ECCOMAS Congress 2016
VII European Congress on Computational Methods in Applied Sciences and Engineering
M. Papadrakakis, V. Papadopoulos, G. Stefanou, V. Plevris (eds.)
Crete Island, Greece, 5–10 June 2016

EFFICIENT PARALLEL GEOMETRY DISTRIBUTION FOR THE SIMULATION OF COMPLEX FLOWS

Andreas Lintermann[1]

[1]JARA-HPC, RWTH Aachen University
Schinkelstr. 2, 52062 Aachen, Germany
e-mail: Lintermann@jara.rwth-aachen.de

Keywords: Parallel Geometry, Lattice-Boltzmann Method, Respiratory Flow, Computational Fluid Dynamics

Abstract. *Highly resolved intrinsic geometrical shapes used in three-dimensional parallel simulations of fluid flows consume a large portion of the available memory when loaded serially on every process. This demands for a memory efficient implementation of a distributed geometry which is however a non-trivial task when complex spatial domain decomposition methods for the flow domain are involved. To overcome this problem, an algorithm to generate a parallel geometry during the mesh generation is proposed that enables a low-memory subdivision of the geometry based on the decomposition of the flow field. The applied meshing method generates computational grids that can be used for simulations on a quasi-arbitrary number of cores on which the geometry is distributed in an efficient preprocessing step. This allows reducing the number of instances of the geometry in the global memory of the simulation to about one.*
The algorithm is used to generate a parallel geometry for a large shape consisting of $7 \cdot 10^6$ *triangles, i.e., for a geometry representing the whole respiratory tract down to the* 12^{th} *lung generation. For this case, performance and memory consumption measurements are given for simulations on 8,192 up to 131,072 cores and juxtaposed to results obtained from simulations using non-parallel geometries. The findings show that with the new method not only the memory usage could be reduced by the factors of 1,802 and 19,936 for core numbers of 8,192 and 131,072 but also a large speed-up factor of about 51 is obtained in the geometry I/O and preprocessing. Furthermore, the parallel geometry allows using the sweet spot with respect to a combination of distributed and shared memory parallelization leading to an increase of the computational speed of about 1.43.*

1 INTRODUCTION

The three-dimensional parallel simulation of fluid flows in and around objects using non-boundary fitting unstructured meshes requires the geometrical representation of the involved object surfaces to set the boundary conditions and to generate the computational mesh. Such geometries often stem from *Computer Aided Design* (*CAD*) software [1, 2, 3], from *Computer Tomography* (*CT*) images [4, 5, 6], or from laser-scan reconstructions of real-world objects. The number of representing surface elements varies with the resolution of the geometrical model. Hence, for numerical simulations of flows using highly-accurate geometrical representations from, e.g., technical or biomechanical background, the corresponding surface data may exceed the process-local memory or consume a large share of it in parallel simulations. That is, loading the complete geometry for such simulations limits the memory available for solving for the flow variables.

Body-fitted structured meshes have the advantage of not requiring a geometrical representation for preprocessing of the simulation and are well suited to resolve boundary layer flows [7, 8, 9, 10, 11]. However, they necessitate the geometry for the often time consuming manual construction of the computational mesh. In contrast, the generation of hierarchical *Cartesian* meshes can be executed completely automatically in a short amount of time [12, 13, 14]. Such meshes are well suited for complex geometries and are used for a variety of applications [2, 3, 6, 15, 16, 17, 18] in which the computational grids are generated from a geometrical surface representation given in *STereo Lithography* (*STL*) format. Lintermann et al. [6] and Freitas et al. [16] use a *Lattice-Boltzmann Method* (*LBM*) to simulate the biofluidmechanics of the human nasal cavity and the human lung. They use the no-slip *BFL*-rule from Bouzidi et al. [19] as wall boundary condition which requires the availability of the geometry for the distance calculation of boundary cell centers to the wall surface in the discrete *LBM* directions. In [2, 17], a *Finite-Volume Method* (*FVM*) is used for the three-dimensional simulation of the flow in a combustion engine. The same method is applied for the simulation of tip-leakage flows in fan geometries [3] and turbulent helicopter engine jets [18]. In these publications, a *cut-cell* method [2, 20] based on the geometry is used to accurately apply wall boundary conditions. Furthermore, in [2, 15, 17] a *level-set* mesh that allows for an efficient tracking of moving boundaries is constructed from the geometry. That is, the geometry is an essential element for pre- and in-simulation processing.

