four different work loads, in order to compare the scaling limit versus the work granularity described in Section 3.2. Passing along a line from right to left, the number of MPI processes is increased, which reduces the work per process and results in a reduction of run time, until the scaling saturates at slightly below 10^{-4} seconds.

In the right panel of Figure 8, the computational throughput of the same experiment is plotted over the resulting run time. A horizontal line would indicate ideal strong scaling. The actually observed double-bump like shape can be explained as follows. Starting from small node counts on the right, the throughput initially goes down as communication between more nodes is involved. As the parallelism further increases and run times go below 10^{-3} seconds, throughput increases due to a cache effect: for small workloads, the relevant data mostly or entirely fits into the 1 + 1.375 MB of L2+L3 cache of an Intel Xeon core. Eventually, around 10^{-4} seconds the communication latency between the compute nodes becomes dominating. This reduces the throughput below 30% of the saturated throughput. The fact that throughput drops sharply before the SIMD granularity of one SIMD cell is reached indicates that the granularity of vectorization discussed in Section 3.2 is not limiting. The results in Figure 8 are encouraging because the throughput of matrix-free evaluation on the complicated lung geometry reaches similar throughput as the matrix-vector product on a volumetric mesh. Only near the scaling limit and for the smaller case with 22 million DoF, the unstructured coarse mesh and the difficult problem of partitioning a partly adapted mesh with many trees lead to a higher communication cost and somewhat lower throughput.

5.2 Evaluation of multigrid performance

In the multigrid solver, the above ingredients are combined on a sequence of coarser meshes in a multiplicative fashion. Hence, the scaling is not only determined by the throughput and scaling limit of the matrix-free evaluation inside the smoother on the fine

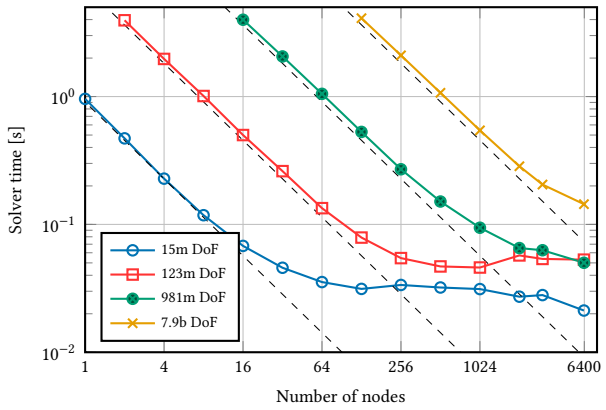

Figure 9: Combined strong/weak scaling study of Poisson solver for a generic bifurcation with refine levels $l = 3, 4, 5, 6$, polynomial degree $k = 3$, tolerance 10^{-10} (converges in 9 CG iterations). Ideal strong scaling is indicated by dashed lines.

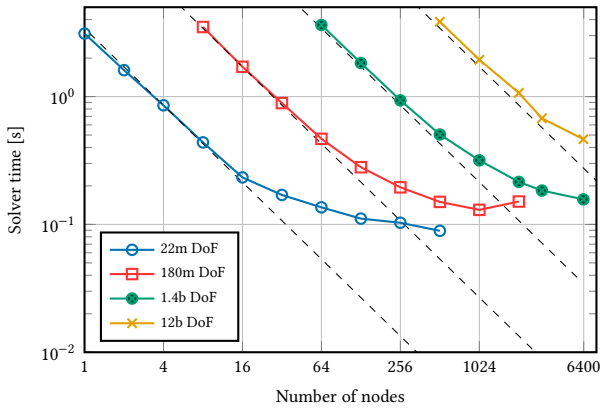

Figure 10: Combined strong/weak scaling study of Poisson solver for the lung geometry with $g = 11$, refine levels $l = 0, 1, 2, 3$, polynomial degree $k = 3$, tolerance 10^{-10} (converges in 21 CG iterations). Ideal strong scaling is indicated by dashed lines.

mesh assessed before, but also by the raw latency in the work on the coarser levels. Besides "horizontal" nearest-neighbor communication in the smoother, the multigrid Algorithm 1 includes also "vertical" communication between both the geometric and algebraic multigrid levels in terms of restriction, the coarse solver (a serial operation), and the subsequent prolongation, a communication pattern akin to a tree-based global reduction followed by a broadcast.

