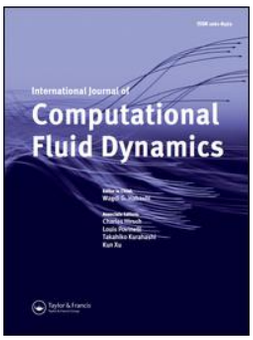

International Journal of Computational Fluid Dynamics

ISSN: 1061-8562 (Print) 1029-0257 (Online) Journal homepage: https://www.tandfonline.com/loi/gcfd20

Zonal Flow Solver (ZFS): a highly efficient multi-physics simulation framework

Andreas Lintermann, Matthias Meinke & Wolfgang Schröder

To cite this article: Andreas Lintermann, Matthias Meinke & Wolfgang Schröder (2020): Zonal Flow Solver (ZFS): a highly efficient multi-physics simulation framework, International Journal of Computational Fluid Dynamics, DOI: 10.1080/10618562.2020.1742328

To link to this article: https://doi.org/10.1080/10618562.2020.1742328

Published online: 18 Mar 2020.

INTERNATIONAL JOURNAL OF COMPUTATIONAL FLUID DYNAMICS
https://doi.org/10.1080/10618562.2020.1742328

Taylor & Francis
Taylor & Francis Group

Zonal Flow Solver (ZFS): a highly efficient multi-physics simulation framework

Andreas Lintermann [a,b], Matthias Meinke [b,c] and Wolfgang Schröder [b,c]

[a]Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Jülich, Germany; [b]Jülich Aachen Research Alliance Center for Simulation and Data Science (JARA-CSD), RWTH Aachen University and Forschungszentrum Jülich, Aachen and Jülich, Germany; [c]Institute of Aerodynamics and Chair of Fluids Mechanics, RWTH Aachen University, Aachen, Germany

ABSTRACT

Multi-physics simulations are at the heart of today's engineering applications. The trend is towards more realistic and detailed simulations, which demand highly resolved spatial and temporal scales of various physical mechanisms to solve engineering problems in a reasonable amount of time. As a consequence, numerical codes need to run efficiently on high-performance computers. Therefore, the framework Zonal Flow Solver (ZFS) featuring lattice-Boltzmann, finite-volume, discontinuous Galerkin, level set and Lagrange solvers has been developed. The solvers can be combined to simulate, e.g. quasi-incompressible and compressible flow, aeroacoustics, moving boundaries and particle dynamics. In this manuscript, the multi-physics implementation of the coupling mechanisms are presented. The parallelisation approach, the involved solvers and their scalability on state-of-the-art heterogeneous high-performance computers are discussed. Various multi-physics applications complement the discussion. The results show ZFS to be a highly efficient and flexible multi-purpose tool that can be used to solve varying classes of coupled problems.

ARTICLE HISTORY
Received 16 October 2019
Accepted 17 February 2020

KEYWORDS
Code coupling; multi-physics simulations; hierarchical cartesian meshes; high-performance computing; performance analysis

1. Introduction

Many disciplines in research and industry heavily rely on computational methods to solve complex problems. In fundamental research, understanding physical phenomena is a prerequisite for the advancement of enabling technologies and the development of reliable abstraction models. Groundbreaking data, methods, and models are first generated and developed in research, and ultimately end, after some evolutionary transformation, in software products applied by industry.

Since the cost for computing power continuously decreases, simulating physical effects with increasing detail by means of numerical simulations becomes more and more attractive. The pace of adapting numerical codes, however, does not compete with the advancement of computing hardware. This becomes obvious when considering the massive leap necessary to bring simulation software from the peta- to the exascale era of high-performance computing (HPC) as discussed by Heroux (2009) and Dongarra et al. (2011). New algorithms need to be developed that employ multi-level parallelisation techniques. Software has to be enabled to efficiently run on large-scale systems on millions of computational cores and on heterogeneous compute clusters including boosters such as GPU systems or other accelerators, see Eicker et al. (2016).

In engineering, a prominent field that needs to be addressed is the efficient computation of multi-physics multi-scale phenomena. Optimisations of existing engineering solutions heavily rely on in-depth understanding of the involved physics and their interplay. For example, to optimise turbojet engines, it is crucial to understand the individual physics that interact on a large variety of scales. To solve such complex problems, i.e. the interplay of combustion processes including spray injection and ignition, flame front propagation, heat transfer, fluid–structure interaction, aeroacoustics, aerodynamics, and multi-phase flow needs to be implemented. Many good co-simulation coupling frameworks such as MpCCI (Joppich and Kürschner 2006; Wolf et al. 2017), OpenPALM (Duchaine et al. 2015, 2017), or preCICE

CONTACT Andreas Lintermann A.Lintermann@fz-juelich.de

Supplemental data for this article can be accessed here. https://doi.org/10.1080/10618562.2020.1742328

(Bungartz et al. 2016) exist and can already deal with sub-problems of the aforementioned turbojet example. In co-simulation approaches, computing resources on HPC systems are given to each program independently and domain decomposition is done individually. As a consequence, the list of processors responsible for coupling needs to be updated continuously during runtime. In many co-simulation tools, e.g. in Bungartz et al. (2016), the coupling is primarily coordinated by some manager process, which informs responsible processes on updates. This, however, may generate a certain inevitable overhead and cause processors to wait for updates. Simulations are governed by many-to-many communications, not only among the individual solver processes but also across the solvers to realise coupling. Furthermore, subcycling techniques are required to reduce the amount of coupling steps. An advantage of co-simulator tools is that mature solvers can directly be coupled. In general, however, the meshes used for different solvers have different topologies. This necessitates to interpolate data between the non-conforming meshes. In OpenPALM (Duchaine et al. 2015), this is done in a two-step approach. First, the communication routes and the interpolation coefficients are computed and then, the inter-code synchronisation is executed. For moving boundaries, the first step needs to be executed at every time step. In MpCCI (Wolf et al. 2017), the approach is similar. It features a globally explicit coupling method for weakly coupled problems and an implicit iterative coupling method for strong physics coupling. The data exchange is managed by a coupling manager and exchanged data is interpolated from various meshes.

To avoid such problems, the Institute of Aerodynamics and Chair of Fluid Mechanics (AIA), RWTH Aachen University, started the development of the multi-physics software framework Zonal Flow Solver (ZFS) in 2004. This framework is also part of joint programs between AIA and the Simulation Laboratory 'Highly Scalable Fluids & Solids Engineering' of the Jülich Aachen Research Alliance Center for Simulation and Data Science, RWTH Aachen University and Forschungszentrum Jülich, the High Performance Computing Center Stuttgart (HLRS), and the Jülich Supercomputing Centre (JSC). In contrast to the aforementioned co-simulators, ZFS unites different solvers in one framework, allowing for process-local coupling that leads to a reduction of the communication effort. Figure 1 shows a brief overview of the framework. ZFS contains a massively parallel grid generator for the generation of large-scale unstructured hierarchical Cartesian meshes from a geometry input. It unites lattice-Boltzmann (LB), discontinuous Galerkin (DG), finite-volume (FV), level set, and Lagrangian solvers to tackle a variety of coupled simulation problems in fluid mechanics, combustion, biomedicine, aeroacoustics,

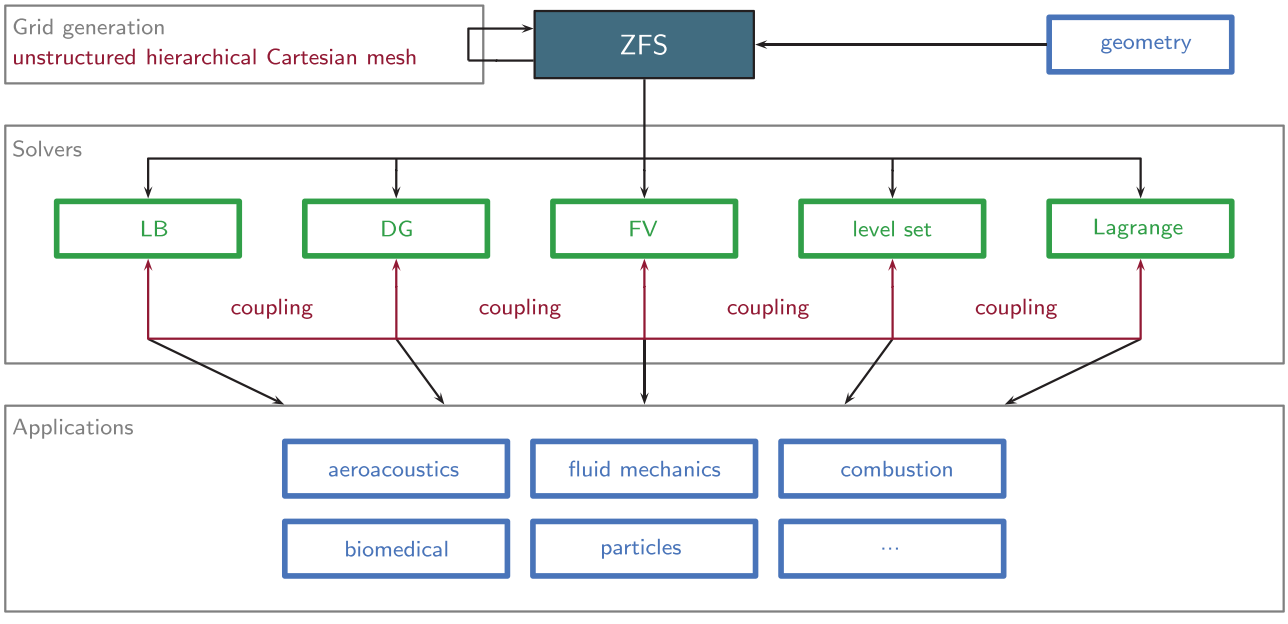

Figure 1. Brief overview of the ZFS framework. The software contains a massively parallel grid generator for the generation of hierarchical Cartesian meshes, lattice-Boltzmann solvers (LB), a discontinuous Galerkin solver (DG), a finite-volume solver (FV), a level set and a Lagrange solver. The solvers can be coupled for a variety of applications.

including moving boundaries and particle dynamics. The single Cartesian mesh hierarchy in ZFS is unique to the whole framework and hence, does not require an additional interpolation of the coupled variables between different solvers across processes, except for process-local data exchange if coupled solvers reside on different mesh levels. Cells of the mesh can arbitrarily be employed, refined, and coarsened by any of the solvers. The dynamic refinement and coarsening is directly linked to a load-balancing procedure, which guarantees reduced communication. The data locality implies that there is no need to transfer information across the network between coupled modules. The modular approach allows to easily extend the capabilities of ZFS. ZFS is written in `C++11` and contains 384,200 lines of code. It is dependent on HDF5 (Folk and Pourmal 2010) or parallel NetCDF as presented in Li et al. (2003) for I/O and FFTW (Frigo and Johnson 2005) for, e.g. spectral flow field initialisation. Multi-level parallelisation is realised by OpenMP and the message passing interface (MPI) using asynchronous communication. Some code sections are furthermore parallelised to be executable on GPUs and Intel Xeon Phi Knights Landing (KNL).

Note that there exist other octree-based approaches such as presented by Burstedde, Wilcox, and Ghattas (2011) or Klimach, Jain, and Roller (2014). The former is an extendable mesh framework that allows for adaptive mesh refinement on a forest of octrees and requires to additionally implement methods and models for multi-physics simulations. The latter uses either LB or DG methods on a single mesh hierarchy to solve flow problems. Of course, there exist further scalable simulation frameworks that implement coupled approaches such as Alya (Vázquez et al. 2016), CODE_Saturne (Fournier et al. 2011), or MOOSE (Gaston et al. 2009). Alya is a finite element code that can use both implicit or explicit schemes to solve coupled problems. It has shown scalability up to 10^5 cores. CODE_Saturne relies on a FV method and scales up to 786,432 cores on BlueGene/Q machines.[1] In contrast, MOOSE relies on fully implicit methods and scales up to 32, 768 cores with an efficiency of 77% (Permann et al. 2019).

In the following, coupling strategies in ZFS and their implementation details as well as the individual solvers are presented in Section 2 before some examples and performance results are presented in Section 3. These results emphasise the capabilities of the coupling methods. Finally, some conclusions are drawn in Section 4.

2. Numerical methods

Multi-physics computations with ZFS necessitate to follow a standard simulation pipeline, i.e. a preprocessing of the input data to the simulation, the simulation itself, and a post-processing of the output data. In more detail, the following steps are subsequently performed:

(a) generation of a computational mesh and a parallel geometry using an input geometry;
(b) multi-physics simulation preparation;
(c) multi-physics simulation using coupled solvers on HPC systems;
(d) in-situ or post-processing of output data.

These steps are explained in Section 2.1. Subsequently, the individual solvers that can be coupled in ZFS are introduced in Section 2.2.

2.1. Simulation pipeline

In this section, the simulation pipeline is described, i.e. Section 2.1.1 presents details on the generation of a computational mesh and a parallelised geometry. Section 2.1.2 explains how the mesh and geometry are prepared to be efficiently used in the solving process and Section 2.1.3 explains the coupling approach. Finally, some brief information is given on the processing methods to analyse simulation data in Section 2.1.4.

2.1.1. Mesh and parallel geometry generation

The massively parallel grid generator and the parallel geometry generator have been developed in Lintermann et al. (2014), Lintermann (2016) and base on the work presented in Hartmann, Meinke, and Schröder (2008a). The software generates hierarchical Cartesian meshes and reads a geometry that describes one or multiple two- or three-dimensional objects defining the volume of interest for a simulation. In general, the geometry is provided in standard tessellation language (STL). Note that initially the geometry is not parallel and that the parallelisation, i.e. the creation of a geometry file that can be processed in parallel at simulation time, is part of the meshing process.

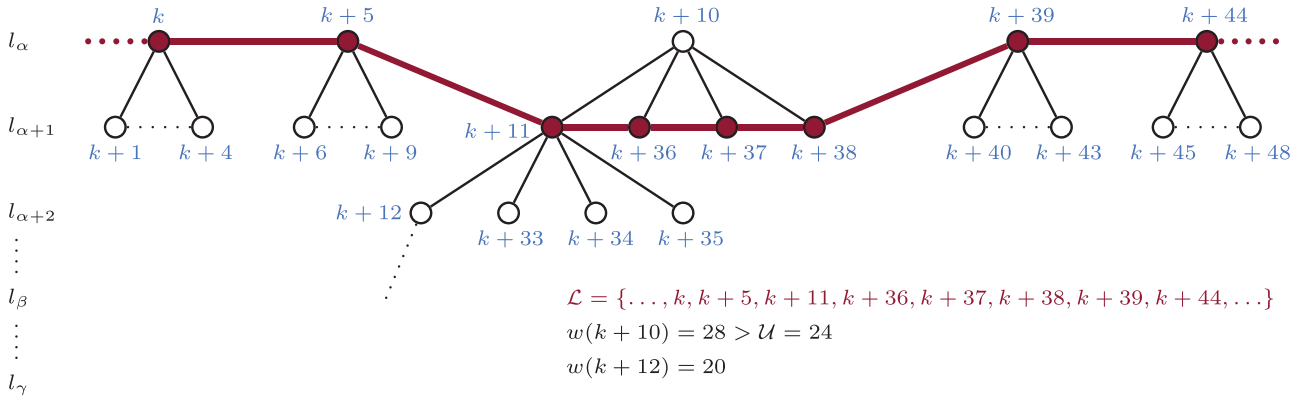

Figure 2. Example of a (quad)tree as generated by the massively parallel gird generator of ZFS. As a threshold, a weight of $\mathcal{U} = 24$ is chosen. The cells are ordered by the Hilbert curve on the coarsest level and in between in depth-first order (*z*-curve). The weight of cell $k+10$ is $w(k+10) = 28$ and includes $w(k+12) = 20$. It is $w(k+10) > \mathcal{U}$ and hence the children of $k+10$ are moved to $\mathcal{L}$ while $k+10$ is removed from $\mathcal{L}$.

In a first step, each processor reads the initial geometry file and stores it in an alternating digital tree (ADT) (Bonet and Peraire 1991) for accelerated triangle lookup. Then, each process starts to refine an initial cube (in 3D) or square (in 2D) placed around the geometry on level l_0. The corresponding parent–child relations are stored in an octree or quadtree $\mathcal{T}$, see Figure 2. The refinement process continues up to a user-defined level l_α. At each iteration, cells outside the geometry are removed. Therefore, cells intersecting the STL are marked as boundary cells. Inside cells are recursively marked by initialising an inside flooding at an internal cell, which is bound by the boundary cells. The initial internal cell is found by casting rays in the Cartesian directions and by counting the number of intersections with the STL, i.e. an even number classifies a cell as outside and an odd number as inside. Those cells that are not marked by the recursive flooding algorithm are removed from the memory and the tree. Note that only those children are checked for an STL intersection that have a parent, which also intersected the geometry on the previous level. On level l_α all coarser levels $l < l_\alpha$ are removed from the tree and the remaining cells are decomposed by the number of processors along a space-filling Hilbert curve, which guarantees local compactness (for an overview on space-filling curves, the reader is referred to Sagan (1994)). This is done by keeping only those cells on l_α that belong to the local processor and by determining the neighbouring processors by the cell neighbourhood of the process-boundary cells. In a second stage, each processor then starts to uniformly refine its own set of cells up to a level $l_\beta > l_\alpha$ and removes cells outside the geometry.

The third stage enables local refinement, which refines the mesh based on user-defined objects placed in the simulation domain (patch refinement) or on STL distances (boundary refinement) up to a level $l_\gamma > l_\beta$. While patch refinement is trivial, boundary refinement, for the first time, requires inter-process communication (IPC). At each refinement iteration, a cell-based level set is generated by recursively counting the distance in cell units from the STL across MPI domains. In more detail, for each level, the user can decide the width of the finest layer and the (smoothing) distance to the next coarser level. The required distances per level are then precomputed and continuously generated. Thereby, it is ensured that only valid meshes with a level difference $\Delta l = l_k - l_{k-1} = 1$ at level interfaces are generated.