The geometry in most of these cases is small and consumes only a marginal amount of memory, e.g., one of the nasal cavity geometries in [6] consumes around 210 MB in process memory. However, considering a *High Performance Computer* (*HPC*) like the IBM BlueGene/Q *JUQUEEN* [21] at the *Jülich Supercomputing Center* (*JSC*), which has 1 GB per computational core available, about 20% of the global process memory is occupied by the nasal cavity geometry and cannot be used for the computation. Future computations for the simulation of flows in complex and large geometries may hence experience memory issues that can only be avoided by a parallel implementation of the geometry or by using less cores per node together with a shared memory parallelization.

In this work, a new algorithm to generate a parallel geometry from a *STL* file during the mesh generation process is presented. It is based on the hierarchical *Cartesian* mesh generator introduced in [14] and avoids wasting memory by distributing the geometry corresponding to the applied domain decomposition. This not only reduces the total and local amount of memory occupied by the geometry, but also the time of pre-simulation processing. Furthermore, the computational time is reduced due to an optimal usage of distributed and shared memory parallelization. To the best of the author's knowledge, such an algorithm, especially for large-scale simulations applying complex spatial domain decomposition methods for the flow domain, has not been presented in the literature and may be beneficial for future simulations.

This paper is organized as follows. The numerical methods for the simulation of flows in intricate geometries, for the parallel grid generator, and for the generation of the parallel geometry are discussed in Sec. 2. Sec. 3 discusses the performance of the parallel geometry generation as well as the performance of the simulation method with and without a parallel geometry, and shows the applicability of the new method by running a simulation in a complex geometry. Finally, conclusions are drawn in Sec. 4.

2 NUMERICAL METHODS

The simulation of the flows presented in this paper makes use of the *Lattice-Boltzmann Method* (*LBM*) which is briefly described in Sec. 2.1. Since the generation of a parallel geometry is based on the parallel grid generator presented in [14], an introduction to the grid generator is given in Sec. 2.2. Then, the algorithms for the parallel geometry are discussed in Sec. 2.3.

2.1 Lattice-Boltzmann Method

The *LBM* considers the evolution of *particle probability density distribution functions* (*PPDFs*) f_i in phase space by solving the discretized *Boltzmann* equation, i.e., the *Lattice-Bhatnagar-Gross-Krook* equation [22, 23] given by

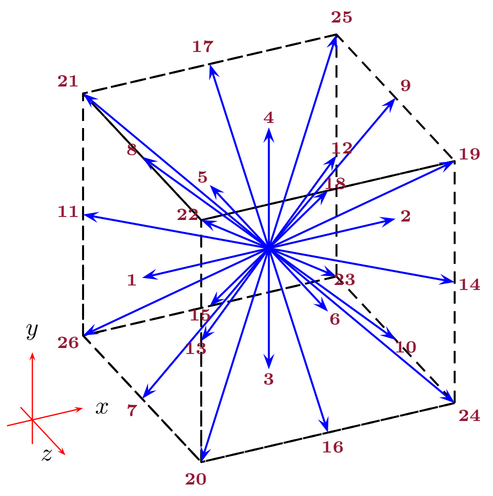
$$f_i\left(\mathbf{x}+\xi_i\delta t, t+\delta t\right) = f_i\left(\mathbf{x},t\right) + \omega\delta t\cdot\left[f_i^{eq}\left[\mathbf{x},t\right) - f_i\left(\mathbf{x},t\right)\right], \tag{1}$$