As a first test, we solve the Poisson equation on a volumetric geometry representing a bifurcation of a single cylinder into two outlets. The opening degree is 60 degrees, with element shapes of similar lengths in the radial and axial directions of the cylinders. On the circumferential surfaces, we set Neumann boundary conditions, whereas we use Dirichlet conditions on the radial in- and outlets, mimicking the case of the flow simulations on the lung. The initial mesh consists of 468 elements and is then uniformly refined. For a polynomial degree $k = 3$, the problem sizes are between 15 million and 7.9 billion DoF. Next to cube-liked geometries considered in [6, 22, 45], this is a best-case scenario for multigrid in terms of complexity as well as communication via the space-filling Morton curve underlying the domain decomposition via the `p4est` library [12]. The multigrid V-cycle is run in single precision as a preconditioner inside a conjugate gradient solver with a termination criterion of 10^{-10} (a solver tolerance used commonly in multigrid analysis [22]), measured as the norm of the unpreconditioned residual relative to the size of the right hand side.[4]

Figure 9 shows the strong and weak scaling of the solver on up to 6,400 nodes of SuperMUC-NG. The strong scaling is represented along each line and almost ideal down to run times of 0.1 seconds. From one line to the next, the problem size increases by a factor of eight. To facilitate interpretation also of weak scaling, the dashed lines indicating ideal strong scaling are a factor of eight apart. It can be seen that weak scaling is optimal as well, which is the result of optimal $O(n)$ complexity of the solver with 9 iterations for all problem sizes as well as optimal communication.

Next, we perform a similar experiment on the geometry of the lung, represented by $g = 11$ generations on the mesh from Figure 3-4 (considering additional global refinements of level l). Figure 10 reports the strong and weak scaling. In contrast to the bifurcation, the scaling saturates at a higher wall-time. The smallest case with 22 million DoF cannot scale below 0.1 seconds per solve, whereas 179 million DoF scale only to around 0.15 seconds per solve, with a loss of efficiency already before. This behavior can be explained by three main factors:

- The iteration count of the CG solver increases from 9 in the bifurcation case to 21–22 in the lung case; an effect of the chosen smoother whose effectivity decreases on the more strongly deformed elements in the patient-specific part of the lung, difficult angles in the airway network, as well as more anisotropy in the axial to radial element lengths.
- The mesh with hanging nodes and complicated bifurcations leads to a larger proportion of faces with differing orientation in coordinate systems of the two adjacent elements, which infers additional costs caused by only partially filled SIMD lanes for face integrals. This overhead is around 25% of the work on faces for the case with 180m DoF on 512 nodes (24k MPI processes). Furthermore, the adaptivity leads to a higher number of multigrid levels overall and thus a higher proportion of latency-limited operations.
- Due to the complicated coarse mesh, `BoomerAMG` employed for the coarse-level solve of the hybrid multigrid scheme becomes noticeable. It is distributed on 710 MPI processes of 15 nodes in the present experiments, taking around 3.5×10^{-3} seconds per call (exemplarily measured for the case of 180m DoF on 1024 nodes). Since `BoomerAMG` is invoked 21 times per solve, this already contributes 0.07 seconds of latency.

In terms of the breakdown of the latency accumulated within the multigrid hierarchy, the multigrid V-cycle of the 180m DoF case on

[4] Assuming a constant multigrid convergence rate ρ, iteration counts n for different solver tolerances can simply be scaled according to the number of digits of residual reduction, $n = \log_{10}(\|r_n\|_2/\|r_0\|_2)/\log_{10}\rho$, where $\|r_n\|_2/\|r_0\|_2 = 10^{-10}$ in the present work, with similar effects on the achievable run time.

1024 nodes (49k MPI processes) spends 18% of time on the finest level, 13% on the next finest level, 26% on all intermediate levels, and 45% in the AMG coarse solver. For comparison, for the same size on 64 nodes, the two finest levels contribute with 48% and 22%, respectively.

Overall, the scaling results and absolute run times are encouraging for the CFD application.