In the last stage, local cell ids are translated to global cell ids, neighbourhood relations across MPI domains are generated, and the mesh is stored in Hilbert order on level l_α and in *z*-order on all levels $l > l_\alpha$ via parallel I/O using either HDF5 or parallel netCDF in a single file. The ordering of the cells in the file is exemplarily shown in Figure 2. Together with the mesh, a version of the Hilbert curve is stored for later mesh decomposition in the simulation pre-processing. Let $\mathcal{L}$ denote such a cell list. Note that this list does not necessarily represent the mesh on level l_α but can include further levels $l > l_\alpha$ to prevent massive numbers of children for cells on l_α, which may lead to imbalances during computation. Such a modified list is highlighted in Figure 2 by the red line connecting

the nodes. For each element $e \in \mathcal{L}$ a weight $w(e)$ is stored. In the most trivial case, where each cell in the mesh has the same computational cost, it is $w(e) = |C_e|$, where $|C_e|$ is the number of children of element e. It is, however, possible to define areas or volumes with different weights, e.g. intersected cells with higher computational costs are assigned a higher weight. Furthermore, the targeted solution method determines the cell weight, which is why cells are assigned to a set of solvers by means of an additional `bitset` $\mathcal{B}$ (see Section 2.1.3). The overall weight, which is the sum of the weights of offsprings and a user-defined threshold $\mathcal{U}$, is then used for the decision if a cell stays in $\mathcal{L}$ or if its offsprings are integrated instead. The (modified) list $\mathcal{L}$ is furthermore used to store the STL triangle information per cell in a separate geometry file, i.e. for each cell in this list the number of intersecting triangles and the triangle information is stored for later parallel retrieval.

To this end, the grid generator is also able to load-balance the grid generation by redistributing cells. Note that the meshing process is decoupled from the simulation. In general, mesh generation requires way less memory and computing power than a simulation. This allows to run the grid generation on a small number of processes and run the simulation on an quasi-arbitrary large number of processes, which is independent from the number of processes employed for meshing. Furthermore, the mesh is stored on disk in a compressed way leading to only a small memory consumption overhead. For more details on the mesh and parallel geometry generation, the interested reader is referred to Lintermann et al. (2014); Lintermann (2016).

2.1.2. Simulation preparation

The simulation is prepared as outlined in Algorithm 1. Note that this algorithm is a simplification of the implementation in ZFS, which generates individual instances of coupling methods at compile time for performance reasons. It is the aim of the pseudo code to structurally describe the runtime of ZFS, which can be used for re-implementation in other simulation codes. In a first step, simulation parameters such as the Reynolds and Mach numbers, which are stored in a separate file, are read. A geometry file contains information on the geometry and associated boundary conditions per geometry segment. Initially, ZFS' first MPI rank reads this information and distributes it among all participating ranks via `MPI_Bcast` (Algorithm 1, line 2).

Subsequently, each rank reads the list $\mathcal{L}$ and the associated weights. The mesh is then decomposed on the number of processors based on the weights stored per $e \in \mathcal{L}$. If each cell has the same weight, each processor receives approximately the same number of cells with a fluctuation in the order of the maximum allowed weight $\mathcal{U}$. In coupled simulations, however, each process receives an amount of cells, which is based on an equal distribution of the weights. That is, the workload per process is approximately the same for all processes although some processes might only contain cells of one solver type. The efficient parallel partitioning is based on the method described in Lieber and Nagel (2014). The initial weights are based on estimated values obtained from a-priori runtime measurements for the various methods. The exact values for a perfect initial balancing can, however, not be determined as they heavily depend on the advanced methods chosen by the user at run time. For example, each solver has a set of boundary conditions on the order of $\mathcal{O}(100)$ from which the user can choose. Furthermore, the user-defined stencil width and the communication pattern may influence the balance. Therefore, it is possible to activate dynamic load-balancing at run time, which measures the execution times for the various MPI ranks. Based on the measured times and the corresponding divergence, a redistribution of the corresponding cells takes place to reestablish balance (see Section 2.1.3). This method is especially useful when considering moving boundaries and adaptive remeshing. Subsequent to setting the weights, each process p determines its neighbouring processors $\mathcal{N}(p)$, detects its window cells, i.e. those cells that are neighbours to cells on some $n \in \mathcal{N}(p)$, and sends this information to n by evaluating the list and the neighbourhood information. These cells are then created as halo cells, i.e. exact copies of the window cells on the foreign domains. That is, ZFS implements a point-to-point communication using MPI. Depending on the method, i.e. the stencil width, one or two layers are generated. Then, the direct neighbours for all cells are determined. With this information at hand, the space- and edge-diagonal neighbourhood is determined. This also includes neighbourhood relations between cells on different levels. This is all performed in `readMesh()` in line 3 of Algorithm 1. Note that for the exchange via MPI, non-blocking

Algorithm 1 Runtime execution of ZFS in pseudo code.

```
1:  function ZFSEnvironment
2:      readSimulationParameters();                          ▷ read simulation parameters
3:      readMesh();                                          ▷ read mesh in parallel; prepare parallelisation
4:      readGeometry();                                      ▷ read geometry in parallel; store in ADT
5:      initProcessing();                                    ▷ initialise in-situ processing
6:
7:      solverlist[] = createSolvers();                      ▷ init solvers, solver/mesh relations, BCs, coupling
8:
9:      while target time steps not reached do
10:         while strong coupling not converged do           ▷ depends on coupling method
11:             for all Solvers in solverlist[] do           ▷ executed concurrently
12:                 advanceSolver();
13:
14:                 //(the following depends on the coupling method)
15:                 dataExchangeCoupledSolvers();            ▷ couple solvers
16:                 subTimeStepping();                       ▷ e.g. smaller time steps for certain solvers
17:                 interleaving();                          ▷ e.g. for Runge-Kutta
18:
19:                 applyBCs();
20:                 IPC();                                   ▷ exchange information in window/halo cells via non-blocking MPI
21:                 processData();                           ▷ write to disk / analyse in-situ based on interval
22:                 adaptAndLoadBalance();                   ▷ mesh adaption, dynamic load-balancing
23:
24: function createSolvers
25:     for all Solvers in parameter file do
26:         initSolver();
27:         createMeshRelations();                           ▷ map cells in mesh to solver
28:         setBoundaryConditions();                         ▷ assign BCs from geometry segments to cells
29:         coupleSolvers();                                 ▷ let environment know about current solver coupling
30:         solverList[i] = solver;                          ▷ let environment know what solvers to couple
31:     return solverList[];
```

communication is used, which allows to continue with the calculation with the communication in the background (see Section 2.1.3).

In line 4 of Algorithm 1, the geometry is read in parallel from the custom geometry file. Therefore, each process p determines the number of triangles and the offset in the file to start the reading process based on the number of triangle elements covered by $e(p) \subset \mathcal{L}$. The according triangles are again stored in an ADT. Since triangles may also live in window cells, relating triangles are copied to $n \in \mathcal{N}(p)$. Obviously, this algorithm introduces redundant information since triangles may live on multiple MPI ranks, also because multiple $e \in \mathcal{L}$ may share a triangle. However, in Section 3 it will be shown that despite this overhead this is still more efficient than reading the geometry into memory on each processor.

Finally, some in-situ data processing is initialised before in a subsequent step the methods for the computation are set and the simulation is started (Algorithm 1, line 5).

2.1.3. Multi-physics simulation

Within ZFS, multiple solvers, each responsible for efficiently computing a solution of a set of partial differential equations representing specific physics, interact to yield a multi-physics simulation. They all operate on the same mesh hierarchy, i.e. the quad-/octree $\mathcal{T}$ is the basis for the spatial discretisation of the governing equations and cells are assigned to different `C++` classes, each implementing a dedicated numerical method. The tree only holds the information on cell locations, cell levels, parent–child relations and neighbourhood information. All other solver-relevant information is stored in corresponding solver classes, i.e. each class has a set of variables $\mathcal{Q}_v$. A unidirectional mapping from cells $c \in \mathcal{T}$ to the corresponding solver class is realised by a `bitset`.

In the beginning, the user-requested solvers $\mathcal{S}$ are collected and the according classes are instantiated and initialised. That is, the cells in $\mathcal{T}$ are identified, the links are created, the variables $\mathcal{Q}_v$ are initialised, the boundary conditions are set, and solver-dependencies are established. Cells are assigned to the different solver classes and coupling links between the solvers are created according to the `bitset` $\mathcal{B}$ defined in the grid generation (see Section 2.1.1). This is done in function `createSolver()` of Algorithm 1 (lines 7, 24–31). All solvers use explicit time stepping and hence, coupling is implemented via a loop iterating over the time steps Δt_s of the solver $s \in \mathcal{S}$ with the largest time step (Algorithm 1, lines 9–22). The explicit

methods are favoured over using implicit methods as they allow for exploiting data locality to achieve higher scalability, to reduce the memory footprint compared to implicit methods, and due to the necessity to resolve the physical time scales. Furthermore, in coupled setups, implicit methods might have the disadvantage to suffer from a slow convergence. Within this loop, solvers either need to iterate to reach convergence when strong coupling is required (Algorithm 1, lines 10–22), or do a weak coupling by ignoring line 10. Furthermore, in this loop, sub-time stepping for solvers requiring smaller time steps can be performed (Algorithm 1, lines 14–17). Note that the decision if strong or weak coupling is necessary is defined by the simulation problem, e.g. fluid–structure interaction problems frequently necessitate to converge the force exchange between solid and fluid domains, while weak coupling is employed, e.g. in one-sided coupled simulations such as in Lagrangian particle simulations. Within each process, the solvers are executed concurrently to avoid idling. Each solver module connects its locally stored data via a proxy class to the octree. Coupling between different modules is realised by a superordinate coupling class that accesses the necessary data required by the underlying solver instances. Note that more details on the employed coupling methods can be found in the examples presented in Section 3.

Communication in ZFS (function `IPC()`, Algorithm 1, line 20) is implemented such that interacting physics and the corresponding cells share the same MPI domain. Thus, information can be exchanged in memory and unnecessary communication or I/O is avoided. It should be noted that to implement moving boundaries, i.e. where surface coupling is necessary, a level-set approach is employed (see Section 2.2.4). That is, cells in the octree are marked as level-set cells and carry additional signed-distance information, from which the moving boundaries are reconstructed. Since the partition of the computational domain is cell- and cell-weight-based, fluid and level-set cells automatically reside on the same MPI rank.

Where possible, Runge-Kutta steps for temporal integration are interleaved to obtain increased efficiency. This interleaving process for explicit time integration can, e.g. be used in computational fluid dynamics (CFD)/computational aeroacoustics (CAA) coupled problems, i.e. using the volume-coupled FV and DG solvers as described in Sections 2.2.1 and 2.2.3. In cases where a load-imbalance for the various solvers exists, some substeps for a specific solver have longer execution times and the faster processes need to wait for synchronisation. This interleaving can be highly advantageous. Instead of having first the substeps of the first solver being executed and subsequently the substeps of the second one, the substeps for both methods are interleaved to avoid idling time. When interleaving, the resources of the waiting solver can be given to the other solver such that a more efficient execution is obtained. Communication uses non-blocking MPI via `MPI_Isend` and `MPI_Irecv` and employs persistent communicators binding the list of communication arguments for faster execution. Note that on top of using non-blocking communication, the workload per process is approximately the same across the whole MPI communicator since the initial distribution is based on the cell weights. This communication concept allows to run a single solver or multiple coupled solvers on a single process.

The function `adaptAndLoadBalance()` (Algorithm 1, line 22) is responsible for refining the computational mesh, i.e. it performs an update of the quad-/octree. For mesh modification, sensors or multi-grid error estimators are employed. Such sensors can, e.g. detect high shear, vorticity, or movement of a body and are indicators for mesh modifications. The mesh refinement is executed as described in Hartmann, Meinke, and Schröder (2008a). The function `adaptAndLoadBalance()` is implemented as a communication class between the solver classes and the tree class. It collects refinement or coarsening requests from the solvers and triggers the according mesh modification. The solvers are subsequently informed on the updated mesh and are responsible to propagate these updates into their internal data structures. Furthermore, the coupling between affected solvers is updated, i.e. new links for correctly exchanging information between the solvers are created. In each refinement execution, the sum of weights per process is recalculated. Furthermore, the imbalance is determined by run-time measurements for the various MPI ranks. If the imbalance diverges by a user-defined percentage from the average weight over all processes, the mesh is redistributed based on the weights and the ratios of imbalance as determined by the run-time measurements to restore load balance, see Schneiders et al. (2015). Therefore, the process offsets are recalculated, affected mesh hierarchies are exchanged via MPI, and solvers are informed and reinitialised.

2.1.4. Processing of simulation data

ZFS features an in-situ processing tool, which is initialised during simulation preparation, see Section 2.1.2 and Algorithm 1, line 5. Via the parameter file the user can request in-situ operations such as the computation of the Reynolds stress tensor, averages, mesh reduction operations, point, line and slice probing. Therefore, a dedicated processing class holds all necessary variables that are written to disk in predefined intervals as analysis results. It is, however, not necessary to run analyses at runtime, they can also be performed before and after a simulation. The former is of interest, when a solution has previously been written to disk and is reloaded into ZFS to massively parallel analyse it, while the latter can be executed with the solution in memory. In any case, the solution and analysis results that reside on disk can be opened and further analysed in parallel using the post-processing tool ParaView as presented in Henderson, Ahrens, and Law (2004). A parallel reader for ParaView has been implemented, which derives its functionality from the function `readMesh()`, Algorithm 1, line 3, converts the mesh to a `vtkUnstructredMesh`, and generates window and halo cells to enable ParaView's filter pipeline execution across MPI ranks. Furthermore, ZFS has recently been coupled to the visualisation toolkit VisIt (Childs et al. 2012) via the in-situ interface JUSITU, developed at JSC.

2.2. Solvers of ZFS

In the following, more details on the implementation of the different solvers are given, i.e. Section 2.2.1 discusses an FM method, Section 2.2.2 describes LB methods, Section 2.2.3 a DG method, and Section 2.2.4 a level set method. Finally, Section 2.2.5 presents the Lagrange-based particle solver of ZFS.

2.2.1. Finite-volume solver

The discretisation uses a cell-centred FV scheme in which the gradients $\nabla\hat{\mathbf{Q}}$ of the primitive variables $\hat{\mathbf{Q}} = \{\rho, v, E\}$ are calculated by a weighted least-squares reconstruction method (Schneiders et al. 2016). For the left and right surface interpolants of the primitive variables a second-order accurate monotone upstream scheme for conservation laws (MUSCL) approach as introduced in van Leer (1979) is used. Time integration is performed by a five-stage second-order accurate and stability-optimised predictor-corrector Runge-Kutta scheme suited for efficient multi-physics coupling (Schneiders et al. 2016). A modified version of the advection upstream splitting method (AUSM) from Liou and Steffen (1993) is employed for the computation of the inviscid fluxes. The viscous fluxes are computed by a finite difference recentring approach (Berger and Aftosmis 2012). The FV solver is, e.g. used for the computation of compressible flow. Examples of the application of the FV solver can be found in Sections 3.3 and 3.4.

In large-eddy simulations (LES), sub-grid scales are modelled by a monotone integrated LES (MILES) approach as presented in Boris et al. (1992) and Meinke et al. (2002). That is, suitable discretisation schemes act as an implicit subgrid-scale model by numerically dissipating the energy at the smallest scales. Various Dirichlet and Neumann conditions for in- and outflow can be set, including characteristic conditions. Additionally, reflective numerical waves are damped by sponge layers, cf. Freund (1997). Finally, no-slip wall boundary conditions are employed by a strictly conservative cut-cell method as presented in Schneiders et al. (2013, 2016). That is, cells are reshaped based on the location of the geometry and the discretisation is adapted such that mass, momentum, and energy is conserved.

2.2.2. Lattice-Boltzmann solver

The LB method solves the discrete Boltzmann equation with the Bhatnagar–Gross–Krook approximation of the collision term, i.e. the lattice-BGK equation (Bhatnagar, Gross, and Krook 1954; Hänel 2004)

$$f_i\left(\mathbf{x} + \boldsymbol{\Xi}_i \Delta t, t + \Delta t\right) = f_i\left(\mathbf{x}, t\right) + \omega \Delta t \cdot \left(f_i^{eq}\left(\mathbf{x}, t\right) - f_i\left(\mathbf{x}, t\right)\right), \quad (1)$$

for the particle probability distribution functions (PPDFs) f_i using discrete directions i in DXQY models from Qian, D'Humières, and Lallemand (1992), i.e. D2Q7, D2Q9, D3Q15, D3Q19, and D3Q27. In Equation (1), $\mathbf{x}$ represents the spatial location, $\boldsymbol{\Xi}_i$ is a non-dimensional molecular velocity vector, Δt is the time increment, and ω is a viscosity-dependent relaxation factor. The discrete Maxwell equilibrium

distribution function is given by

$$f^{eq}(x_\alpha, t) = \rho t_p \left[1 + \frac{\nu_\alpha \xi_\alpha}{c_s^2} + \frac{\nu_\alpha \nu_\beta}{2c_s^2} \cdot \left(\frac{\xi_\alpha \xi_\beta}{c_s^2} - \delta_{\alpha\beta}\right)\right]. \tag{2}$$

The quantities x_α define the vector components of $\mathbf{x}$, $\xi_{\alpha,\beta}$ are the vector components of $\mathbf{\Xi}$, t_p is a direction-dependent weighting coefficient, $\nu_{\alpha,\beta}$ define the local velocities, $\delta_{\alpha,\beta}$ is the Kronecker delta, c_s is the non-dimensional speed of sound, and ρ is the density. For the subscripts α, β it is $\alpha, \beta \in \{1, 2\}$ in 2D and $\alpha, \beta \in \{1, 2, 3\}$ in 3D. The macroscopic variables of the flow can be obtained from the moments of the PPDFs, cf. Hänel (2004).