where $\mathbf{x}$ represents the grid location, ξ_i is the grid velocity vector, t is the time, and δt is the time increment. The equation describes a relaxation towards a thermodynamical equilibrium by the collision operator on the right hand side. That is, the system is relaxed towards the *Maxwell* equilibrium distribution function f_i^{eq} with a collision frequency given by $\omega\delta t$. The left hand side of Eq. 1 describes the streaming operation in which the collision information is propagated to the neighboring grid locations. The index $i \in [0,\ldots,18]$ represents a discrete direction in three dimensions in the *D3Q19* discretization model [24] as shown in Fig. 1.
In Sec. 3, the simulation of the flow in the human respiratory tract at inspiration is used as benchmark case to measure the performance of the code. It uses an adaptive pressure *Dirichlet* condition at the outlets and the *Saint-Vernant and Wantzel* extrapolation boundary condition for the pressure at the inlets [6]. On both sides, at the inlets and outlets, a *Neumann* condition is used for the velocity. For the wall, the no-slip *BFL*-rule [19] is imposed. For more details on the *LBM* and the boundary conditions the reader is referred to [6].

2.2 Parallel grid generation

The parallel grid generator [14] enables a fully automatic creation of large-scale computational *Cartesian* meshes stored in a hierarchical octree in a short amount of time on hundreds of thousand of processes. For process communication it makes use of the *Message Passing Interface* (*MPI*) and I/O is performed by the *parallel NetCDF* library [25]. The meshing algorithm consist of three basic steps:

Step 1 - serial generation of a coarse, uniformly refined mesh
Step 2 - decomposition of this coarse mesh and parallel uniform refinement
Step 3 - local boundary and patch refinement



$$\xi_i = \xi_0 \begin{cases} (\mp1,0,0),(0,\mp1,0),(0,0,\mp1), & i = 1,\dots,6\\ (\mp1,\mp1,0),(\mp1,0,\mp1),(0,\mp1,\mp1), & i = 7,\dots,18\\ \left\{(0,0,0),\right. & i = 19, \text{(D3Q19)}\\ \left\{\begin{array}{l}(\mp1,\mp1,\mp1),\\ (0,0,0),\end{array}\right. & \begin{array}{l} i = 19,\dots,26, \text{(D3Q19)}\\ i = 27, \text{(D3Q27)}\end{array} \end{cases}$$

Figure 1: Three-dimensional *D3Qm* discretization model of the *LBM*, where $m \in \{15, 19, 27\}$.

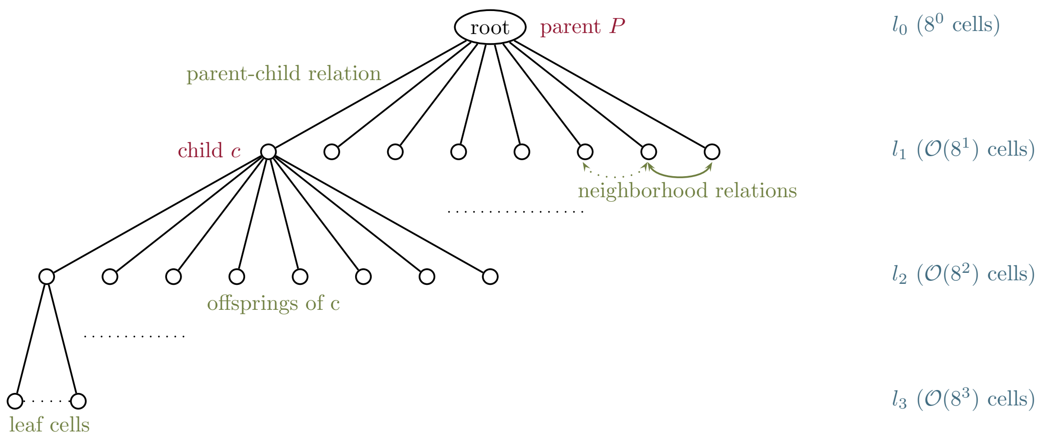


Figure 2: Hierarchical octree of the *Cartesian* mesh including parent-child relations (links between the levels) and the neighborhood relations among cells on equal levels. The upper bound of the number of cells is given by the maximum level, i.e., for l_3 the number of cells is bound by $\mathcal{O}(8^3)$.