5.3 Application runs

We simulate the flow of air (density $\rho = 1.2$ kg/m^3, kinematic viscosity $\nu = 1.7 \cdot 10^{-5}$m^2/s) through the conducting airways of a tracheally intubated patient under realistic conditions of mechanical ventilation. We use physiologically sound, pressure-based boundary conditions both at the trachea inlet and the terminal airways: To mimic the behavior of the mechanical ventilator in our computational lung model, a pressure of PEEP + Δp is provided at the tracheal inlet during inhalation and PEEP during exhalation, with the positive end-expiratory pressure (PEEP) being 8 cmH_2O. The breathing period is $T = 3$ s with an inhalation-to-exhalation time ratio of 1 : 2. To mimic the behavior of mechanical ventilation applied in a clinical setting, we implemented a discrete controller that dynamically adjusts the pressure Δp from one breathing cycle to the next in order to reach the desired tidal volume of $V_T = 500$ ml. The present work simulates only the first breathing cycle. The pressure drop over the tubus (from the mechanical ventilator to the trachea of the patient) is regarded according to [31]. The pressure boundary conditions at the terminal airways are governed by appended linear single-compartment models according to [8] to consider resistive and compliant effects of the remaining, non-resolved airways and tissue components below the outlets. For each of those models, the resistance of the remaining airway tree (from generation g to 25) is calculated analytically, exploiting the assumption of laminar Poiseuille flow and using the diameter and length dimensions specified in [48], and combined with an additional tissue resistance (modelled as 20 % [61] of the total resistance of 0.15 kPa s/l [53]). The compliance for each outlet is deduced from the overall compliance $C = 100$ ml/cmH_2O [53], uniformly distributed over all terminal airways. We use a polynomial degree of $k = 3$ and a relative solver tolerances of 10^{-3} for the application runs. The coarse tolerances are enabled by extrapolations to start with accurate initial guesses from previous time steps [24, 41]. As noted above, the achievable wall time limits are therefore approximately 3× lower than with the strict 10^{-10} tolerance of Figure 10. The CFL number is CFL = 0.4. Further, the multigrid preconditioner described previously is used for the pressure Poisson equation, and the other sub-steps of the splitting scheme are efficiently preconditioned by the inverse mass operator.

Table 2 lists the performance of lung application runs in terms of the minimal wall-time per time step (averaged over all time steps) and the wall-time per breathing cycle (or per liter of tidal volume) vs. the number of resolved generations (see Figure 3 for a visualization). We reach wall-times as low as $0.017 - 0.045$ s per time step for $g = 3 - 11$ generations. A comparison to state-of-the-art results shown in Table 3 reveals that the present solver is highly competitive. While the present results approach the strong-scaling performance shown recently by Krank et al. [41] and Arndt et al. [6]

Table 2: Performance of lung application runs (simulating the first breathing cycle): The global refinement level is $l = 0$ in all cases and the number of nodes is chosen such that each simulation runs in the strong-scaling limit (with number of SIMD cells per core between 2 and 8).

g	#node	#cell	#DoF	$N_{\Delta t}$	$t_{\text{wall}}/N_{\Delta t}$	h/cycle	h/l
3	2	2.0e3	4.4e5	1.8e5	0.0174 s	0.9	1.9
5	16	1.8e4	3.6e6	5.2e5	0.0232 s	3.4	7.3
7	32	4.2e4	9.2e6	1.0e6	0.0229 s	6.4	14
9	128	2.1e5	4.5e7	1.6e6	0.0419 s	19	43
11	128	3.5e5	7.7e7	2.0e6	0.0451 s	25	57

Table 3: Minimum wall-time per time step of state-of-the-art high-order incompressible flow solvers for large-scale simulations operating in the strong-scaling limit. SB: Sandy Bridge, Sky: Skylake.

publication	supercomputer	min. $t_{\text{wall}}/N_{\Delta t}$
[51, Figure 3]	Mira (Power BQC)	0.1 s
[39, Table 2]	Summit (Nvidia V100)	$0.066 - 0.1$ s
[40, Tables 3–4]	Fugaku (Fujitsu A64FX)	$0.1 - 0.2$ s
[41, Figure 7]	SuperMUC (Intel SB)	0.05 s
[6, Figure 13]	SuperMUC-NG (Intel Sky)	$0.015 - 0.03$ s

for turbulence simulations on a Cartesian geometry with trivial coarse grid problem, they outperform results published by Offermans et al. [51] and the CEED center [39, 40] for the state-of-the-art spectral element incompressible flow solver `Nek5000`/`NekRS` [28]. The CEED milestone reports [39, 40] specify a minimal wall-time per time step of approximately 0.1 s on the GPU-based supercomputer Summit as well as the ARM-based supercomputer Fugaku (currently Number 2 and Number 1 of Top500 list). Note that we achieve significantly lower wall-times per time step for a highly complex geometry with adaptively refined meshes, confirming the results of Figure 6. Towards larger problem sizes, we expect a slow-down of the strong-scaling limit by a factor of around 2 according to Figures 9 and 10. Then, our results would still be within the threshold of 0.1 s, so that we consider the present results substantially ahead of the current state-of-the-art.[5]