In addition to the lattice-BGK approach, which realises a single relaxation time (SRT) method, ZFS features the multiple relaxation time (MRT) method as presented in D'Humières et al. (2002). Unlike in former formulations, different relaxation times are used to achieve optimised stability, e.g. using the moment and diagonal relaxation matrices $\underline{\mathbf{M}}$ and $\underline{\mathbf{K}}$ (Lallemand and Luo 2000). The MRT-LBGK equation reads

$$\begin{aligned}&\mathbf{f}\left(\mathbf{x} + \mathbf{\Xi}_i \Delta t, t + \Delta t\right)\\ &\quad = \mathbf{f}\left(\mathbf{x}, t\right) - \underline{\mathbf{M}}^{-1}\underline{\mathbf{K}} \cdot \left[\mathbf{m}\left(\mathbf{x}, t\right) - \mathbf{m}^{eq}\left(\mathbf{x}, t\right)\right],\end{aligned} \tag{3}$$

with $\mathbf{f}$ being the vector of the PPDFs and

$$\mathbf{m} = \left(\rho, e, \epsilon, j_0, q_0, j_1, q_1, \tau_{00}, \tau_{11}\right)^T, \tag{4}$$

$$\begin{aligned}\mathbf{m}^{eq} = \big(&1, -2\rho + 3(j_0^2 + j_1^2), \rho - 3(j_0^2 + j_1^2), j_0,\\ &- j_0, j_1, -j_1, j_0^2 + j_1^2, j_0 j_1\big)^T,\end{aligned} \tag{5}$$

where e is related to the kinetic energy, ϵ is related to the kinetic energy squared, $j_{0,1}$ are the two momentum components, $q_{0,1}$ are proportional to the energy fluxes, and $\tau_{00,11}$ are related to the diagonal and off-diagonal components of the viscous stress tensor $\underline{\boldsymbol{\tau}}$.

Thermal flow is simulated by additionally solving a passive scalar transport equation for the temperature with a multi-distribution function (MDF) approach, i.e. by solving

$$\begin{aligned}&g_i\left(\mathbf{x} + \mathbf{\Xi}_i \Delta t, t + \Delta t\right)\\ &\quad = g_i\left(\mathbf{x}, t\right) + \Omega \Delta t \cdot \left(g_i^{eq}\left(\mathbf{x}, t\right) - g_i\left(\mathbf{x}, t\right)\right),\end{aligned} \tag{6}$$

with Ω representing the heat-conductivity-dependent relaxation of the thermal model, the equilibrium distribution function

$$g^{eq}(x_\alpha, t) = T t_p \left[1 + \frac{\nu_\alpha \xi_\alpha}{c_s^2} + \frac{\nu_\alpha \nu_\beta}{2c_s^2} \cdot \left(\frac{\xi_\alpha \xi_\beta}{c_s^2} - \delta_{\alpha\beta}\right)\right], \tag{7}$$

and the temperature T.

Local refinement is implemented by the method presented in Dupuis and Chopard (2003) and Lintermann, Meinke, and Schröder (2012). That is, an overlay of coarse and fine cells is required at the interface region. In the transition region, not all required PPDFs are available on the coarse and fine grid. The missing incoming PPDFs are reconstructed by a transformation step, whereas the macroscopic variables are recovered by tri-linear interpolation. To keep the viscosity constant across two succeeding mesh levels l_k and l_{k+1} the relaxation time is adapted to

$$\omega \Delta t_{k+1} = \frac{\Delta x_k}{\Delta x_{k+1}} \left(\omega \Delta t_k - \frac{1}{2}\right) + \frac{1}{2} \tag{8}$$

and missing PPDFs are reconstructed by

$$f_{i,k+1}^{in} = \tilde{f}_i^{eq} + \left(\tilde{f}_{i,k} - \tilde{f}_i^{eq}\right) \frac{\Delta x_{k+1}}{\Delta x_k} \cdot \frac{\omega_{k+1}}{\omega_k}, \tag{9}$$

$$f_{i,k}^{in} = f_i^{eq} + \left(f_{i,k+1} - f_i^{eq}\right) \frac{\Delta x_k}{\Delta x_{k+1}} \cdot \frac{\omega_k}{\omega_{k+1}}, \tag{10}$$

where PPDFs denoted by a $\langle\tilde{\cdot}\rangle$ are spatially interpolated from the other mesh level.

Various boundary conditions exist for the LB method, e.g. adaptive Dirichlet conditions for the density, Neumann conditions for the density and velocity, Saint-Venant/Wanzel conditions at inlets (Lintermann, Meinke, and Schröder 2013; Waldmann et al. 2020), and second-order accurate no-slip all boundary conditions from Bouzidi, Firdaouss, and Lallemand (2001). For more information on the LB method implemented in ZFS, the reader is referred to Freitas and Schröder (2008), Eitel et al. (2010), Eitel, Soodt, and Schröder (2010), Freitas et al. (2011), Lintermann et al. (2013), Lintermann, Meinke, and Schröder (2011, 2012, 2013), Eitel-Amor, Meinke, and Schrder (2013), Lintermann and Schröder (2017a, 2017b), Waldmann et al. (2020), and Vogt et al. (2018).

2.2.3. Discontinuous Galerkin solver

The DG spectral element method (DGSEM) developed in Kopriva, Woodruff, and Hussaini (2000) is

used. The weak formulation of a general system of hyperbolic equations in Cartesian coordinates for an element E with volume V and surface area A is given by

$$\int_V \underline{\mathbf{J}} \frac{\partial \mathbf{Q}}{\partial t} \phi \, \mathrm{d}\boldsymbol{x} + \int_A (\mathbf{f} \cdot \mathbf{n}) \phi \, \mathrm{d}s - \int_V \mathbf{f} \cdot \nabla \phi \, \mathrm{d}\mathbf{x} = \int_V \underline{\mathbf{J}} \mathbf{O} \phi \, \mathrm{d}\mathbf{x}, \tag{11}$$

where $\underline{\mathbf{J}}$ is the Jacobian, ϕ is a test function, $(\mathbf{f} \cdot \mathbf{n})$ is the surface flux, $\mathbf{Q}$ is the vector of conservative variables, and $\mathbf{O}$ is a source term. The solution, fluxes and sources $\mathbf{G} = \{\mathbf{Q}, \mathbf{f}, \mathbf{O}\}$ in three dimensions $\{i, j, k\}$ are approximated by

$$\mathbf{G} \approx \mathbf{G}_h = \sum_{i,j,k=0}^{N} \hat{\mathbf{G}}_{ijk} \psi_{ijk}, \tag{12}$$

with nodal coefficients $\hat{\mathbf{G}}_{ijk}$, and basis functions $\psi_{ijk} = l_i l_j l_k$, generated from linear combinations of Lagrange polynomials $l_{i,j,k}$. For the fluxes the local Lax–Friedrichs formulation is employed. This leads to

$$\int_V \underline{\mathbf{J}} \frac{\partial \mathbf{Q}_h}{\partial t} \psi_{ijk} \, \mathrm{d}\boldsymbol{x} + \int_A (\mathbf{f} \cdot \mathbf{n}) \psi_{ijk} \, \mathrm{d}s - \int_V \mathbf{f}_h \cdot \nabla \psi_{ijk} \, \mathrm{d}\mathbf{x} = \int_V \underline{\mathbf{J}} \mathbf{O}_h \psi_{ijk} \, \mathrm{d}\mathbf{x}, \tag{13}$$

which is numerically solved using Gauss quadrature. The solution is advanced by integrating the semi-discrete DG operator $\mathcal{L}(\mathbf{Q}, t) = \partial \mathbf{Q} / \partial t$ in time by a low-storage implementation of an explicit five-stage fourth-order Runge-Kutta scheme. Given a polynomial degree of N, the overall number of degrees of freedom is given by

$$D = (N+1)^d \cdot |C|, \tag{14}$$

where d denotes the dimensionality of the problem and $|C|$ is the number of cells. The DG solver is, e.g. used for the simulation of aeroacoustics. Examples are discussed in Section 3.4. For more information on the implementation, also on refinement using mortar elements, the reader is referred to Schlottke et al. (2015) and Schlottke-Lakemper et al. (2017, 2016).

2.2.4. Level set solver

Interfaces of moving boundaries are represented as a zero level set φ_0 (Osher and Fedkiw 2003; Hartmann, Meinke, and Schröder 2008b; Günther, Meinke, and Schröder 2014), which separates, e.g. the solid from a fluid region. The scalar level set function evolves in time and is defined in $\mathbb{R}^d$, i.e. it is $\varphi_0 = \{(\mathbf{x}, t) : \varphi(\mathbf{x}, t) = 0\}$. The interface decomposes the space $\mathbf{x} \in \mathbb{R}^d$ into

$$\mathbf{x} = \begin{cases} \Omega_\varphi^+, & \varphi > 0, \\ \varphi_0, & \varphi = 0, \\ \Omega_\varphi^-, & \varphi < 0. \end{cases} \tag{15}$$

The zero level set moves with the extension velocity

$$\mathbf{v}_\varphi = \mathbf{v} + s_\varphi \mathbf{n}_\varphi, \tag{16}$$

where $\mathbf{v}$ is the fluid velocity, s_φ is the local speed of propagation induced by, e.g. curvature, and $\mathbf{n}_\varphi$ is the normal on the interface on φ_0 pointing into Ω_φ^-, i.e.

$$\mathbf{n} = -\frac{\nabla \varphi}{|\nabla \varphi|}. \tag{17}$$

The curvature is then given by $\kappa = \nabla \cdot \mathbf{n}$. With this, the level set equation can be expressed as

$$\frac{\partial \varphi}{\partial t} + \mathbf{v}_\varphi \cdot \nabla \varphi = 0. \tag{18}$$

For efficient computation of the level set, a localised approach from Peng et al. (1999) is employed, i.e. a band around φ_0 is formed that holds those cells that have a distance of $\delta\varphi_0$ from φ_0. Temporal integration of Equation (18) uses a three-step third-order accurate TVD Runge-Kutta scheme. Spatial discretisation is performed with unlimited third- and fifth-order upstream central schemes. Instabilities at the boundary at distance $\delta\varphi_0$ are avoided by employing a Heavyside function $c(\mathbf{x})$

$$\frac{\partial \varphi}{\partial t} + c(\mathbf{x}) \mathbf{v}_\varphi \cdot \nabla \varphi = 0 \tag{19}$$

and a reduced-order discretisation near the boundary, see Hartmann, Meinke, and Schröder (2008b). To furthermore avoid the displacement of the zero level set within the reinitialisation step, which is necessary to reestablish the signed distance property, a constrained reinitialisation step presented in Hartmann, Meinke, and Schröder (2008b, 2008c, 2010, 2011) is employed.

For the numerical prediction of flame fronts the G-equation (Schlimpert et al. 2016)

$$\frac{\partial G}{\partial t} + \left(\tilde{\mathbf{v}} + \frac{\rho_0}{\bar{\rho}} \hat{\mathbf{v}}_f \check{\mathbf{n}} \right) \cdot \nabla G = 0 \tag{20}$$

is solved with the normal vector on the flame front pointing into the unburnt gas region

$$\check{\mathbf{n}} = -\frac{1}{|\nabla G|} \left(\frac{\partial G}{\partial x_0}, \frac{\partial G}{\partial x_1}, \frac{\partial G}{\partial x_2} \right)^T, \tag{21}$$

the local Favre-filtered flow velocity $\tilde{\mathbf{v}}$, flame speed $\hat{\mathbf{v}}_f$, and the freestream and the spatial Reynolds-filtered densities ρ_0 and $\bar{\rho}$ are. The curvature of the flame front is then given by $\check{\kappa} = \nabla \cdot \check{\mathbf{n}}$.

2.2.5. Lagrange particle solver

The movement of each particle is simulated using either a one- or two-way coupling, i.e. particles do either not influence the flow solution and the movement of the particles only depends on the local flow velocity, or there is a two-way force interaction between particles and fluid. The particle Reynolds number is given by $Re_p = v_r \sigma_p / \nu$, where σ_p is the diameter of the particle, v_r is the relative velocity magnitude between the flow and the particle, and ν is the kinematic viscosity. For small and heavy particles with vanishing particle Reynolds number, linear Stokes drag can be assumed, cf. Siewert, Kunnen, and Schröder (2014). With these restrictions the equations of particle motion (Maxey and Riley 1983; Kunnen et al. 2013) are given by

$$\mathbf{v}_p = \frac{\mathrm{d}\mathbf{x}_p}{\mathrm{d}t} \tag{22}$$

$$\mathbf{a}_p = \frac{\mathrm{d}\mathbf{v}_p}{\mathrm{d}t} = \frac{1}{\tau_p} \cdot \left(\mathbf{v} - \mathbf{v}_p \right) + \mathbf{g}, \tag{23}$$

where $\mathbf{x}_p$ is the particle position, $\mathbf{v}_p$ is the particle velocity, and $\mathbf{g}$ is the gravitational force. The particle response time is given by

$$\tau_p = \frac{\sigma_p^2}{18\nu} \cdot \tilde{\rho}, \tag{24}$$

using $\tilde{\rho} = \rho_p / \rho$ as the ratio of the particle density ρ_p to the fluid density ρ. This yields the corresponding Stokes number $St = \tau_p \cdot v_b / \sigma_p$ with the bulk velocity v_b. The discrete time integration is performed using the trapezoidal rule. The particle position $\mathbf{x}_p$ is interpolated using a linear least squares interpolation of third-order accuracy. To solve the discrete set of particle equations, a predictor/corrector method using an Adams–Bashforth and an Adams–Moulton scheme (Hairer, Nørsett, and Wanner 1993) is used.

To determine collisions with surfaces, cells intersecting the geometry are marked as boundary cells in a pre-processing step. If the particle position $\mathbf{x}_p$ is inside such a boundary cell or in a cell neighbouring a boundary cell, an intersection test is performed. In case of an intersection, the collision is added to a collision list and collectively written to disk in a previously defined iteration step interval. Additionally, the corresponding particle is removed from the simulation. For more information on the Lagrange particle tracker, the reader is referred to Kunnen et al. (2013), Lintermann and Schröder (2017b), and Schneiders, Meinke, and Schröder (2017b).

3. Results

To substantiate ZFS's capabilities, performance and multi-physics simulation results are presented in the following. In Section 3.1, the performance of the massively parallel grid generator is discussed and in Section 3.2 the advantages of using parallelised geometries with a LB method for respiratory flows are shown. Next, in Section 3.3 some technical applications using the FV method coupled to the level set solver for the simulation of moving boundaries are presented. Finally, in Section 3.4 a coupled DG / FV approach for the simulation of aeroacoustics is discussed. The simulations and the performance measurements are performed on the JUQUEEN supercomputer (Appendix A.1.4), the JURECA supercomputer (Appendix A.1.5), the JULIA pilot system (Appendix A.1.6), and a local GPU cluster (Appendix A.1.7) located at Jülich Supercomputing Centre (JSC), Forschungszentrum Jülich. Furthermore, the HERMIT (Appendix A.1.1), HORNET (Appendix A.1.2), and HAZEL HEN (Appendix A.1.3) systems of HLRS are employed.

3.1. Performance of the massively parallel grid generator

Two strong scaling experiments E_1 and E_2 for the massively parallel grid generator are performed on a cubic domain (Lintermann et al. 2014). In E_1, a number of cells of $9.8 \cdot 10^9$ is generated, while in E_2 the number

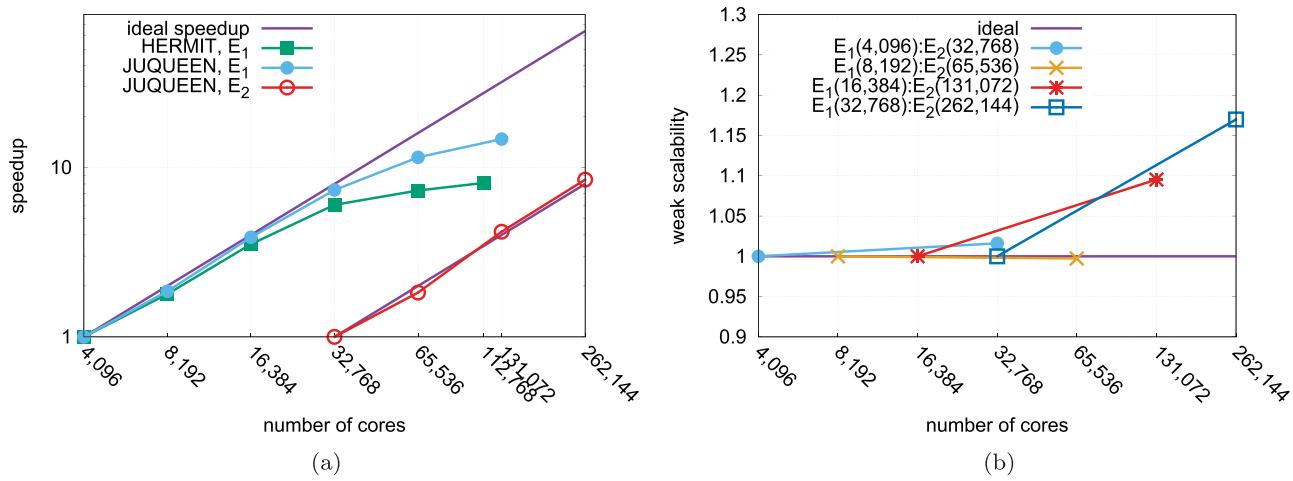

Figure 3. Performance of the massively parallel grid generator on HERMIT and JUQUEEN (Lintermann et al. 2014). The first experiment E_1 uses a grid size of $9.8 \cdot 10^9$ and the second experiment E_2 a grid of size of $78.54 \cdot 10^9$. (a) Strong scalability of the massively parallel grid generator and (b) weak scalability of the massively parallel grid generator on JUQUEEN as obtained from the strong scalability tests.

of cells sums up to $78.54 \cdot 10^9$. Case E_1 starts at level $l_\alpha = 7$ and is refined up to $l_\beta = 11$. It is run on both HERMIT at HLRS and JUQUEEN at JSC. Case E_2 has a start level of $l_\alpha = 7$, is refined up to level $l_\beta = 12$, and is only run on JUQUEEN. Note that for E_2 a minimum number of cores of 32, 768 is necessary to fulfil the local memory requirements. Figure 3(a) shows the results of the scaling experiments. The grid generation for E_1 perfectly scales up to 32,768 cores on both machines. For higher core numbers, the performance drop is due to the overhead of the serial part of the mesh generation, i.e. the base level already contains 8^7 cells. However, considering absolute wall times, the grid generation on the full HERMIT only requires 6.23 s and on a quarter of the JUQUEEN only 27.63 s, i.e. the generation is with a factor of 8.26 for 4096 cores and a factor of 4.44 for the maximum number of cores used in this experiment much faster on the HERMIT. For case E_2, a perfect speedup from 32, 768 up to the full JUQUEEN is visible and the results even show a slightly superlinear increase, which is due to the parallel job configuration, strongly machine dependent, and possibly due to cache effects. The mesh is generated on the full JUQUEEN in only 47.21 s. Although the ratio of the number of elements in E_1 and E_2 is approx. 10 and the number of cores increases from the baseline case by a factor of 8, a weak scaling analysis reveals some additional performance insights on JUQUEEN. Figure 3(b) shows the weak scalability of the grid generator by grouping the times for the number of cores that correspond to approx. 10 times the number of cells for case E_2 compared to E_1 into individual lines $E_1 : E_2$. From these graphs, it is obvious that in general, although more cells are present in E_2, the performance of the grid generator increases with increasing number of cells. This substantiates that the code is especially tailored for huge mesh sizes using large-scale HPC systems. Finally, a grid consisting of $0.64 \cdot 10^{12}$ cells is generated on the entire HERMIT in only 267.99 s. Considering I/O performance and grid compression, measurements are performed on 10,000 HAZEL HEN cores. For a uniformly refined mesh from $l_\alpha = 6$ to $l_\beta = 11$ with $2.1 \cdot 10^9$ cells, the output time is 5 s and the grid consumes 2 GB of disk space. A second experiment with heavy boundary refinement from $l_\alpha = 9$ to $l_\gamma = 18$ and $300 \cdot 10^6$ cells requires only 5 s to write and only 400 MB of disk space. For more details on the performance of the grid generator, also on its I/O performance on Lustre file systems, the reader is referred to Lintermann et al. (2014).