In ***Step 1***, the geometry available in *STL* format is first loaded from disk and stored in an *Alternating Digital Tree* (*ADT*) [26]. Then, each *MPI*-process starts to serially refine an initial cube that is placed around the geometry. During each refinement step, the newly generated cells are inserted into an octree (see Fig. 2), where the links between the consecutive levels l_k, l_{k+1} constitute a parent-child relationship and neighborhood relations among cells on the same level are recorded. As depicted in Fig. 3, cells outside the geometry are removed from the octree by tagging cells intersecting with the geometry as boundary cells $c_b \in C_b$ and by using a recursive fill algorithm originating at an inside cell c_{in} to color the cells enclosed by the boundary cell hull. The cells of C_b are determined by an intersection test with candidate triangles retrieved from the *ADT* in $\log$-time. The corresponding inside/outside determination for the originating cell c_{in} is performed using a ray intersection test with the geometry yielding an inside cell for an odd number of intersections and an outside cell for an even number of intersections. The serial refinement is performed until a user-defined level l_α has been reached. At the end of this step, all levels except l_α are removed from the octree.
The remaining cells on level l_α are partitioned in ***Step 2*** using a *Hilbert* decomposition [27] (see Fig. 4). The neighboring processes are determined by a balanced decomposition along the

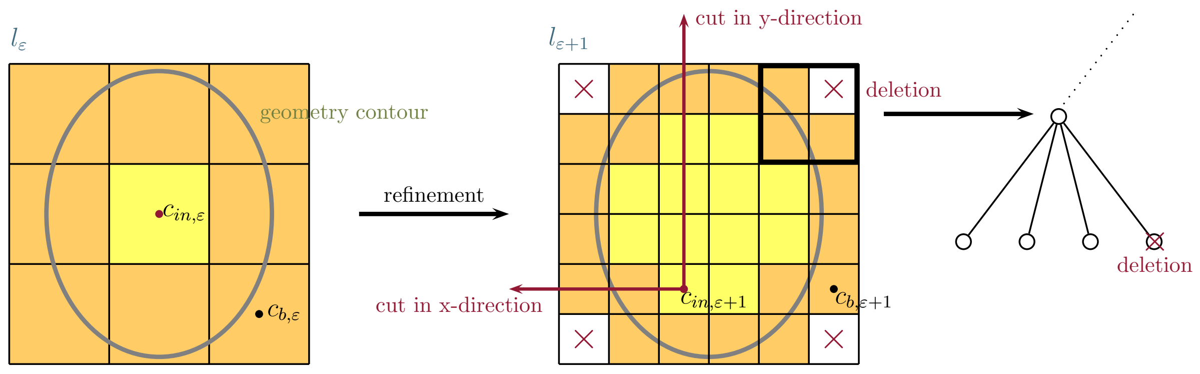


Figure 3: 2D inside/outside detection of cells and removal from the octree. Cells with a geometry intersection on levels l_ε and $l_{\varepsilon+1}$ are marked as $c_{b,\varepsilon}$ and $c_{b,\varepsilon+1}$ (orange). From the inside cells $c_{in,\varepsilon}$ and $c_{in,\varepsilon+1}$, detected by counting ray geometry intersections, inside cells (yellow) are recursively labeled and all non-labeled cells are removed.