6 IMPLICATIONS

Our results reveal that different hardware bottlenecks are reached for different regimes of the incompressible flow solver. The memory bandwidth limits the problem size in the saturated regime in order to achieve reasonable wall-times on a per-time-step basis. The latency

[5]Note that for example the Gordon Bell 2020 finalist paper [37] dealing with incompressible flow solvers using low-order finite elements did not quantify performance in terms of this scaling limit, which is most important for practical turbulence simulations [47].

associated to the multigrid communication limits the minimal wall-time in the strong-scaling limit particularly relevant for unsteady problems. Among these bottlenecks, the latency-related strong-scaling limit appears to be most pressing for our lung application problem, for which millions of time steps need to be performed. The maximum number of generations and the maximum mesh resolution is currently limited by this scaling limit when striving for "overnight" runs.

However, our findings are also encouraging, in reaching or exceeding the state-of-the-art for complex-geometry cases simulated with high-order CFD methods in terms of minimizing wall time. We expect that further tuning of the multigrid coarsening and interaction with the algebraic multigrid coarse solver can deliver additional improvements near the scaling limit, by limiting the number of additional levels that add latency as well as by reducing point-to-point communication by a holistic partitioning approach of all multigrid levels.

The beneficial behavior demonstrated for a CPU-only supercomputer point to opportunities for future hybrid CPU/GPU systems: While a GPU provides more compute units and a higher memory bandwidth for increasing throughput on the finest multigrid levels, CPUs with their latency focus are favorable for the coarser levels. While additional effort would be needed to integrate GPU capabilities with optimal throughput, e.g. by frameworks like libCEED [1, 40], and to carefully coordinate between both architectures, further enhancements for a broader spectrum of problem sizes can be envisioned in the future.

7 CONCLUSIONS

Pulmonary diseases are rated among the major causes of global morbidity and mortality [26] and have in recent times even reached a higher level of urgency due to COVID-19. Computational methods play a key role in understanding the impact of such diseases as well as the potential damages that can be caused by treatments like mechanical ventilation. This creates enormous computational needs for an already highly compute intensive problem.

In this work we presented extensions to ExaDG [21], a high-performance turbulent flow solver. In this work, we focus on matrix-free algorithms enabling a better compute-to-memory-access ratio and introduced an C++ abstraction layer that enabled us to easily exploit SIMD vectorization in a platform agnostic manner. We further integrated the work into a highly scalable, MPI-based framework with efficient neighbor communication and provided a thorough evaluation on the SuperMUC-NG system, one of the largest CPU-only platforms listed on the Top500 list. Our experiments demonstrated a run time of around or below 0.1 seconds per time step for geometrically complex large-scale simulations, which is a significant step forward compared to the state-of-the-art and provides the basis for future usage scenarios of personalized simulations of the human respiratory system.

ACKNOWLEDGMENTS

This work was supported by the German Research Foundation (DFG) via the project "High-order discontinuous Galerkin for the exa-scale" (ExaDG) within the priority program 1648 "Software for Exascale Computing" (SPPEXA, www.sppexa.de). The authors acknowledge the support and hardware access through the Bayerische Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen (KONWIHR) in the framework of the project "Performance tuning of high-order discontinuous Galerkin solvers for SuperMUC-NG". The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC-NG at Leibniz Supercomputing Centre (LRZ, www.lrz.de) through project id pr83te. The authors thank Timo Heister, Laura Prieto Saavedra, and Marc Fehling for fruitful discussions and their support in the context of global-coarsening multigrid algorithms.

REFERENCES

[1] A. Abdelfattah, V. Barra, N. Beams, J. Brown, J.-S. Camier, V. Dobrev, Y. Dudouit, L. Ghaffari, T. Kolev, D. Medina, T. Rathnayake, J. L. Thompson, and S. Tomov. 2020. libCEED user manual. https://doi.org/10.5281/zenodo.4302737

[2] M. Adams, M. Brezina, J. Hu, and R. Tuminaro. 2003. Parallel multigrid smoothing: polynomial versus Gauss–Seidel. *J. Comput. Phys.* 188 (2003), 593–610. https://doi.org/10.1016/S0021-9991(03)00194-3