3.2. Performance and application of the lattice-Boltzmann method and parallel geometries

First, the performance of the LB method is discussed. Subsequently, an application and the advantages of using parallel geometries are presented.

Figure 4(a) shows the results of a strong scaling experiment of the LB method. For this experiment, a mesh consisting of $1.23 \cdot 10^9$ cells for a cubic domain is generated containing levels $l \in \{7, \ldots, 10\}$. The simulations cover 100 LB iterations and use the D3Q19 discretisation scheme. They are carried out on 2^{10} up to 2^{18} JUQUEEN cores utilising 64 MPI ranks per

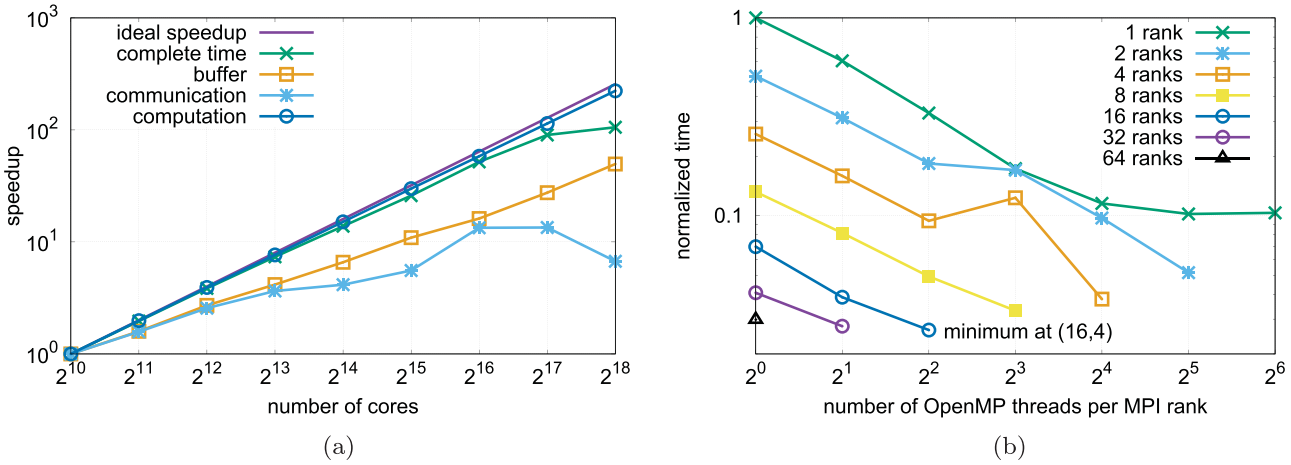

Figure 4. Strong scaling across multiple nodes (a) and non-dimensional compute time analysis on a single node (b) of the 3D LBGK flow solver using the D3Q19 discretisation scheme on the JUQUEEN at JSC. (a) Strong scaling results of the LB. The time is split into communication buffer setup, communication via MPI, and pure computation and (b) non-dimensional compute time of the LB method on a single node using different MPI/OpenMP combinations.

node. For the measurements, process-local times are recorded and averaged by the number of cores after the computation. The results of the base case at 2^{10} cores are used as reference to calculate the speedup of the code. The total time consumed by the iterations is split into the time required for the communication, setting up the communication buffer, and the pure computation time. From Figure 4(a), it is evident that the LB method scales up to 2^{18} cores. The slight performance drop from 2^{17} to 2^{18} cores is due to the small amount of cells per domain and the increased share of the communication time in the total computation time. Interestingly, the gradient of the speedup for the communication does not decrease continuously, but increases again from 2^{14} up to 2^{16} cores before it levels out from 2^{16} up to 2^{17} cores and then changes its sign. This can be explained by the job placement chosen by the job scheduler and by the network traffic caused by other running jobs. The speedup for setting up the communication buffer shows a linear, not optimal behaviour. The setup of the buffer for all measurement points is below 0.6% of the individual complete times with a decrease from 1.99 s on 2^{10} cores down to 0.04 s on 10^{18} cores. That is, minimal deviations in the time measurements might already have a great impact on the scalability. Furthermore, the setup of the communication buffer is a pure memory operation in which the information from the window cells corresponding to a neighbour $n \in \mathcal{N}(p)$ of a process p are copied to individual buffers. Since unstructured meshes are used, the source of this copy process may be quasi-randomly distributed in memory. Although an increase of the number of MPI ranks leads to a decrease of the overall number of cell information to be copied, the number of buffers increases with the number of MPI ranks. That is, for each neighbour and cell an additional entry of a 2D array, one dimension representing the neighbour number, the other holding the actual data to copy, needs to be touched. Since such memory operations are expensive and the considered timings are quite small, a loss in efficiency is visible. In contrast to the communication buffer, for the pure computation time an almost perfect scalability is visible up to 2^{18} cores.

To analyse the scalability of the LB method on Intel architectures, measurements are performed on the JURECA and HAZEL HEN systems. Therefore, a mesh consisting of $1.07 \cdot 10^9$ cells on refinement levels $l \in \{8, \ldots, 10\}$ is employed. For both systems, the minimum number of nodes that can be used is 16, i.e. the corresponding memory is sufficient to hold the mesh. The scalability analysis on JURECA is performed up to 3072 cores (128 nodes) and on HAZEL HEN up to 12, 288 cores (512 nodes). On both machines the number of spawned MPI ranks corresponds to the number of available physical cores, i.e. to 24 cores. Similar to the runs on JUQUEEN, the measurements are performed over 100 LB iterations of the D3Q19 model and averaged over several runs. Figure 5 shows the results of the scaling experiment. Obviously, the LB code scales well on both supercomputers. While the scalability for smaller core counts, i.e. on 384, 758, and 1536 cores, is

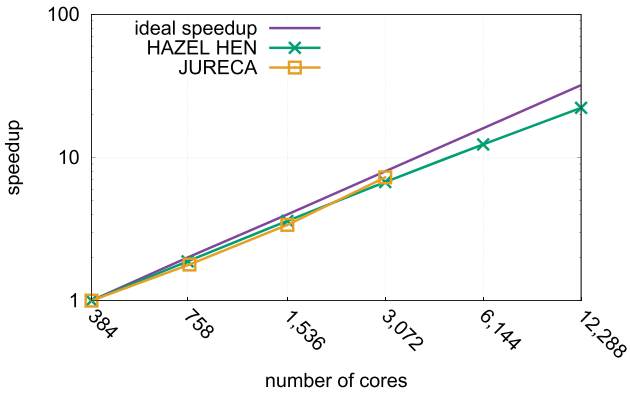

Figure 5. Strong scaling of the 3D LBGK flow solver using the D3Q19 discretisation scheme on the JURECA and HAZEL HEN at JSC and HLRS.

on the HAZEL HEN slightly better than on JURECA, the scalability on JURECA experiences a slight boost from 1536 to 3072 cores. The scalability on HAZEL HEN drops only marginally for core counts of 6144 and 12,288. To this end, the LB code scales on the JURECA up to 128 nodes with an efficiency of 91% and on HAZEL HEN up to 512 nodes with an efficiency of 70%.

For the analysis of the code performance using shared memory parallelisation with OpenMP, the D3Q19 LB method is tested on a single node of the JUQUEEN system, i.e. strong scalability experiments using $2.36 \cdot 10^6$ cells (levels $l \in \{6, 7\}$) are performed. Simulations cover 10 iterations of the LB, an increasing number of MPI ranks is tested, and for each of these cases a different number of OpenMP threads is spawned. Figure 4(b) shows the results of the experiments, where the total time has been normalised by the time required by one rank on one node using a single OpenMP thread. It is evident that the best performance is achieved using the MPI/OpenMP tuple $(16, 4)$. Considering that each IBM A2 processor consisted of 16 cores, each capable of four-way SMT, the results make sense. Comparing the results of $(16, 4)$ to those of $(32, 2)$ and $(64, 1)$ only a slight difference can be found, i.e. the absolute timings are 3.62 s, 3.79 s and 4.09 s for $(16, 4)$, $(32, 2)$, and $(64, 1)$. This yields corresponding non-dimensional timings of $2.64 \cdot 10^{-2}$, $2.76 \cdot 10^{-2}$, and $2.98 \cdot 10^{-2}$.

In addition, the LB method has been ported to the KNL system JULIA and the OpenMP performance has been measured by scaling with one MPI rank on one KNL node across all available 64 cores. Simulations are performed on a cubic mesh containing 8^8 cells, using the MCDRAM capability of the node, and employing different CPU affinities with `KMP_AFFINITY={scatter,compact}`. While the option `scatter` distributes the threads as evenly as possible across the entire system the option `compact` assigns the OpenMP thread $n+1$ to a free thread context as close as possible to the thread context where OpenMP thread n was placed. Furthermore, `KMP_HW_SUBSET=2T` is chosen, i.e. using two threads per core. It has been shown in Lintermann, Pleiter, and Schröder (2019) that setting `OMP_SCHEDULE=guided` delivers the best performance, which is why this option is additionally passed to the runs as run-time parameter. From Figure 6(a) it is obvious that using `KMP_AFFINITY=compact` performs best. In this case, a perfect scaling up to all available physical 64 cores is achieved before up to 128 threads, i.e. using hyperthreading, a marginal drop is visible. Using the combination `KMP_AFFINITY=scatter` leads to a similar behaviour up to 64 threads. The performance increase from 32 to 64 threads is, however, smaller compared to using `KMP_AFFINITY=compact`. Comparing the results to those obtained on JUQUEEN with the combination $(1, 64)$, cf. Figure 4(b), it is obvious that the scalability of the pure OpenMP parallelisation is better on the KNLs. The reason for this is manifold. The systems JUQUEEN and JULIA were consecutively installed at JSC, i.e. they feature different software stacks with different compiler, OpenMP, and library versions, which of course impact the code performance. ZFS and its parallelisation, single core performance, and memory access pattern have changed a lot over the last years. This also means that different versions of ZFS have been used on the two systems. The version that ran on JUQUEEN lacked a full OpenMP parallelisation, i.e. only the collision kernel was OpenMP parallelised, leaving some additional code executed in serial (per MPI rank), i.e. the propagation, the buffer setup, the communication, and the boundary condition kernels. In contrast, the runs on JULIA utilise a code with additional OpenMP parallelisation of the propagation kernel, the boundary condition kernels, the buffer setup, the interaction with the other modules, dynamic refinement, etc., reducing the serial part of the code and the effect of Amdahl's law. This leads to a better scalability. Furthermore, the runs on JUQUEEN used no specific

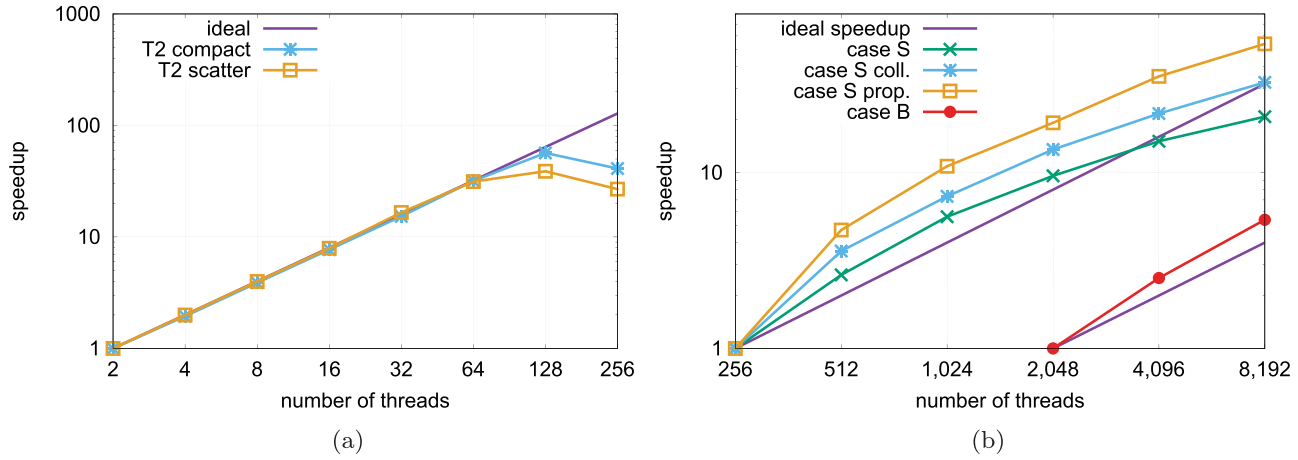

Figure 6. Scalability of the LBM on Intel Xeon Phi Knights Landing on JULIA and JURECA Booster module at JSC, Jülich: (a) scalability on a single JULIA KNL node and (b) scalability on JURECA Booster module.

OpenMP scheduling, i.e. the runs were performed in `OMP_SCHEDULE=static` mode. On JULIA, the option `OMP_SCHEDULE=guided` is enabled to enhance the scalability of the code. In addition, the newer architecture of the KNL with its 64 cores clocked at 1.3 GHz and its 16GB MCDRAM, which enables fast memory access, delivers 2662 GFLOPS. In contrast, the IBM A2 installed in JUQUEEN had 16 cores and delivered 204.8 GFLOPS at 1.6 GHz. Obviously, it is quite challenging to directly compare these systems based on code and OpenMP parallelisation performance. Nevertheless, it is assumed that the execution of the serial part of the code is faster on the KNL due to (i) the newer architecture and (ii) the MCDRAM utilisation. ZFS as a bandwidth-bound code obviously benefits from the latter.

To investigate the scalability on more than a single KNL node, performance measurements are performed on the JURECA Booster module using $2, \ldots, 64$ nodes. On each node, one MPI rank and 128 OpenMP threads are spawned. Two cases are considered, i.e. simulations are run for 100 iterations of the D3Q19 model on meshes $\mathcal{S}$ and $\mathcal{B}$ consisting of $158 \cdot 10^6$ and $1.23 \cdot 10^9$ cells with hierarchy levels $l_{\mathcal{S}} \in \{7, \ldots, 9\}$ and $l_{\mathcal{B}} \in \{7, \ldots, 10\}$. Note that the meshes fit into the MCDRAM of at least 2 nodes and that the maximum partition being allocated is 64 nodes. From Figure 6(b), it is clear that for case $\mathcal{S}$ the D3Q19 algorithm scales well up to 4096 threads (32 nodes) on the Booster with 94.4% efficiency. Subsequently, the performance drops, leading to an efficiency of 65.1% on 81,982 threads. Interestingly, on 512 and on 1024 threads a superlinear behaviour is visible. A more detailed analysis of the individual parts of the D3Q19 algorithm reveals that especially from 256 to 512 threads a massive speedup of the collision and propagation step is responsible for the gain in efficiency. From 2048 threads on, the communication time and the time to set up the buffer for the communication are more time-consuming and are responsible for the drop in efficiency. For case $\mathcal{B}$, a similar speedup increase behaviour as for case $\mathcal{S}$ is visible from 2048 to 8192 threads. This yields an efficiency of the D3Q19 algorithm of 134.81% on 8192 threads. Due to the node number restriction on the JURECA Booster, it is at present not possible to analyse the values for higher thread numbers to determine if a similar efficiency loss caused by increased communication and buffer setup appears.

As an application for the parallel geometry, a flow computation in the whole respiratory tract including the nasal cavity, the pharynx, larynx, trachea, and the lung down to the 12th bifurcation generation is used. Memory consumptions using a non-parallel geometry G_s^L and a parallel geometry G_p^L are performed on the JUQUEEN on $2^{13}, \ldots, 2^{17}$ cores. The original geometry consumes approx. 408 MB in memory. Figure 7(a) shows the memory consumption $\mathcal{M}_{s,p}^L$ for G_s^L and G_p^L, where the left ordinate is associated with G_s^L and the right with G_p^L. The amount of memory consumed by the serial geometry stays constant across all MPI ranks, i.e. the total allocated memory is $\mathcal{M}_{s,g}^L(2^{13}) \approx 2,780.7$ GB, the corresponding ADT uses $\mathcal{M}_{s.adt}^L(2^{13}) \approx 481.3$ GB, in sum $\mathcal{M}_{s,\Sigma}^L(2^{13}) \approx 3.19$ TB. For 131,072 this leads to $\mathcal{M}_{s,\Sigma}^L(2^{17}) \approx 51$ TB.