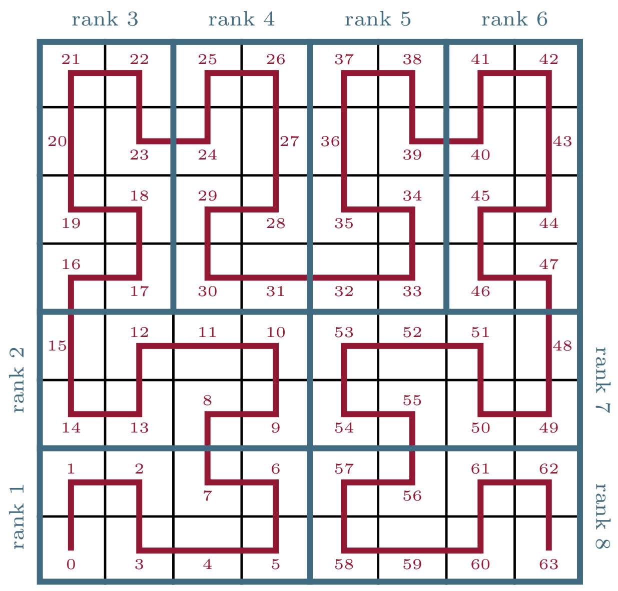
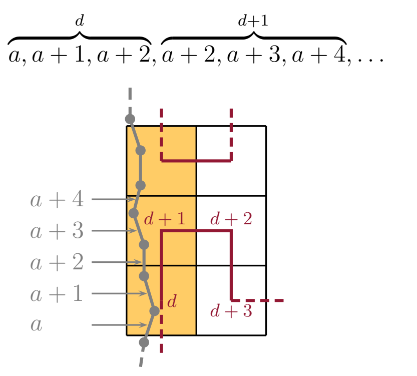


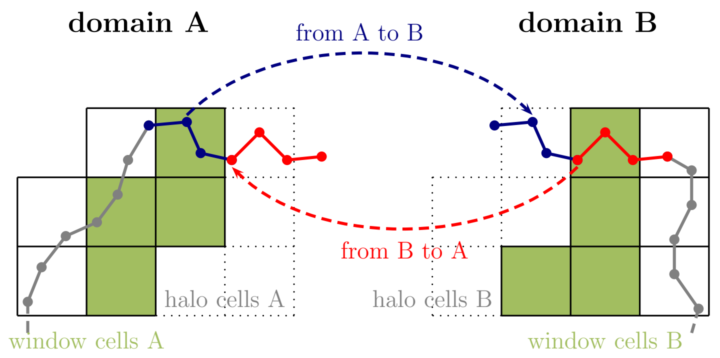
Figure 4: Two-dimensional *Hilbert* curve (red) and the corresponding decomposition (blue) based on the *Hilbert* id.

Hilbert curve and the respective cell neighborhood relations. Then, each process removes the process-foreign cells and continues with parallel refinement of the remaining cells including the removal of outliers until a level l_β has been reached.
In the last ***Step 3***, local refinement based on user-defined patches and/or the boundary distance is performed. For patch refinement, cells inside a previously defined patch geometry are tagged and are then refined. Local boundary refinement is based on a user-defined distance from the original geometry. The algorithm recursively counts the cell distance from the geometry surface across processes and based on the layer count it refines the respective cells. This step is repeated on the finer levels until a final level l_γ has been reached that guarantees a maximum level distance between neighboring cells of one, i.e., $\Delta l = l(c_d) - l(c_e) = 1$ for neighboring cells c_e and c_d. Subsequently, it is necessary to translate local cell ids into global cell ids, which is done by locally counting the number of cells, sharing this information among all processes, and by determining the local offsets from these numbers. Note that the cells below the level l_α are ordered in depth-first manner. That is, the cells on l_α are ordered by the *Hilbert* curve



$\overbrace{a, a+1, a+2}^{d}, \overbrace{a+2, a+3, a+4}^{d+1}, \ldots$
$a+4$
$a+3$
$a+2$
$a+1$
a
$d+1$
$d+2$
d
$d+3$

(a) Triangle detection for the definition of a parallel geometry file.