[3] D. Arndt, W. Bangerth, B. Blais, T. C. Clevenger, M. Fehling, A. V. Grayver, T. Heister, L. Heltai, M. Kronbichler, M. Maier, P. Munch, J.-P. Pelteret, R. Rastak, I. Tomas, B. Turcksin, Z. Wang, and D. Wells. 2020. The deal.II library, version 9.2. *J. Numer. Math.* 28, 3 (2020), 131–146. https://doi.org/10.1515/jnma-2020-0043

[4] D. Arndt, W. Bangerth, B. Blais, M. Fehling, R. Gassmöller, T. Heister, L. Heltai, U. Köcher, M. Kronbichler, M. Maier, P. Munch, J.-P. Pelteret, S. Proell, K. Simon, B. Turcksin, D. Wells, and J. Zhang. 2021. The deal.II library, version 9.3. *J. Numer. Math.* 29, 3 (2021). https://doi.org/10.1515/jnma-2021-0081

[5] D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells. 2021. The deal.II finite element library: design, features, and insights. *Comput. Math. Appl.* 81 (2021), 407–422. https://doi.org/10.1016/j.camwa.2020.02.022

[6] D. Arndt, N. Fehn, G. Kanschat, K. Kormann, M. Kronbichler, P. Munch, W. A. Wall, and J. Witte. 2020. ExaDG – high-order discontinuous Galerkin for the exa-scale. In *Software for Exascale Computing – SPPEXA 2016–2019 (Lecture Notes in Computational Science and Engineering 136)*, H.-J. Bungartz, S. Reiz, B. Uekermann, P. Neumann, and W. E. Nagel (Eds.). Springer International Publishing, Cham, 189–224. https://doi.org/10.1007/978-3-030-47956-5_8

[7] W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler. 2011. Algorithms and data structures for massively parallel generic finite element codes. *ACM Trans. Math. Softw.* 38, 2 (2011), 14:1–28. https://doi.org/10.1145/2049673.2049678

[8] J. Bates. 2009. *Lung Mechanics: An Inverse Modeling Approach.* Cambridge University Press, Cambridge, UK New York.

[9] R. Becker and M. Braack. 2000. Multigrid techniques for finite elements on locally refined meshes. *Numer. Lin. Algebr. Appl.* 7, 6 (2000), 363–379. https://doi.org/10.1002/1099-1506(200009)7:6<363::aid-nla202>3.0.co;2-v

[10] R. G. Brower, M. A. Matthay, A. Morris, D. Schoenfeld, B. T. Thompson, A. Wheeler, H. P. Wiedemann, A. C. Arroliga, C. J. Fisher, J. J. Komara, et al. 2000. Ventilation with lower tidal volumes as compared with traditional tidal volumes for acute lung injury and the acute respiratory distress syndrome. *New Engl. J. Med.* 342, 18 (2000), 1301–1308.

[11] J. Brown. 2010. Efficient nonlinear solvers for nodal high-order finite elements in 3D. *J. Sci. Comput.* 45, 1-3 (2010), 48–63. https://doi.org/10.1007/s10915-010-9396-8

[12] C. Burstedde, L. C. Wilcox, and O. Ghattas. 2011. p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM J. Sci. Comput.* 33, 3 (2011), 1103–1133. https://doi.org/10.1137/10079163

[13] J. Choi, G. Xia, M. H. Tawhai, E. A. Hoffman, and C.-L. Lin. 2010. Numerical study of high-frequency oscillatory air flow and convective mixing in a CT-based human airway model. *Ann. Biomed. Eng.* 38, 12 (2010), 3550–3571.

[14] T. C. Clevenger, T. Heister, G. Kanschat, and M. Kronbichler. 2021. A flexible, parallel, adaptive geometric multigrid method for FEM. *ACM Trans. Math. Softw.* 47, 1 (2021), 7:1–27. https://doi.org/10.1145/3425193

[15] J. W. De Backer, W. G. Vos, S. C. Vinchurkar, R. Claes, A. Drollmann, D. Wulfrank, P. M. Parizel, P. Germonpré, and W. De Backer. 2010. Validation of computational fluid dynamics in CT-based airway models with SPECT/CT. *Radiology* 257, 3 (2010), 854–862.

[16] M. O. Deville, P. F. Fischer, and E. H. Mund. 2002. *High-order methods for incompressible fluid flow.* Vol. 9. Cambridge University Press.