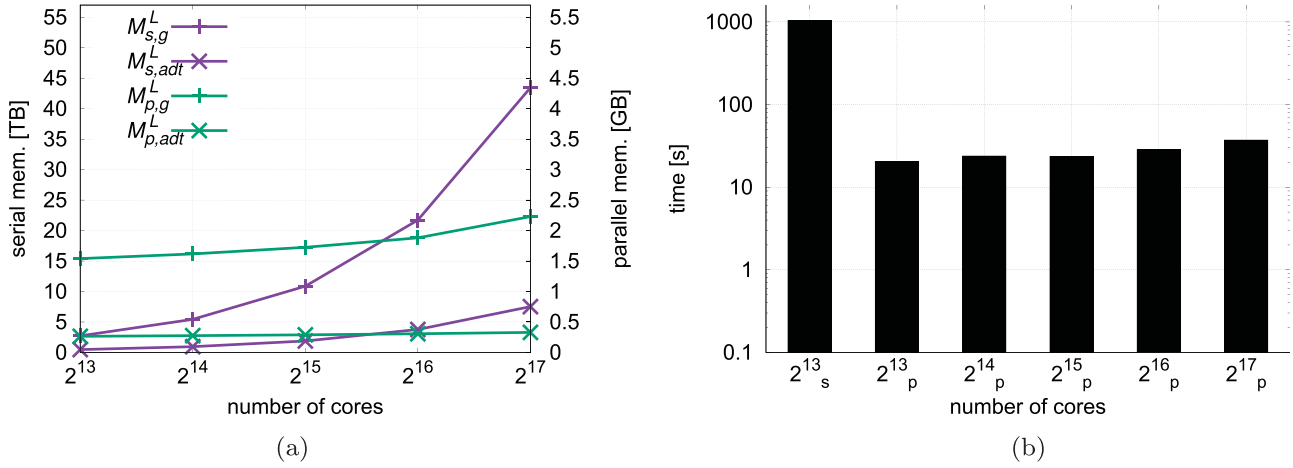

Figure 7. Memory consumption and preprocessing performance of the 3D LBGK flow solver employing the D3Q19 discretisation scheme on the JUQUEEN at JSC using serial and parallel geometries: (a) geometry and ADT memory consumption using serial (index *s*, left ordinate) and parallel (index *p*, right ordinate) geometries (Lintermann 2016) and (b) preprocessing performance using a serial (index *s*) and parallel (index *p*) geometry (Lintermann 2016).

In contrast, the total amount of consumed memory of the parallel geometry increases slightly from $\mathcal{M}^L_{p,g}(2^{13}) \approx 1.54\,\text{GB}$ for 2^{13} cores to $\mathcal{M}^L_{p,g}(2^{17}) \approx 2.28\,\text{GB}$ for 2^{17} cores. The same trend is visible for the ADT, with an increase from $\mathcal{M}^L_{p,adt}(2^{13}) \approx 0.27\,\text{GB}$ to $\mathcal{M}^L_{p,adt}(2^{17}) \approx 0.34\,\text{GB}$. Hence, the sum for all core number also increases from $\mathcal{M}^L_{p,\Sigma}(2^{13}) \approx 1.81\,\text{GB}$ to $\mathcal{M}^L_{p,\Sigma}(2^{13}) \approx 2.62\,\text{GB}$. To summarise, parallel geometries allow to reduce the overall memory consumption by a factor of ≈1802 and 19,936 for 2^{13} and 2^{17} cores, or in other words the memory consumption per process drops from 408 MB down to roughly 0.2 MB and 0.03 MB averaged over all processes for the minimum and maximum number of processors. That is, way more memory is available for the simulation. The parallelised geometry furthermore speeds up the pre-processing performance of the LB method, since each process only needs to read a fraction of the geometry and many operations such as wall-distance calculations for the interpolated bounce back (see Section 2.2.2) only need to operate on a small ADT tree. Figure 7(b) shows the performance gain for core numbers $2^{13}, \ldots, 2^{17}$. The left-most bar represents the serial case and requires 1045 s for pre-processing. In contrast, the second bar is with 20.5 s around 51 times faster. Even on 2^{17} cores the speedup is 28.

Ultimately, Figure 8 shows the result of a computation using the parallel geometry on 2^{15} cores, i.e. using 16 MPI ranks per node and 4 OpenMP threads per core. The computational mesh is generated from a geometry of the whole respiratory tract and consists of $\approx 2 \cdot 10^9$ cells. The flow is coloured by the velocity magnitude. At the nostrils, 10,000 Lagrangian particles with a diameter of $\sigma_p = 2.5\ \mu m$ and a density ratio of particle to air density of $\tilde{\rho} = 800$ are released and traced down the airway. The particle properties correspond to those of fine dust. The inset in the right corner shows the particle distribution at the pharynx region. The simulations are performed using the coupling method described in Lintermann and Schröder (2017b). That is, first an LB time step is executed before the particle movement is computed. For each particle movement, the predictor/corrector method described in Section 2.2.5 is performed. The position of the particle and the cell it is living in are continuously tracked and the fluid velocity in Equation (23) is trilinearly interpolated from the neighbouring cells. The accuracy of the temporal integration is guaranteed by an adaptive time step procedure that prevents particle velocities larger than the speed of sound c_s and particle displacements larger than the local grid distance $\Xi_i \Delta t$. All data exchange is handled by the superordinate coupling class (see Section 2.1.3). For more details on these simulations, the reader is referred to Lintermann (2016); Lintermann, Habbinga, and Göbbert (2017). Additional scaling results on ARMv7 clusters can furthermore be found in Lintermann, Pleiter, and Schröder (2019).

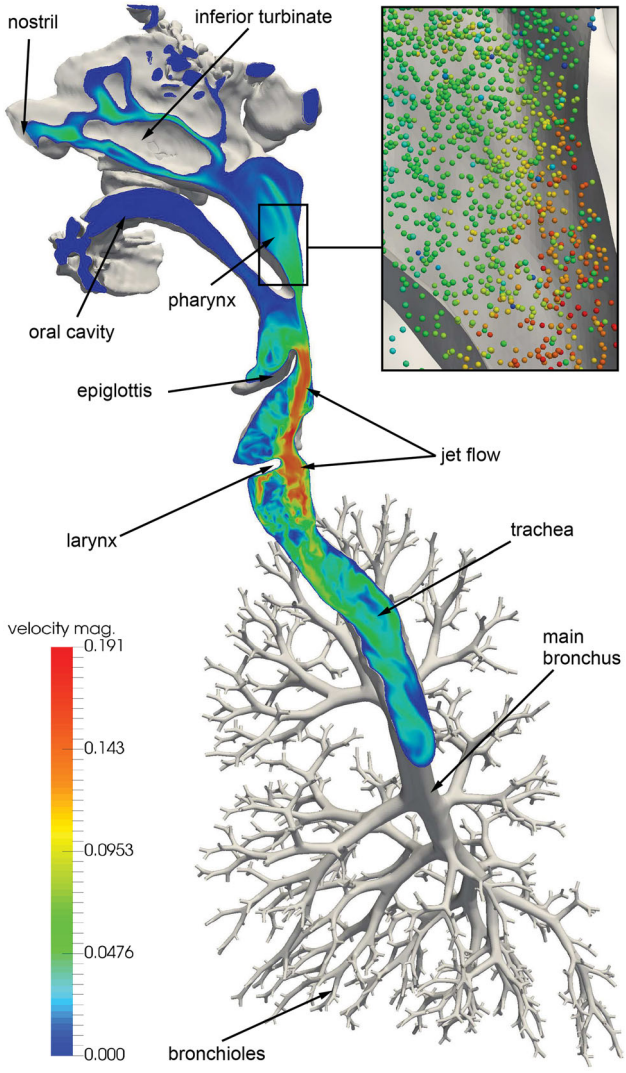

Figure 8. Simulation of the flow and particle dynamics in the whole respiratory tract (Lintermann 2016; Lintermann, Habbinga, and Göbbert 2017).

3.3. Finite-volume and moving boundary computations with dynamic load-balancing and parallel geometry

To analyse the parallel performance of the FV method, different scaling experiments are performed. For a strong scaling experiment on HAZEL HEN utilising a pure MPI parallelisation, i.e. using 24 cores per node, a computational mesh with 10^9 cells for a fan configuration is used.

For the scaling results in Figure 9, a 72° sector covering a full blade of the fan shown in Figure 9(c) is considered and LES computations are performed using 1000 time steps. Periodic boundary conditions are imposed on the azimuthal faces of the sector and the movement of the outer housing is enforced by the corresponding wall boundary condition in the rotating frame of reference. Measurements are employed on 5472, 10,968, 21,936, 32,880, and 91,872 cores using 24 cores per node. Obviously, with 86% efficiency when running on approximately 91,872 cores, the results are very satisfactory, cf. Pogorelov, Meinke, and Schröder (2015) and Cetin et al. (2016). Additionally, production runs employ multiple level sets (Günther, Meinke, and Schröder 2014) to track the movement of the fan mounted downstream of the stationary turbulence generating grid, cf. Pogorelov et al. (2018). The setup is a standardised fan test rig, which has been used for acoustic measurements in Sturm and Carolus (2012). It consists of a low pressure rotor-only fan with five twisted NACA 0010-63 airfoil blades for which six full 360° rotations are calculated. Unlike the scaling runs, the computational mesh of the production runs consists of $\approx 275 \cdot 10^6$ cells, employs the full configuration as shown in Figure 9(c) plus a turbulence generating grid, and adapts according to the location of the turbulence generating grid and the NACA blades. For the simulation, 4608 processes were employed for a total time of 340 h. The colours in Figure 9(c) represent the MPI ranks of the parallel geometry of the fan configuration analysed on the JUQUEEN system. The distribution for 8192 cores is shown. The geometry has been partitioned in accordance to the corresponding Hilbert-curve- and weight-based mesh partition, i.e. first the mesh is partitioned and then the elements of the geometry per MPI rank are determined. The serial geometry consumes 779 MB in memory and contains $3.32 \cdot 10^6$ triangles. Following the analysis of the parallelised geometry in Section 3.2, the plot in Figure 10 shows the memory consumption $\mathcal{M}^F_{s,p}$ for the serial geometry G^F_s and the parallel geometry G^F_p, where the left ordinate is associated with G^F_s and the right with G^F_p. The number of cores on the JUQUEEN system is varied from 2^{13} to 2^{17} cores. Again, the amount of memory consumed by the serial geometry is constant across all MPI ranks, i.e. the total allocated memory is $\mathcal{M}^F_{s,g}(2^{13}) \approx 5,318.3$ GB, the corresponding ADT uses $\mathcal{M}^F_{s,adt}(2^{13}) \approx 912.5$ GB, yielding in total $\mathcal{M}^F_{s,\Sigma}(2^{13}) \approx 6.14$ TB. That is, on 2^{17} cores this leads to $\mathcal{M}^F_{s,\Sigma}(2^{17}) \approx 98$ TB. The development of the green line in Figure 10 emphasises the memory advantage of the parallel geometry, i.e. the memory consumption increases from $\mathcal{M}^F_{p,g}(2^{13}) \approx 2.31$ GB for 2^{13} cores to $\mathcal{M}^F_{p,g}(2^{17}) \approx 3.34$ GB for 2^{17} cores. For the ADT,

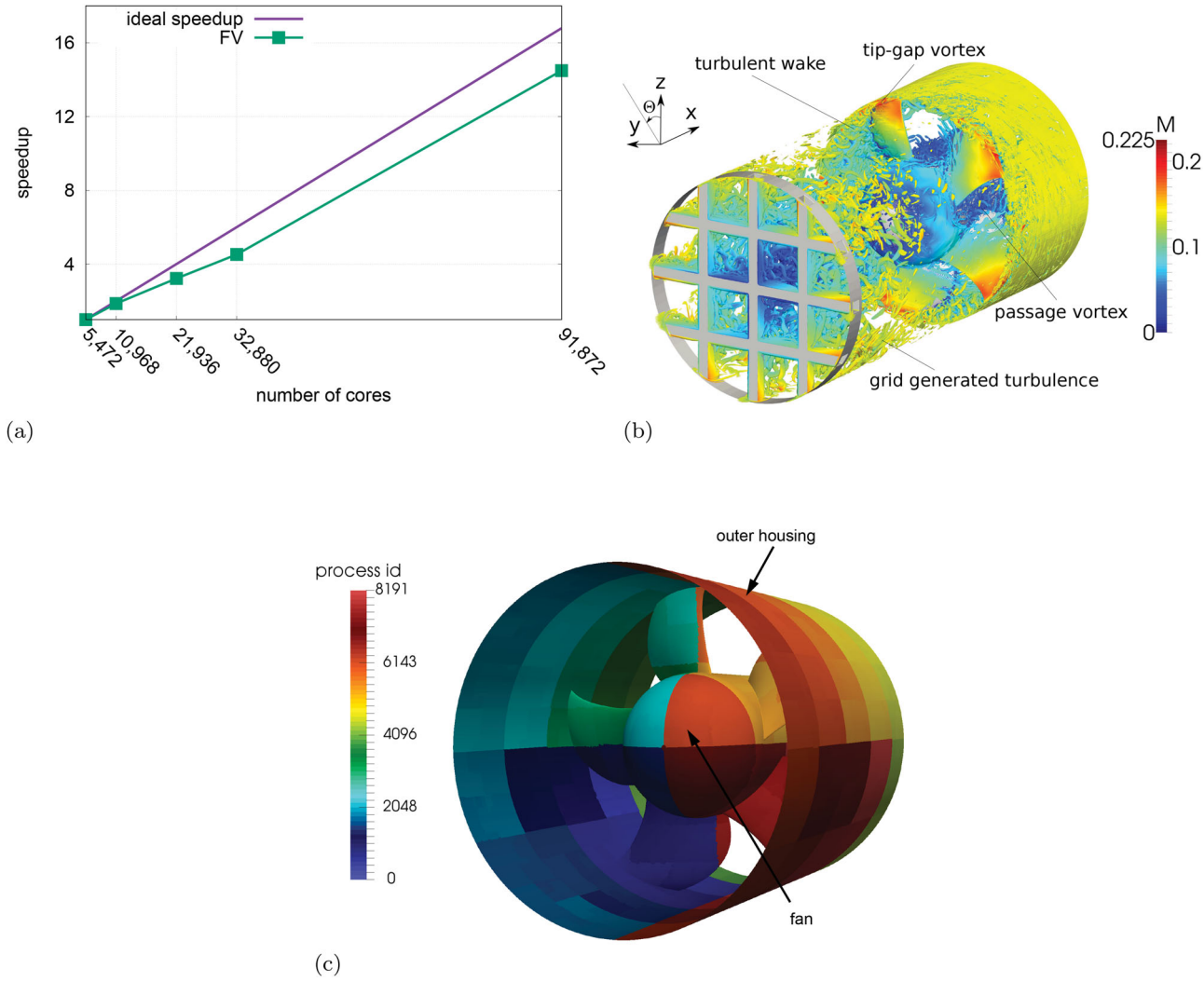

Figure 9. Parallel performance of the FV method on HAZEL HEN at HLRS and physical results for a fan configuration. (a) Strong scaling results of the FV method on the HAZEL HEN system at HLRS (Pogorelov, Meinke, and Schröder 2015; Cetin et al. 2016). (b) Simulation of turbulent flow in a fan configuration. Contours of the Q-criterion of the instantaneous flow field coloured by the relative Mach number (Pogorelov et al. 2018) and (c) parallelised geometry of a fan configuration. The geometry is subdivided into 8192 subunits.

the same behaviour is observed. An increase from $\mathcal{M}^F_{p,adt}(2^{13}) \approx 0.26\,\text{GB}$ to $\mathcal{M}^F_{p,adt}(2^{17}) \approx 0.33\,\text{GB}$ is visible. In total, the sum for all core numbers increases from $\mathcal{M}^F_{p,\Sigma}(2^{13}) \approx 2.57\,\text{GB}$ to $\mathcal{M}^F_{p,\Sigma}(2^{13}) \approx 3.67\,\text{GB}$. Averaged across all cores, the memory footprint per core can be reduced from 779 MB to 0.32 MB and 0.03 MB for 2^{13} and 2^{17} cores. Note that for a moving boundary simulation it is not necessary to move triangles from one process to another since the interface is solely tracked by level sets, which are generated once at the very beginning of the simulation.

In Figure 9, contours of the Q-criterion coloured by the relative Mach number in the frame of reference of the fan blades are shown. The turbulent secondary flow phenomena generated by the turbulence generating mesh decay downstream of the mesh due to mixing, leading to a homogenisation of the flow. The interaction of the flow with the rotating blades and the hub of the fan leads to passage vortices near the hub and to the development of a turbulent boundary layer. At the tips of the blades, where also the maximum relative Mach number is found, tip-gap vortices generate highly turbulent wakes.