[17] A. Esteban, A. Anzueto, F. Frutos, I. Alía, L. Brochard, T. E. Stewart, S. Benito, S. K. Epstein, C. Apezteguía, P. Nightingale, et al. 2002. Characteristics and outcomes

in adult patients receiving mechanical ventilation: a 28-day international study. *JAMA* 287, 3 (2002), 345–355.

[18] R. D. Falgout, J. E. Jones, and U. M. Yang. [n.d.]. The design and implementation of hypre, a library of parallel high performance preconditioners. In *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, 267–294. https://doi.org/10.1007/3-540-31619-1_8

[19] N. Fehn, J. Heinz, W. A. Wall, and M. Kronbichler. 2021. High-order arbitrary Lagrangian–Eulerian discontinuous Galerkin methods for the incompressible Navier–Stokes equations. *J. Comput. Phys.* 430 (2021), 110040. https://doi.org/10.1016/j.jcp.2020.110040

[20] N. Fehn, M. Kronbichler, C. Lehrenfeld, G. Lube, and P. W. Schroeder. 2019. High-order DG solvers for under-resolved turbulent incompressible flows: A comparison of L^2 and $H(\mathrm{div})$ methods. *Int. J. Numer. Meth. Fluids* 91, 11 (2019), 533–556. https://doi.org/10.1002/fld.4763

[21] N. Fehn, M. Kronbichler, P. Munch, and M. Bergbauer. 2021. *ExaDG: High-Order Discontinuous Galerkin for the Exa-Scale*. https://doi.org/10.5281/zenodo.5176507 Code available on https://github.com/exadg/exadg.

[22] N. Fehn, P. Munch, W. A. Wall, and M. Kronbichler. 2020. Hybrid multigrid methods for high-order discontinuous Galerkin discretizations. *J. Comput. Phys.* 415 (2020), 109538. https://doi.org/10.1016/j.jcp.2020.109538

[23] N. Fehn, W. A. Wall, and M. Kronbichler. 2017. On the stability of projection methods for the incompressible Navier–Stokes equations based on high-order discontinuous Galerkin discretizations. *J. Comput. Phys.* 351 (2017), 392–421. https://doi.org/10.1016/j.jcp.2017.09.031

[24] N. Fehn, W. A. Wall, and M. Kronbichler. 2018. Efficiency of high-performance discontinuous Galerkin spectral element methods for under-resolved turbulent incompressible flows. *Int. J. Numer. Meth. Fluids* 88, 1 (2018), 32–54. https://doi.org/10.1002/fld.4511

[25] N. Fehn, W. A. Wall, and M. Kronbichler. 2018. Robust and efficient discontinuous Galerkin methods for under-resolved turbulent incompressible flows. *J. Comput. Phys.* 372 (2018), 667–693. https://doi.org/10.1016/j.jcp.2018.06.037

[26] T. Ferkol and D. Schraufnagel. 2014. The global burden of respiratory disease. *Ann. Amer. Thorac. Soc.* 11, 3 (2014), 404–406.

[27] P. Fischer, M. Min, T. Rathnayake, S. Dutta, T. Kolev, V. Dobrev, J.-S. Camier, M. Kronbichler, T. Warburton, K. Świrydowicz, and J. Brown. 2020. Scalability of high-performance PDE solvers. *Int. J. High Perf. Comput. Appl.* 34, 5 (2020), 562–586. https://doi.org/10.1177/1094342020915762

[28] P. F. Fischer, S. Kerkemeier, et al. 2020. Nek5000 Web page. https://nek5000.mcs.anl.gov.

[29] T. Gemci, V. Ponyavin, Y. Chen, H. Chen, and R. Collins. 2008. Computational model of airflow in upper 17 generations of human respiratory tract. *J. Biomech.* 41, 9 (2008), 2047–2054.

[30] W. J. Gordon and L. C. Thiel. 1982. Transfinite mappings and their application to grid generation. *Appl. Math. Comput.* 10 (1982), 171–233. https://doi.org/10.1016/0096-3003(82)90191-6

[31] J. Guttmann, L. Eberhard, B. Fabry, W. Bertschmann, and G. Wolff. 1993. Continuous calculation of intratracheal pressure in tracheally intubated patients. *Anesthesiology* 79, 3 (1993), 503–513.

[32] L. Heltai, W. Bangerth, M. Kronbichler, and A. Mola. 2021. Propagating geometry information to finite element computations. *ACM Trans. Math. Softw.* 47, 3 (2021), in press. https://doi.org/10.1145/3468428

[33] J. Hoberock. 2019. *Working Draft, C ++ Extensions for Parallelism Version 2*. Technical Report.