Figure 11(a) shows the results of a weak scaling experiment performed on HAZEL HEN for a direct particle-fluid simulation (DPFS) of 6400 spherical particles in decaying homogeneous isotropic turbulence, cf. Schneiders et al. (2016). Corresponding physical results are shown in Figure 11(b). Two measurement series are performed, i.e. particle dynamics in decaying

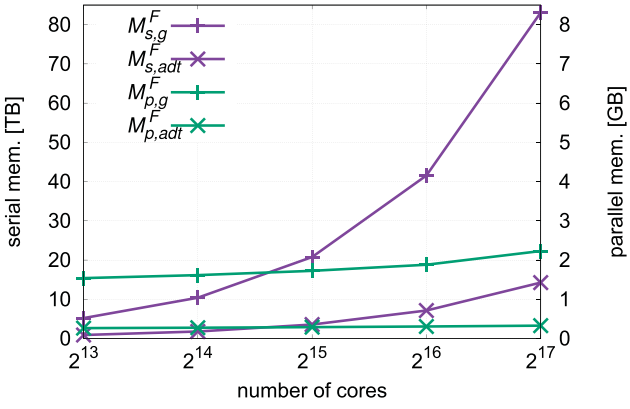

Figure 10. Geometry and ADT memory consumption using serial (index *s*, left ordinate) and parallel (index *p*, right ordinate) geometries for a FV simulation of a fan.

turbulence using fully resolved particles and simulations of only single-phase flow of the same configuration are performed. For the experiments the number of cells is kept constant at 2^{17} per core and execution times are recorded for a single time step on 128, 432, 1024, 3456, and 8192 cores. The wall time for the single-phase configuration increases only marginally with the number of cores and amounts to an excellent parallel efficiency of 98% at 8192 cores. Considering the particle-laden case, the overall number of particles and their diameters is kept constant such that the physical problem remains the same. The baseline case at 128 cores uses 256^3 cells. The corresponding computational overhead for the particle tracking and

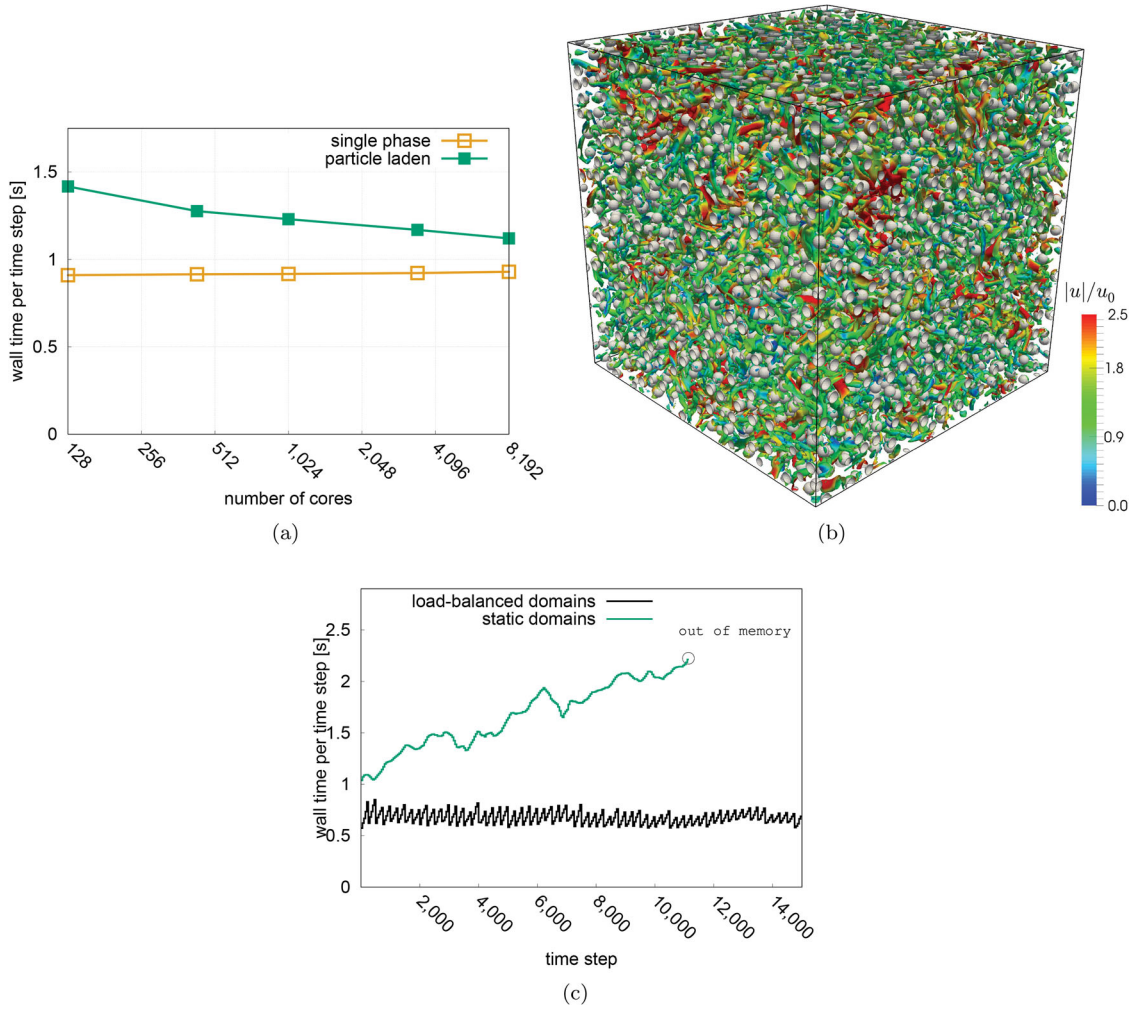

Figure 11. Weak scaling, physical and dynamic load-balancing results for the simulation of single-phase and particle-laden decaying isotropic turbulence: (a) weak scaling results for single-phase and particle-laden decaying isotropic turbulence using 2^{17} cells per core (Schneiders et al. 2016). (b) Simulation of 6400 fully resolved spherical particles in decaying homogeneous isotropic turbulence. Contours of the *Q*-criterion are coloured by the velocity magnitude (Schneiders et al. 2016) and (c) computational time as a function of the time step. Mesh adaptation performed on static domains results in steady increases of costs. Dynamic load-balancing avoids this problem (Schneiders et al. 2015).

solver reinitialisation amounts to 38%. Obviously, the time difference to the single phase computation is due to the imbalance invoked by the initially random particle locations. However, the increasing mesh size leads to a reduction of the gap between the single-phase and particle-laden cases and the overhead successively decreases to 22% for a number of cells of 1024^3. At each particle surface, a no-slip boundary condition is prescribed using a cut-cell method, see Section 2.2.1. That is, MPI ranks containing more particles require more computation to set the corresponding boundary conditions compared to ranks with less or no particles. Increasing the resolution for a fixed number of particles shortens the gap between single-phase and particle-laden timings. This allegedly atypical behaviour is explainable by the decrease of the ratio of cut cells to regular grid cells under increasing resolution.

To overcome an imbalance introduced by an uneven distribution of particles and hence an imbalance of the computational mesh, dynamic load-balancing is used to restore load balance (Schneiders et al. 2015); cf. Section 2.1.2. Figure 11(c) shows the time required to execute a single time step between the dynamic load-balancing method and a static uniform domain decomposition, i.e. the wall time per time step is plotted over the number of FV time steps. A static domain distribution suffers from load imbalance and becomes increasingly slower and finally one process runs out of memory. In contrast, using a dynamic load-balancing strategy at an interval of 250 time steps leads to a sawtooth-like curve with almost constant mean work load preventing out-of memory errors. Furthermore, after 10^4 time steps, the new scheme is roughly 300% faster rendering the total overhead for load-balancing of 6% negligible. Note that unlike the unbalanced case, the load-balanced computation starts with an initial rebalancing and hence with a smaller wall time per time step. Such simulations are necessary to fundamentally understand pulverised coal and biomass combustion processes. The particle surfaces are tracked with a level set method, i.e. at each iteration the level set equation is solved and the cut-cell method is applied to yield a proper boundary condition prescription. Forces on the particle surface as well as forces acting on the fluid are calculated. This way, accurate results for the dynamics of spherical and non-spherical particles are generated. The investigations were extended to 45,000 spherical particles in Schneiders, Meinke, and Schröder (2017a) and showed the particles to absorb energy from large-scale phenomena of the carrier flow. In contrast, small-scale turbulent motion is determined by the inertial particle dynamics. The particles locally increase the level of dissipation, the transfer of momentum to the fluid, however, compensates this process. Simultaneously, the viscous dissipation rate of the bulk flow is attenuated. Non-spherical particles have been considered in Schneiders, Meinke, and Schröder (2017b) using $2 \cdot 10^9$ cells. The results have been compared to Lagrangian approaches and underline the inability of such methods to accurately predict interactions in high wave number regimes, i.e. the interaction of the particle boundary layers and wakes with the smallest flow scales.

A third example considers the flame surface developing at the exit of a slot burner, cf. Schlimpert et al. (2016) and Schlimpert, Meinke, and Schröder (2016). The evolution of the flame front employs the level set method by solving the G-equation, see Equation (20) and Section 2.2.4. To reduce the computational costs, the G-equation is only solved in a narrow band around the flame front. Within this narrow band, the G-cells are adaptively refined and the normal $\check{\mathbf{n}}$ and curvature $\check{\kappa}$ are approximated by second-order accurate finite differences in the narrow band. The mesh for the flow computation consists of approximately $7 \cdot 10^6$ cells. The investigations in Schlimpert et al. (2016) focus on an analysis of the influence of the shear-layer development on the flame-front wrinkling. Furthermore, hydrodynamic instabilities and their effect on the flame front amplitude and flame pocket generation are discussed. The flame in Figure 12 is coloured by the velocity magnitude. In the background the computational mesh is shown. The inset shows a close-up in the tip region where flame pockets develop.

Note that the coupling of the FV solution and the level sets is performed by the superordinate coupling class (see Section 2.1.3). In case of, e.g. strong-coupled fluid–structure interaction, the FV advancement alternates with the advancement of the level sets until convergence is reached. This includes a continuous exchange of forces acting on interfaces and a change in position of the corresponding interfaces leading to a reshaping of the cut cells and a new prescription of the boundary conditions at the interface. The data exchange is handled by the coupling class.

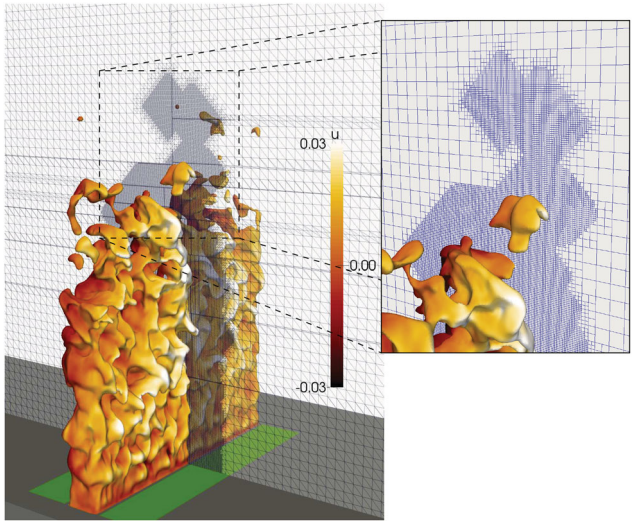

Figure 12. Tracking of a flame front by a level set representation. The isosurface of the flame front is coloured by the velocity magnitude. The inset shows a close-up (Schlimpert et al. 2016).

The FV method has also been ported to GPUs. Performance analyses are performed on up to 32 Tesla K40m GPUs (Kraus et al. 2014). For the measurements, simulations on a mesh with $\approx 2.5 \cdot 10^6$ cells are performed on the GPU cluster described in Appendix A.1.7. In Figure 13(a), the total time is split into the pure computational time and the time without I/O. Considering that the code uses unstructured meshes and is with its quasi-random memory access pattern rather bandwidth-bound than compute-bound, the main kernel loop scales with 88.27% efficiency quite well across all 32 GPUs. From Figure 13(a,b), the latter showing the share of the communication and the I/O of the total time in percent, it is obvious that communication kicks in at 8 GPUs when two nodes are employed. While the I/O times increase only slightly from 1 to 8 GPUs, a drastic increase is visible from 8 to 16 GPUs reducing the overall performance. The synchronous execution of the I/O and communication routines are responsible for this effect. Excluding I/O, the efficiency is roughly at 60% using 32 GPUs. Including I/O, the efficiency drops to roughly 40%. It should, however, be noted that the GPU cluster employed for the measurements does not feature a parallel file system and has no HPC interconnect and hence the performance drops are self-explanatory.

For more details on the applications presented in this section, the interested reader is referred to Schneiders et al. (2013, 2015, 2016), Meinke et al. (2013), Cetin et al. (2016), Pogorelov, Meinke, and Schröder (2015, 2016), Pogorelov et al. (2018), Schlimpert et al. (2015, 2016), Schlimpert, Meinke, and Schröder (2016), and Schneiders, Meinke, and Schröder (2017a, 2017b).

3.4. Aeroacoustics computations with a discontinuous Galerkin/finite-volume approach

Figure 14(a) shows the results of two strong scaling experiments performed on HAZEL HEN at HLRS Stuttgart and on JUQUEEN at JSC Jülich using 10^9 degrees of freedom and polynomial degrees $N = 3$ and $N = 7$, i.e. using $|C|_{N=3} \approx 16.8 \cdot 10^6$ and $|C|_{N=7} \approx 2.1 \cdot 10^6$ cells according to Equation (14). On HAZEL HEN, core numbers from 48 (2 nodes) up to 93,600 cores (3900 nodes) with 24 MPI ranks per node are employed for the measurements. JUQUEEN scalings start at 2048 cores and reach up to the whole machine. From Figure 14(a), it is evident that the DG solver scales with 98% and 80% almost perfectly on both machines. The results on JUQUEEN led to the High-Q club membership of ZFS in 2015.[2]

To perform coupled simulations, the results of an LES using the FV method are fed into the DG aeroacoustics method to solve the acoustic perturbation equations (APE), cf. Ewert and Schröder (2003), Schlottke et al. (2015), and Schlottke-Lakemper et al. (2016, 2017). Figure 14(a) also shows the scalability of a coupled and load-balanced simulation for a full production simulation of a turbulent jet. The computational mesh consists of $57.3 \cdot 10^6$ and $3.71 \cdot 10^6$ cells for the CFD and CAA parts (Niemöller et al. 2020). For the latter, the polynomial degree is chosen to be $N = 3$, which leads to $\approx 237.4 \cdot 10^6$ degrees of freedom. It is sufficient to couple the solvers in the source region, e.g. for the analysis of the aeroacoustics of a turbulent jet, the source region is placed in the vicinity of the potential core. To compute the aeroacoustic field via the DG solver, the LES field is first computed using the FV method. Then, the major source term of the APE-4 system, i.e. the linearised Lamb vector $\mathbf{L} = -(\boldsymbol{\omega} \times \mathbf{v})' = -(\boldsymbol{\omega}' \times \bar{\mathbf{v}} + \bar{\boldsymbol{\omega}} \times \mathbf{v}')$, where $\boldsymbol{\omega} = \nabla \times \mathbf{v}$ defines the vorticity vector, and $\langle \cdot \rangle$ and $\langle \cdot' \rangle$ denote mean and fluctuating quantities, is spatially and temporally interpolated from the LES solution. The LES is continuously advanced together with the aeroacoustic solution and the interpolation position and time is at every FV and DG time step known from

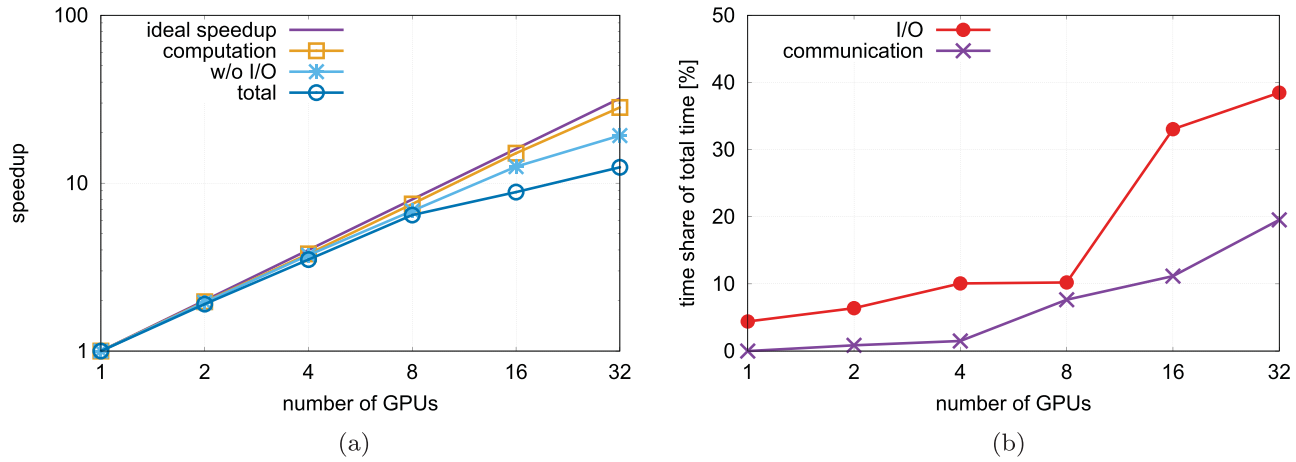

Figure 13. Parallel performance of the FV method on up to 32 Tesla K40m (Kraus et al. 2014): (a) strong scaling and (b) communication and I/O overhead.

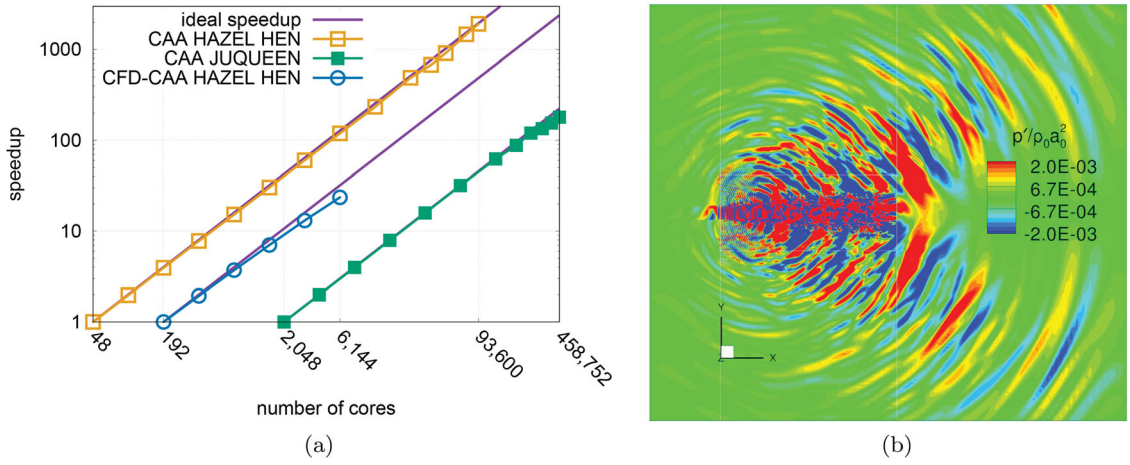

Figure 14. Results of a performance analysis of the DG and DG-FV-coupled methods, and of an aeroacoustic jet simulation. (a) Scalability of the standalone DG solver on HAZEL HEN at HLRS Stuttgart and on JUQUEEN at JSC Jülich (Schlottke-Lakemper et al. 2017), as well as of the coupled and load-balanced DG-FV method on HAZEL HEN (Niemöller et al. 2020) and (b) results of a coupled FV-DG aeroacoustic jet simulation (Cetin et al. 2015). Acoustic pressure contours are shown.

the mesh mapping in the unique octree and the individual time steps of the solutions. All data exchange is handled by the superordinate coupling class (see Section 2.1.3). From Figure 14(a), it becomes clear that the CFD-CAA-coupled and load-balanced simulation scales well up to 6144 cores on HAZEL HEN, where it achieves an efficiency of 74%.

To analyse acoustic fields, the non-dimensional pressure fluctuations $p'/(\rho_0 c_s^2)$ are usually considered. An example of an aeroacoustic jet simulation is shown in Figure 14(b). In general, it is the Lamb vector that is the main contributor to excited pressure waves (Ewert and Schröder 2003). Note that for such simulations the computational weights associated with the computational cells are chosen such that a balanced computation is achieved. That is, test computations are performed to calculate the ratio w_A/w_{FV} of an aeroacoustic and FV cell for various polynomial degrees and to assign the according weights for a balanced domain decomposition. The coupled simulation implements an interleaved execution of the solver steps to minimise waiting times between processes. More details on the efficient coupling strategy employed in the context of aeroacoustics simulations and on the cell weight ratio can be found in Schlottke et al. (2015) and Schlottke-Lakemper et al. (2016, 2017, 2019).

4. Summary and conclusion

In this work, the multi-physics simulation framework ZFS has been presented. The framework unites a whole simulation chain. It starts by processing a geometry file

as input for a massively parallel grid generator that can be employed for efficiently solving a variety of multi-physics problems using the solver base implemented in ZFS. Furthermore, parallelised geometries can be generated by this tool. With these inputs at hand quasi-incompressible flow in complex and intricate geometries can be computed by means of a lattice-Boltzmann solver. A finite-volume method is used to compute compressible flow for a whole spectrum of technical applications. A level set solver enables to track moving boundaries, even for arbitrarily shaped interfaces such as flame fronts. A discontinuous Galerkin solver is used to perform aeroacoustic simulations by coupling to a flow solver. With a Lagrangian approach particles can be tracked. ZFS furthermore features dynamic refinement and dynamic load-balancing. Processing of the simulation data can either be performed in-situ at compute time or in a post-processing step.

ZFS runs on various high-performance computers, has been ported to Xeon Phi Knights Landing and NVIDIA GPU accelerators with good scalability, and employs multi-level parallelisation with MPI and OpenMP. The individual solvers scale well across current top HPC systems such as the HAZEL HEN at HLRS Stuttgart and the JUQUEEN and JURECA systems at JSC in Jülich. In addition, the parallel grid generator enables to generate large-scale meshes in a short amount of time on a large number of processes, has a small memory footprint, and the number of processes used for the grid generation is independent from the number of processes employed for the computation. Parallelised geometries allow to massively reduce memory costs and preprocessing times for simulations. The introduction of computational weights for individual solvers and methods as well as in-memory coupling lead to balanced computations with reduced communication overhead, which is a bottleneck in co-simulation frameworks. Unlike in co-simulations, no management module needs to coordinate mesh and process coupling since the mesh hierarchy is shared between different solvers process locally, mesh adaptivity is complemented by dynamic load-balancing, and moving boundaries are tracked by global level sets.

All these features make ZFS a multi-purpose tool that efficiently solves a variety of complex interdisciplinary multi-physics problems. Its performance on state-of-the-art HPC systems makes it flexible in application complexity, in scaling problem sizes with only small efficiency loss, and in utilising heterogeneous architectures which characterise the way to exascale computing.

Notes

1. CODE_Saturne https://www.fz-juelich.de/ias/jsc/EN/Expertise/High-Q-Club/Code_Saturne/_node.html
2. High-Q Club http://www.fz-juelich.de/ias/jsc/EN/Expertise/High-Q-Club/_node.html

Acknowledgments

The authors would like to thank the authors and co-authors A. Adinetz, G. Bachmann-Harildstad, K. Beheng, S. Berger, G. Eitel-Amor, R. Ewert, A. Feldhusen, R.K. Freitas, O. Garyuk, J.H. Göbbert, J.H. Grimmen, C. Günther, S. Habbinga, D. Hartmann, S. Hemchandra, A. Henze, J. Kraus, E. Krause, R.P. Kunnen, F. Klemp, A. Nechyporenko, F. Peters, D. Pleiter, A. Pogorelov, T. Rister, B. Roidl, S. Schlimpert, M. Schlottke-Lakemper, L. Schneiders, C. Siewert, T. Soodt, and H. Yu. Furthermore, M. Albers, G. Brito Gadeschi, O. Cetin, H.-J. Chen, E. Fares, K. Fröhlich, S. Herff, P. Meysonnat, A. Niemöller, T. Schilden, K. Vogt, M. Waldmann, K.-D. Wernecke, and F. Wietbüscher of the papers (Meinke et al. 2002, 2013; Ewert and Schröder 2003; Freitas and Schröder 2008; Hartmann, Meinke, and Schröder 2008a, 2008b, 2008c, 2010, 2011; Eitel, Soodt, and Schröder 2010; Eitel et al. 2010; Freitas et al. 2011; Lintermann, Meinke, and Schröder 2011, 2012; Eitel-Amor, Meinke, and Schrder 2013; Kunnen et al. 2013; Lintermann, Meinke, and Schröder 2013; Lintermann et al. 2013, 2014; Schneiders et al. 2013; Günther, Meinke, and Schröder 2014; Kraus et al. 2014; Siewert, Kunnen, and Schröder 2014; Schlimpert et al. 2015; Pogorelov, Meinke, and Schröder 2015; Cetin et al. 2016; Pogorelov, Meinke, and Schröder 2016; Schlottke et al. 2015; Schneiders et al. 2015, 2016; Lintermann 2016; Schlimpert, Meinke, and Schröder 2016; Schlimpert et al. 2016; Schlottke-Lakemper et al. 2016; Lintermann, Habbinga, and Göbbert 2017; Lintermann and Schröder 2017a, 2017b; Schlottke-Lakemper et al. 2017; Schneiders, Meinke, and Schröder 2017a, 2017b; Pogorelov et al. 2018; Vogt et al. 2018; Waldmann et al. 2020; Lintermann, Pleiter, and Schröder 2019; Schlottke-Lakemper et al. 2019) who contributed with their work to the functionality and features of ZFS, and the results presented in this manuscript. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (GCS) (GCS, https://www.gauss-centre.eu), the Jülich Aachen Research Alliance Center for Simulation and Data Science (JARA-CSD) (JARA-CSD, https://www.jara.org/csd) Vergabegremium, and the Partnership for Advanced Supercomputing in Europe (PRACE) (PRACE, https://www.prace-ri.eu) for funding this project by providing computing time on the supercomputers JUQUEEN (Stephan and Docter 2015) and JURECA (Jülich Supercomputing Centre 2018) at Jülich Supercomputing Centre (JSC), and the HERMIT, HORNET and HAZEL HEN systems at the High-Performance Computing Center Stuttgart

(HLRS) through the John von Neumann Institute for Computing (NIC), the JARA-CSD partition and PRACE partitions. Furthermore, the authors thank the Human Brain Project and NVIDIA for providing the KNL-based system JULIA located at JSC Jülich and for GPU porting of ZFS.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

The research was funded by the German Research Foundation (Deutsche Forschungsgemeinsschaft e.V. - DFG) [grant numbers SCHR 309/63, SCHR 309/30, WE-2186/5, TRR 129 TP B2, SFB 686 TP C1] and the Federal Ministry of Economics and Technology (Bundesministerium für Wirtschaft und Technologie) [grant number AiF 17747 N].

ORCID

Andreas Lintermann http://orcid.org/0000-0003-3321-6599
Matthias Meinke http://orcid.org/0000-0003-4812-8495
Wolfgang Schröder http://orcid.org/0000-0002-3472-1813

References

Berger, M., and M. Aftosmis. 2012. "Progress Towards a Cartesian Cut-Cell Method for Viscous Compressible Flow." In *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, AIAA 2012-1301. Reston, Virigina: American Institute of Aeronautics and Astronautics. doi:10.2514/6.2012-1301.

Bhatnagar, P. L., E. P. Gross, and M. Krook. 1954. "A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems." *Physical Review* 94 (3): 511–525. doi:10.1103/PhysRev.94.511.

Bonet, J., and J. Peraire. 1991. "An Alternating Digital Tree (ADT) Algorithm for 3D Geometric Searching and Intersection Problems." *International Journal for Numerical Methods in Engineering* 31 (1): 1–17. doi:10.1002/nme.1620310102.

Boris, J. P., F. F. Grinstein, E. S. Oran, and R. L. Kolbe. 1992. "New Insights Into Large Eddy Simulation." *Fluid Dynamics Research* 10 (4–6): 199–228. doi:10.1016/0169-5983(92)90023-P.

Bouzidi, M., M. Firdaouss, and P. Lallemand. 2001. "Momentum Transfer of a Boltzmann-lattice Fluid with Boundaries." *Physics of Fluids* 13 (11): 3452–3459. doi:10.1063/1.1399290.

Bungartz, H.-J., F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, and B. Uekermann. 2016. "preCICE – A Fully Parallel Library for Multi-physics Surface Coupling." *Computers & Fluids* 141: 250–258. doi:10.1016/j.compfluid.2016.04.003.

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

Cetin, M. O., S. R. Koh, M. H. Meinke, and W. Schröder. 2015. "Aeroacoustic Analysis of a Helicopter Engine Jet Including a Realistic Nozzle Geometry". In *21st AIAA/CEAS Aeroacoustics Conference*, AIAA 2015-2533. Reston, Virginia: American Institute of Aeronautics and Astronautics. doi:10.2514/6.2015-2533.

Cetin, M. O., A. Pogorelov, A. Lintermann, H.-J. Cheng, M. Meinke, and W. Schröder. 2016. "Large-Scale Simulations of a Non-generic Helicopter Engine Nozzle and a Ducted Axial Fan." In *High Performance Computing in Science and Engineering '15*, edited by W. E. Nagel, D. H. Kröner, M. M. Resch, 389–405. Cham: Springer International Publishing. doi:10.1007/978-3-319-24633-8_25.

Childs, H., E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, et al. 2012. "VisIt: An End-User Tool for Visualizing and Analyzing Very Large Data." In High Performance Visualization–Enabling Extreme-Scale Scientific Insight, 357–372. Boca Raton.

D'Humières, D., I. Ginzburg, M. Krafczyk, P. Lallemand, and L.-S. Luo. 2002. "Multiple-relaxation-time Lattice Boltzmann Models in Three Dimensions." *Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences* 360 (1792): 437–51. doi:10.1098/rsta.2001.0955.

Dongarra, J., P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, et al. 2011. "The International Exascale Software Project Roadmap." *The International Journal of High Performance Computing Applications* 25 (1): 3–60. doi:10.1177/1094342010391989.

Duchaine, F., S. Berger, G. Staffelbach, and L. Gicquel. 2017. "Partitioned High Performance Code Coupling Applied to CFD." In *JHPCS 2016: High-Performance Scientific Computing, Lecture Notes in Computer Science Book Series (LNCS, Vol. 10164)*, edited by E. Di Napoli, M. A. Hermanns, H. Iliev, A. Lintermann, A. Peyser, 3–12. Cham, Aachen: Springer. doi:10.1007/978-3-319-53862-4_1.

Duchaine, F., S. Jauré, D. Poitou, E. Quémerais, G. Staffelbach, T. Morel, and L. Gicquel. 2015. "Analysis of High Performance Conjugate Heat Transfer with the OpenPALM Coupler." *Computational Science & Discovery* 8 (1): 015003. doi:10.1088/1749-4699/8/1/015003.

Dupuis, A., and B. Chopard. 2003. "Theory and Applications of An Alternative Lattice Boltzmann Grid Refinement Algorithm." *Physical Review E* 67 (6): 1–7. doi:10.1103/PhysRevE.67.066707.

Eicker, N., T. Lippert, T. Moschny, and E. Suarez. 2016. "The DEEP Project An Alternative Approach to Heterogeneous Cluster-computing in the Many-core Era." *Concurrency and Computation: Practice and Experience* 28 (8): 2394–2411. doi:10.1002/cpe.3562.

Eitel, G., R. K. Freitas, A. Lintermann, M. Meinke, and W. Schröder. 2010. "Numerical Simulation of Nasal Cavity Flow Based on a Lattice-Boltzmann Method." In *New Results in Numerical and Experimental Fluid Mechanics VII, Vol. 112 of Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, edited by A. Dillmann, G. Heller, M. Klaas, H.-P. Kreplin, W. Nitsche, W. Schröder, 513–520. Berlin/Heidelberg: Springer.

Eitel, G., T. Soodt, and W. Schröder. 2010. "Investigation of Pulsatile Flow in the Upper Human Airways." *International Journal of Design & Nature and Ecodynamics* 5 (4): 335–353. doi:10.2495/DNE-V5-N4-335-353.

Eitel-Amor, G., M. Meinke, and W. Schröder. 2013. "A Lattice-Boltzmann Method with Hierarchically Refined Meshes." *Computers & Fluids* 75: 127–139. doi:10.1016/j.compfluid.2013.01.013.

Ewert, R., and W. Schröder. 2003. "Acoustic Perturbation Equations Based on Flow Decomposition Via Source Filtering." *Journal of Computational Physics* 188 (2): 365–398. doi:10.1016/S0021-9991(03)00168-2.

Folk, M., and E. Pourmal. 2010. "Balancing performance and preservation lessons learned with HDF5." In *Proceedings of the 2010 Roadmap for Digital Preservation Interoperability Framework Workshop on – US-DPIF '10*, 1–8. doi:10.1145/2039274.2039285.

Fournier, Y., J. Bonelle, C. Moulinec, Z. Shang, A. Sunderland, and J. Uribe. 2011. "Optimizing Code_Saturne Computations on Petascale Systems." *Computers & Fluids* 45 (1): 103–108. doi:10.1016/j.compfluid.2011.01.028.

Freitas, R. K., A. Henze, M. Meinke, and W. Schröder. 2011. "Analysis of Lattice-Boltzmann Methods for Internal Flows." *Computers & Fluids* 47 (1): 115–121. doi:10.1016/j.compfluid.2011.02.019.

Freitas, R. K., and W. Schröder. 2008. "Numerical Investigation of the Three-dimensional Flow in a Human Lung Model." *Journal of Biomechanics* 41 (11): 2446–2457. doi:10.1016/j.jbiomech.2008.05.016.

Freund, J. B. 1997. "Proposed Inflow/Outflow Boundary Condition for Direct Computation of Aerodynamic Sound." *AIAA Journal* 35 (4): 740–742. doi:10.2514/2.167.

Frigo, M., and S. Johnson. 2005. "The Design and Implementation of FFTW3." *Proceedings of the IEEE* 93 (2): 216–231. doi:10.1109/JPROC.2004.840301.

Gaston, D., C. Newman, G. Hansen, and D. Lebrun-Grandié. 2009. "MOOSE: A Parallel Computational Framework for Coupled Systems of Nonlinear Equations." *Nuclear Engineering and Design* 239 (10): 1768–1778. doi:10.1016/j.nucengdes.2009.05.021.

Günther, C., M. Meinke, and W. Schröder. 2014. "A Flexible Level-set Approach for Tracking Multiple Interacting Interfaces in Embedded Boundary Methods." *Computers & Fluids* 102: 182–202. doi:10.1016/j.compfluid.2014.06.023.

Hänel, D. 2004. *Molekulare Gasdynamik, Einführung in die kinetische Theorie der Gase und Lattice-Boltzmann-Methoden*. Berlin Heidelberg: Springer-Verlag.

Hairer, E., S. P. Nørsett, and G. Wanner. 1993. *Solving Ordinary Differential Equations I: Nonstiff Problems*. 2nd ed. Berlin: Springer.

Hartmann, D., M. Meinke, and W. Schröder. 2008a. "An Adaptive Multilevel Multigrid Formulation for Cartesian Hierarchical Grid Methods." *Computers & Fluids* 37 (9): 1103–1125. doi:10.1016/j.compfluid.2007.06.007.

Hartmann, D., M. Meinke, and W. Schröder. 2008b. "Differential Equation Based Constrained Reinitialization for Level Set Methods." *Journal of Computational Physics* 227 (14): 6821–6845. doi:10.1016/j.jcp.2008.03.040.

Hartmann, D., M. Meinke, and W. Schröder. 2008c. "Erratum to "Differential Equation Based Constrained Reinitialization for Level Set Methods." *Journal of Computational Physics* 227: 6821–6845; *Journal of Computational Physics* 227 (22): 9696. doi:10.1016/j.jcp.2008.08.001.

Hartmann, D., M. Meinke, and W. Schröder. 2010. "The Constrained Reinitialization Equation for Level Set Methods." *Journal of Computational Physics* 229 (5): 1514–1535. doi:10.1016/j.jcp.2009.10.042.

Hartmann, D., M. Meinke, and W. Schröder. 2011. "A Level-set Based Adaptive-grid Method for Premixed Combustion." *Combustion and Flame* 158 (7): 1318–1339. doi:10.1016/j.combustflame.2010.11.007.

Henderson, A., J. Ahrens, and C. Law. 2004. "The ParaView Guide."

Heroux, M. A. 2009. "Software Challenges for Extreme Scale Computing: Going From Petascale to Exascale Systems." *The International Journal of High Performance Computing Applications* 23 (4): 437–439. doi:10.1177/1094342009347711.

Joppich, W., and M. Kürschner. 2006. "MpCCI – a Tool for the Simulation of Coupled Applications." *Concurrency and Computation: Practice and Experience* 18 (2): 183–192. doi:10.1002/cpe.913.

Jülich Supercomputing Centre. 2018. "JURECA: Modular Supercomputer At Jülich Supercomputing Centre." *Journal of Large-Scale Research Facilities* 4: A132. doi:10.17815/jlsrf-4-121-1.

Klimach, H., K. Jain, and S. Roller. 2014. "End-to-End Parallel Simulations with APES." In *Advances in Parallel Computing, Vol. 25: Parallel Computing: Accelerating Computational Science and Engineering (CSE)*, 703–711. IOS Press. doi:10.3233/978-1-61499-381-0-703.