[34] K. Horsfield and G. Cumming. 1968. Morphology of the bronchial tree in man. *Journal of Applied Physiology* 24, 3 (1968), 373–383.

[35] S. Kabilan, C.-L. Lin, and E. A. Hoffman. 2007. Characteristics of airflow in a CT-based ovine lung: a numerical study. *J. Appl. Physiol.* 102, 4 (2007), 1469–1482.

[36] G. E. Karniadakis, M. Israeli, and S. A. Orszag. 1991. High-order splitting methods for the incompressible Navier–Stokes equations. *J. Comput. Phys.* 97, 2 (1991), 414–443. https://doi.org/10.1016/0021-9991(91)90007-8

[37] C. Kato, Y. Yamade, K. Nagano, K. Kumahata, K. Minami, and T. Nishikawa. 2020. Toward realization of numerical towing-tank tests by wall-resolved large eddy simulation based on 32 billion grid finite-element computation. In *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 23–35.

[38] D. Kempf, R. Heß, S. Müthing, and P. Bastian. 2021. Automatic code generation for high-performance discontinuous Galerkin methods on modern architectures. *ACM Trans. Math. Softw.* 47, 1 (2021), 6/1–31. https://doi.org/10.1145/3424144

[39] T. Kolev et al. 2020. *Support CEED-enabled ECP applications in their preparation for Aurora/Frontier*. Technical Report ECP Milestone CEED-MS35. US Department of Energy.

[40] T. Kolev et al. 2021. *High-order algorithmic developments and optimizations for large-scale GPU-accelerated simulations*. Technical Report ECP Milestone CEED-MS36. US Department of Energy.

[41] B. Krank, N. Fehn, W. A. Wall, and M. Kronbichler. 2017. A high-order semi-explicit discontinuous Galerkin solver for 3D incompressible flow with application to DNS and LES of turbulent channel flow. *J. Comput. Phys.* 348 (2017), 634–659. https://doi.org/10.1016/j.jcp.2017.07.039

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

[43] M. Kronbichler and K. Kormann. 2019. Fast matrix-free evaluation of discontinuous Galerkin finite element operators. *ACM Trans. Math. Softw.* 45, 3 (2019), 29:1–40. https://doi.org/10.1145/3325864

[44] M. Kronbichler and K. Ljungkvist. 2019. Multigrid for matrix-free high-order finite element computations on graphics processors. *ACM Trans. Parallel Comput.* 6, 1 (2019), 2:1–32. https://doi.org/10.1145/3322813

[45] M. Kronbichler and W. A. Wall. 2018. A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers. *SIAM J. Sci. Comput.* 40, 5 (2018), A3423–A3448. https://doi.org/10.1137/16M110455X

[46] K. Ljungkvist. 2017. Matrix-free Finite-element Computations on Graphics Processors with Adaptively Refined Unstructured Meshes. In *Proceedings of the 25th High Performance Computing Symposium* (Virginia Beach, Virginia) *(HPC '17)*. Society for Computer Simulation International, San Diego, CA, USA, Article 1, 12 pages. http://dl.acm.org/citation.cfm?id=3108096.3108097

[47] R. Löhner. 2019. Towards overcoming the LES crisis. *Int. J. Comput. Fluid Dyn.* 33, 3 (2019), 87–97. https://doi.org/10.1080/10618562.2019.1612052

[48] M. G. Ménache, W. Hofmann, B. Ashgarian, and F. J. Miller. 2008. Airway geometry models of children's lungs for use in dosimetry modeling. *Inhalation toxicology* 20, 2 (2008), 101–126.

[49] T. Möller and B. Trumbore. 1997. Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools* 2, 1 (1997), 21–28. https://doi.org/10.1080/10867651.1997.10487468 arXiv:https://doi.org/10.1080/10867651.1997.10487468

[50] D. Moxey, R. Amici, and M. Kirby. 2020. Efficient matrix-free high-order finite element evaluation for simplicial elements. *SIAM J. Sci. Comput.* 42, 3 (2020), C97–C123. https://doi.org/10.1137/19m1246523

[51] N. Offermans, O. Marin, M. Schanen, J. Gong, P. Fischer, P. Schlatter, A. Obabko, A. Peplinski, M. Hutchinson, and E. Merzari. 2016. On the strong scaling of the spectral element solver Nek5000 on petascale systems. In *Proceedings of the Exascale Applications and Software Conference 2016* (Stockholm, Sweden) *(EASC '16)*. ACM, New York, NY, USA, Article 5, 10 pages. https://doi.org/10.1145/2938615.2938617

[52] S. A. Orszag. 1980. Spectral methods for problems in complex geometries. *J. Comput. Phys.* 37 (1980), 70–92.