Kopriva, D. A., S. L. Woodruff, and M. Y. Hussaini. 2000. "Discontinuous Spectral Element Approximation of Maxwell's Equations." In *Lecture Notes in Computational Science and Engineering*, 355–361. Vol. 11. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-59721-3_33.

Kraus, J., M. Schlottke, A. Adinetz, and D. Pleiter. 2014. "Accelerating a C++ CFD Code with OpenACC." In *2014 First Workshop on Accelerator Programming using Directives*, 47–54. IEEE. doi:10.1109/WACCPD.2014.11.

Kunnen, R. P., C. Siewert, M. Meinke, W. Schröder, and K. Beheng. 2013. "Numerically Determined Geometric Collision Kernels in Spatially Evolving Isotropic Turbulence Relevant for Droplets in Clouds." *Atmospheric Research* 127: 8–21. doi:10.1016/j.atmosres.2013.02.003.

Lallemand, P., and L.-S. Luo. 2000. "Theory of the Lattice Boltzmann Method: Dispersion, Dissipation, Isotropy, Galilean Invariance, and Stability." *Physical Review E* 61 (6): 6546–6562. doi:10.1103/PhysRevE.61.6546.

Li, J., M. Zingale, W.-k. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, and B. Gallagher. 2003. "Parallel netCDF: A High-Performance Scientific I/O Interface." In *Proceedings of the 2003 ACM/IEEE conference on*

Supercomputing – SC '03, 39. New York, NY: ACM Press. doi:10.1145/1048935.1050189.

Lieber, M., and W. E. Nagel. 2014. "Scalable High-Quality 1D Partitioning." In *2014 International Conference on High Performance Computing & Simulation (HPCS)*, 112–119. IEEE. doi:10.1109/HPCSim.2014.6903676.

Lintermann, A. 2016. "Efficient Parallel Geometry Distribution for the Simulation of Complex Flows." In *Proceedings of the VII European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS Congress 2016)*, edited by M. Papadrakakis, V. Papadopoulos, G. Stefanou, V. Plevris, 1277–1293. Greece, Athens: Institute of Structural Analysis and Antiseismic Research School of Civil Engineering National Technical University of Athens (NTUA). doi:10.7712/100016.1885.5067.

Lintermann, A., G. Eitel-Amor, M. Meinke, and W. Schröder. 2013. "Lattice-Boltzmann Solutions with Local Grid Refinement for Nasal Cavity Flows." In *New Results in Numerical and Experimental Fluid Mechanics VIII*, 583–590. Springer. doi:10.1007/978-3-642-35680-3_69.

Lintermann, A., S. Habbinga, and J. H. Göbbert. 2017. "Comprehensive Visualization of Large-Scale Simulation Data Linked to Respiratory Flow Computations on HPC Systems." In *SC '17: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. New York, NY: ACM. Denver, CO.

Lintermann, A., M. Meinke, and W. Schröder. 2011. "Investigations of the Inspiration and Heating Capability of the Human Nasal Cavity Based on a Lattice-Boltzmann Method." In *Proceedings of the ECCOMAS Thematic International Conference on Simulation and Modeling of Biological Flows (SIMBIO 2011)*. Brussels, Belgium.

Lintermann, A., M. Meinke, and W. Schröder. 2012. "Investigations of Nasal Cavity Flows based on a Lattice-Boltzmann Method." In *High Performance Computing on Vector Systems 2011*, edited by M. Resch, X. Wang, W. Bez, E. Focht, H. Kobayashi, S. Roller, 143–158. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-22244-3.

Lintermann, A., M. Meinke, and W. Schröder. 2013. "Fluid Mechanics Based Classification of the Respiratory Efficiency of Several Nasal Cavities." *Computers in Biology and Medicine* 43 (11): 1833–1852. doi:10.1016/j.compbiomed.2013.09.003.

Lintermann, A., D. Pleiter, and W. Schröder. 2019. "Performance of ODROID-MC1 for Scientific Flow Problems." *Future Generation Computer Systems* 95: 149–162. doi:10.1016/j.future.2018.12.059.

Lintermann, A., S. Schlimpert, J. Grimmen, C. Günther, M. Meinke, and W. Schröder. 2014. "Massively Parallel Grid Generation on HPC Systems." *Computer Methods in Applied Mechanics and Engineering* 277: 131–153. doi:10.1016/j.cma.2014.04.009.

Lintermann, A., and W. Schröder. 2017a. "A Hierarchical Numerical Journey Through the Nasal Cavity: From Nose-Like Models to Real Anatomies." *Flow, Turbulence and Combustion*. doi:10.1007/s10494-017-9876-0.

Lintermann, A., and W. Schröder. 2017b. "Simulation of Aerosol Particle Deposition in the Upper Human Tracheobronchial Tract." *European Journal of Mechanics – B/Fluids* 63: 73–89. doi:10.1016/j.euromechflu.2017.01.008.

Liou, M.-S., and C. J. Steffen. 1993. "A New Flux Splitting Scheme." *Journal of Computational Physics* 107 (1): 23–39. doi:10.1006/jcph.1993.1122.

Maxey, M. R., and J. J. Riley. 1983. "Equation of Motion for a Small Rigid Sphere in a Nonuniform Flow." *Physics of Fluids* 26: 883–889.

Meinke, M., L. Schneiders, C. Günther, and W. Schröder. 2013. "A Cut-cell Method for Sharp Moving Boundaries in Cartesian Grids." *Computers & Fluids* 85: 135–142. doi:10.1016/j.compfluid.2012.11.010.

Meinke, M., W. Schröder, E. Krause, and T. Rister. 2002. "A Comparison of Second- and Sixth-order Methods for Large-eddy Simulations." *Computers & Fluids* 31 (4–7): 695–718. doi:10.1016/S0045-7930(01)00073-1.

Niemöller, A., M. Schlottke-Lakemper, M. Meinke, and W. Schröder. 2020. "Dynamic Load Balancing for Direct-Coupled Multiphysics Simulations." *Computers & Fluids* 104437. doi:10.1016/j.compfluid.2020.104437.

Osher, S., and R. Fedkiw. 2003. *Level Set Methods and Dynamic Implicit Surfaces*, *Vol. 153 of Applied Mathematical Sciences*. New York, NY: Springer. doi:10.1007/b98879.

Peng, D., B. Merriman, S. Osher, H. Zhao, and M. Kang. 1999. "A PDE-Based Fast Local Level Set Method." *Journal of Computational Physics* 155 (2): 410–438. doi:10.1006/jcph.1999.6345.

Permann, C. J., D. R. Gaston, D. Andrs, R. W. Carlsen, F. Kong, A. D. Lindsay, J. M. Miller, et al. 2019. arXiv:1911.04488. http://arxiv.org/abs/1911.04488.

Pogorelov, A., M. Meinke, and W. Schröder. 2015. "Cut-cell Method Based Large-eddy Simulation of Tip-leakage Flow." *Physics of Fluids* 27 (7): 075106. doi:10.1063/1.4926515.

Pogorelov, A., M. Meinke, and W. Schröder. 2016. "Effects of Tip-gap Width on the Flow Field in An Axial Fan." *International Journal of Heat and Fluid Flow* 61: 466–481. doi:10.1016/j.ijheatfluidflow.2016.06.009.

Pogorelov, A., L. Schneiders, M. Meinke, and W. Schröder. 2018. "An Adaptive Cartesian Mesh Based Method to Simulate Turbulent Flows of Multiple Rotating Surfaces." *Flow, Turbulence and Combustion* 100 (1): 19–38. doi:10.1007/s10494-017-9827-9.

Qian, Y. H., D. D'Humières, and P. Lallemand. 1992. "Lattice BGK Models for Navier-Stokes Equation." *Europhysics Letters (EPL)* 17 (6): 479–484. doi:10.1209/0295-5075/17/6/001.

Sagan, H. 1994. *Space-Filling Curves*. 1st ed. New York, NY: Universitext, Springer New York. doi:10.1007/978-1-4612-0871-6.

Schlimpert, S., A. Feldhusen, J. H. Grimmen, B. Roidl, M. Meinke, and W. Schröder. 2016. "Hydrodynamic Instability and Shear Layer Effects in Turbulent Premixed Combustion." *Physics of Fluids* 28 (1): 017104. doi:10.1063/1.4940161.

Schlimpert, S., S. Hemchandra, M. Meinke, and W. Schröder. 2015. "Hydrodynamic Instability and Shear Layer Effect

on the Response of An Acoustically Excited Laminar Premixed Flame." *Combustion and Flame* 162 (2): 345–367. doi:10.1016/j.combustflame.2014.08.001.

Schlimpert, S., M. Meinke, and W. Schröder. 2016. "Nonlinear Analysis of An Acoustically Excited Laminar Premixed Flame." *Combustion and Flame* 163: 337–357. doi:10.1016/j.combustflame.2015.09.035.

Schlottke, M. A., H.-J. Cheng, A. Lintermann, M. H. Meinke, and W. Schröder. 2015. "A Direct-Hybrid Method for Computational Aeroacoustics." In *21st AIAA/CEAS Aeroacoustics Conference*, AIAA 2015–3133. Dallas, TX. doi:10.2514/6.2015-3133.

Schlottke-Lakemper, M., F. Klemp, H.-J. Cheng, A. Lintermann, M. Meinke, and W. Schröder. 2016. "CFD/CAA Simulations on HPC Systems." In *Sustained Simulation Performance 2016*, 139–157. Cham: Springer International Publishing. doi:10.1007/978-3-319-46735-1_12.

Schlottke-Lakemper, M., A. Niemöller, M. Meinke, and W. Schröder. 2019. "Efficient Parallelization for Volume-coupled Multiphysics Simulations on Hierarchical Cartesian Grids." *Computer Methods in Applied Mechanics and Engineering* 352: 461–487. doi:10.1016/j.cma.2019.04.032.

Schlottke-Lakemper, M., H. Yu, S. Berger, M. Meinke, and W. Schröder. 2017. "A Fully Coupled Hybrid Computational Aeroacoustics Method on Hierarchical Cartesian Meshes." *Computers & Fluids* 144: 137–153. doi:10.1016/j.compfluid.2016.12.001.

Schneiders, L., J. H. Grimmen, M. Meinke, and W. Schröder. 2015. "An Efficient Numerical Method for Fully-resolved Particle Simulations on High-performance Computers." *Proceedings in Applied Mathematics and Mechanics* 15 (1): 495–496. doi:10.1002/pamm.201510238.

Schneiders, L., C. Günther, M. Meinke, and W. Schröder. 2016. "An Efficient Conservative Cut-cell Method for Rigid Bodies Interacting with Viscous Compressible Flows." *Journal of Computational Physics* 311: 62–86. doi:10.1016/j.jcp.2016.01.026.

Schneiders, L., D. Hartmann, M. Meinke, and W. Schröder. 2013. "An Accurate Moving Boundary Formulation in Cut-cell Methods." *Journal of Computational Physics* 235: 786–809. doi:10.1016/j.jcp.2012.09.038.

Schneiders, L., M. Meinke, and W. Schröder. 2017a. "Direct Particle–fluid Simulation of Kolmogorov-length-scale Size Particles in Decaying Isotropic Turbulence." *Journal of Fluid Mechanics* 819: 188–227. doi:10.1017/jfm.2017.171.

Schneiders, L., M. Meinke, and W. Schröder. 2017b. "On the Accuracy of Lagrangian Point-mass Models for Heavy Non-spherical Particles in Isotropic Turbulence." *Fuel* 201: 2–14. doi:10.1016/j.fuel.2016.11.096.

Siewert, C., R. P. Kunnen, and W. Schröder. 2014. "Collision Rates of Small Ellipsoids Settling in Turbulence." *Journal of Fluid Mechanics* 758: 686–701. doi:10.1017/jfm.2014.554.

Stephan, M., and J. Docter. 2015. "JUQUEEN: IBM Blue Gene/Q^{6} Supercomputer System at the Jülich Supercomputing Centre." *Journal of Large-Scale Research Facilities JLSRF* 1: A1. doi:10.17815/jlsrf-1-18.

Sturm, M., and T. Carolus. 2012. "Tonal Fan Noise of an Isolated Axial Fan Rotor Due to Inhomogeneous Coherent Structures at the Intake." *Noise Control Engineering Journal* 60 (6): 699–706. doi:10.3397/1.3701041.

van Leer, B. 1979. "Towards the Ultimate Conservative Difference Scheme. V. A Second-order Sequel to Godunov's Method." *Journal of Computational Physics* 32 (1): 101–136. doi:10.1016/0021-9991(79)90145-1.

Vázquez, M., G. Houzeaux, S. Koric, A. Artigues, J. Aguado-Sierra, R. Arís, D. Mira, et al. 2016. "Alya: Multiphysics Engineering Simulation Toward Exascale." *Journal of Computational Science* 14: 15–27. doi:10.1016/j.jocs.2015.12.007.

Vogt, K., G. Bachmann-Harildstad, K.-D. Wernecke, O. Garyuk, A. Lintermann, A. Nechyporenko, and F. Peters. 2018. "The New Agreement of the International RIGA Consensus Conference on Nasal Airway Function Tests." *Rhinology* 56 (2): 133–143. doi:10.4193/Rhino17.084.

Waldmann, M., A. Lintermann, Y. J. Choi, and W. Schröder. 2020. "Analysis of the Effects of MARME Treatment on Respiratory Flow Using the Lattice-Boltzmann Method." In *New Results in Numerical and Experimental Fluid Mechanics XII*, 853–863. Darmstadt, Germany: Springer. doi:10.1007/978-3-030-25253-3_80.

Wolf, K., P. Bayrasy, C. Brodbeck, I. Kalmykov, A. Oeckerath, and N. Wirth. 2017. "MpCCI: Neutral Interfaces for Multiphysics Simulations." In *Scientific Computing and Algorithms in Industrial Simulations*, 135–151. Cham: Springer International Publishing. doi:10.1007/978-3-319-62458-7_7.

Yu, W., J. Vetter, R. S. Canon, and S. Jiang. 2007. "Exploiting Lustre File Joining for Effective Collective IO." In *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, 267–274. IEEE. doi:10.1109/CCGRID.2007.51.

Appendix

A.1 HPC system overview

This appendix describes the high-performance computing systems that were employed for the computation of the performance and application results in Section 3.

A.1.1 HERMIT

The CRAY HERMIT system was located at the High Performance Computing Center Stuttgart (HLRS) and consisted of 3552 nodes containing each two AMD Opteron 6276 (Interlagos) CPUs with 16 cores per CPU, clocked at 2.3 GHz. The system had a peak performance of 1.045 PFlops for 113,664 cores. 3072 nodes contained 32 GB of RAM, 480 nodes contained 64 GB of RAM. Parallel I/O is implemented via a Lustre File System (LFS) (Yu et al. 2007). The HERMIT system was first replaced by HORNET, which was upgraded to HAZEL HEN.

A.1.2 HORNET

The CRAY HORNET system was located at the HLRS and was the first expansion stage of its predecessor HAZEL HEN. For details on the node configuration, see Appendix A.1.3.

A.1.3 HAZEL HEN

The CRAY HAZEL HEN system is located at the High Performance Computing Center Stuttgart (HLRS) and consists of 7712 dual socket nodes containing each 2 Intel Haswell E5-2680v3 CPUs, each with 12 cores clocked at 2.5 GHz. The system has a peak performance of 7.4 PFlops for 185,088 cores. The nodes contain 128 GB of RAM. Parallel I/O is implemented via a Lustre File System (LFS) (Yu et al. 2007). The HAZEL HEN system replaces the HORNET system.

A.1.4 JUQUEEN

The JUQUEEN system (Stephan and Docter 2015) is located at the Jülich Supercomputing Centre (JSC), Forschungszentrum Jülich. The JUQUEEN is an IBM BlueGene/Q system and consists of 28,672 nodes containing IBM PowerPC A2 CPUs at 1.6 GHz, 16 cores, and 16 GB of RAM per node. The overall peak performance is 5.9 PFlops. Due to its four-way SMT hardware threaded floating point units it is capable of running a maximum number of four OpenMP threads per core. The JUQUEEN system uses the IBM LoadLeveler as job scheduler and has a 5D Torus network with a 40 GB/s bandwidth.

A.1.5 JURECA

The JURECA system (Jülich Supercomputing Centre 2018) is located at the Jülich Supercomputing Centre (JSC), Forschungszentrum Jülich. The JURECA has 1872 compute nodes, each equipped with 2 Intel Haswell Xeon E5-2680 v3 CPUs, each with 12 cores clocked at 2.5 GHz. In 75 compute nodes, two NVIDIA K80 GPUs are installed. The JURECA nodes are interconnected via a Mellanox EDR InfiniBand high-speed network with non-blocking fat tree topology. The Booster module of JURECA consists of 1640 compute nodes with Intel Xeon Phi 7250-F Knights Landing (KNL) CPUs. Each of the KNLs has 68 cores clocked at 1.4 GHz and 96 GB memory plus 16 GB MCDRAM high-bandwidth memory. The KNL nodes are connected via an Intel Omni-Path Architecture high-speed network with non-blocking fat tree topology.

A.1.6 JULIA

The JULIA system is one of the two pilot systems developed by CRAY in the pre-commercial procurement during the Human Brain Project ramp-up phase. It is located at Jülich Supercomputing Centre (JSC), Forschungszentrum Jülich. This pilot system is based on the CRAY CS-Storm architecture and comprises 60 compute nodes, each equipped with an Intel Xeon Phi Processor 7210 Knights Landing, KNL. Each KNL is clocked at 1.3 GHz, has 64 cores, allows 4 threads per core, and features 16 GB MCDRAM and 96 GB DDR4 memory. The nodes are interconnected in a tree topology using the new Intel OPA network technology. Additionally, the system comprises DataWarp and visualisation nodes.

A.1.7 GPU cluster

The GPU cluster used for the scaling measurements in Section 3.3 consists of eight nodes connected with FDR InfiniBand. Each node contains two Intel Xeon E5-2690 v2 clocked at 3.00 GHz. Every Xeon processor has 10 cores with disabled hyperthreading and turbo boost. Each node is equipped with 4 Tesla K40m devices running at 875 MHz and with ECC enabled.