[53] H.-C. Pape, A. Kurtz, and S. Silbernagl. 2018. *Physiologie*. Georg Thieme Verlag, Stuttgart New York.

[54] S. Qi, B. Zhang, Y. Teng, J. Li, Y. Yue, Y. Kang, and W. Qian. 2017. Transient dynamics simulation of airflow in a CT-scanned human airway tree: more or fewer terminal bronchi? *Comput. Math. Method Med.* 2017 (2017).

[55] C. J. Roth, K. M. Förster, A. Hilgendorff, B. Ertl-Wagner, W. A. Wall, and A. W. Flemmer. 2018. Gas exchange mechanisms in preterm infants on HFOV–a computational approach. *Scientific reports* 8, 1 (2018), 1–8.

[56] H. Sundar, G. Biros, C. Burstedde, J. Rudi, O. Ghattas, and G. Stadler. 2012. Parallel geometric-algebraic multigrid on unstructured forests of octrees. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–11.

[57] M. Tawhai, A. J. Pullan, and P. Hunter. 2000. Generation of an anatomically based three-dimensional model of the conducting airways. *Annals of biomedical engineering* 28 (08 2000), 793–802. https://doi.org/10.1114/1.1289457

[58] J. Treibig, G. Hager, and G. Wellein. 2010. 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. https://doi.org/10.1109/ICPPW.2010.38 `https://github.com/RRZE-HPC/likwid`.

[59] D. K. Walters, G. W. Burgreen, D. M. Lavallee, D. S. Thompson, and R. L. Hester. 2011. Efficient, physiologically realistic lung airflow simulations. *IEEE T. Biomed. Eng.* 58, 10 (2011), 3016–3019.

[60] E. R. Weibel, A. F. Cournand, and D. W. Richards. 1963. *Morphometry of the human lung*. Vol. 1. Springer.

[61] J. B. West and A. M. Luks. 2016. *West's Respiratory Physiology: The Essentials*. Wolters Kluwer, Philadelphia.

[62] S. Williams, A. Waterman, and D. Patterson. 2009. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76. https://doi.org/10.1145/1498765.1498785

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

All experiments were performed on the SuperMUC-NG system with Intel Skylake architecture on up to 3072 nodes with 2×24 cores of Intel Xeon Platinum 8174, running at a fixed frequency of 2.3 GHz. The GNU compiler gcc-9.2 was used with options "-O3 -funroll-loops -march=skylake-avx512". The Intel MPI implementation, version 2019, was employed. All experiments are based on a series of 20 repetitions, taking the best-performing sample. It has been ensured that, apart from disturbances of other jobs on the network of the machine, the deviation of the mean to the minimum is less than a few percent, which makes the reported performance data realistic.

The code is based on the two artifacts of the ExaDG and deal.II codes listed below. The configuration for SuperMUC-NG is based on the configuration listed at https://github.com/exadg/exadg/tree/860623c532bfb0c8a5a454426a3d1fd5a9161075/scripts/supermuc-ng Besides the main libraries listed below, we also include hypre-v2.20, petsc-3.14.5, METIS version 5.1.0. The full solver optimization are contained in the above libraries. The specific lung test case including the specific configuration of the grid is based on patient-specific data that cannot be shared at this point. Up to the final AD/AE deadline in August, we will add a generic test case with the same performance properties as public-domain input file. By then, Zenodo links for the artifacts will be created.

Author-Created or Modified Artifacts:

```
Persistent ID: 10.5281/zenodo.5176507
Artifact name: Fluid dynamics solver ExaDG, providing
↪  the application framework, as well as the
↪  underlying deal.II finite element library with
↪  mathematical basis of experiments
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: SuperMUC-NG, using Intel Xeon Platinum 8174 processors

Operating systems and versions: SUSE Linux Enterprise Server 12 SP3

Compilers and versions: gcc version 9.2 C++ and C compilers

Libraries and versions: Intel MPI 2019

Key algorithms: multigrid; conjugate gradient