(b) Triangle exchange between two neighboring domains with one window and halo layer. Each triangle carries an original id such that duplicates are avoided.

Figure 5: Generation of a parallel geometry and window and halo cell exchange in the initialization phase of the simulation.

while for all levels l_ζ with $\zeta > \alpha$ the ordering follows a z-curve. The coarse level cells ordered by the *Hilbert* curve are additionally stored in a list $\mathcal{L}$ which is later used in the simulation to decompose the domain with the help of workloads stored per cell. Furthermore, to avoid large subtrees originating from cells in $\mathcal{L}$, cells with a number of descendants larger than a specified threshold t_d are replaced by their direct children in this list. Finally, the mesh is written to disk in parallel using *parallel NetCDF* [25]. The output of the grid generator enables to run simulations on a quasi-arbitrary number of processes and is only limited by the number of elements in $\mathcal{L}$ and by the available memory per *MPI*-rank.
For further details on the meshing algorithm and its parallel performance, the reader is referred to [14].

2.3 Parallel geometry processing

The exploitation of parallelism for geometries involves the generation of a parallel geometry file from serial information. Furthermore, suitable initializations and boundary conditions for parallel geometries are necessary. In the following, the corresponding algorithms are explained.

Generation of a parallel geometry The order of the cells in the list $\mathcal{L}$ constructed in ***Step 3*** (Sec. 2.2) and the respective workloads constitute the spatial decomposition of the flow domain for a parallel computation. It ensures process-locality for the cells and can hence also be used to generate a parallel geometry that allows for a process-locality of the corresponding triangles. For this purpose, after the generation of $\mathcal{L}$, its elements $c_\mathcal{L}$ are traversed and the original triangle ids $o(T_i)$ of the geometry inside each such cell is collected in a variable two dimensional array, where candidate triangles are retrieved from the *ADT* (see Fig. 5(a)). The first dimension of this array is given by the number of cells $|\mathcal{L}|$ of $\mathcal{L}$ and the second dimension holds the ids

$$o(c_\mathcal{L}) = \{o(T_i) : \exists \mathbf{c}(T_i)\,(\mathbf{c}(T_i) \text{ inside } c_\mathcal{L})\}, \tag{2}$$

where $\mathbf{c}(T_i)$ is a vertex in T_i. Note that elements of $o(c_\mathcal{L})$ may be associated with more than one cell as the respective triangle vertices may be located in different cells. To this end, each process holds a list of triangles incidental to the local fraction of $\mathcal{L}$. Finally, the number of triangles per cell $\Xi(c_\mathcal{L})$, the original triangle ids $o(c_\mathcal{L})$, the vertex coordinates $\mathbf{c}$, and the triangle normals $\mathbf{n}$ are written to disk using *parallel NetCDF* [25]. In sum, the information given in Tab. 1 is

no.	name	meaning	no. elements	size
1	`vertices[3][3]`	triangle coordinates	$\|o(c_{\mathcal{L}})\|$	9×`double`
2	`normal[3]`	normal	$\|o(c_{\mathcal{L}})\|$	3×`double`
3	`segmentId`	geometry segment id	$\|o(c_{\mathcal{L}})\|$	`int`
4	`originalTriId`	original triangle id $o(c_{\mathcal{L}})$	$\|o(c_{\mathcal{L}})\|$	`int`
5	`noTriPerMinCell`	number of triangles per cell $\Xi(c_{\mathcal{L}})$	$\|\mathcal{L}\|$	`int`
6	`noRealTri`	size of $o(c_{\mathcal{L}})$	1	`int`
7	`boundingBox[6]`	min. and max. of T_i	-	6×`double`
8	`BC`	the boundary condition id	-	`int`

Table 1: Triangle variables and their sizes. The information $1, \ldots, 6$ is written to disk, while 7 and 8 are assigned in the solver preprocessing.

stored per triangle in the file. As will be shown in Sec. 3.3 the overhead for the creation of the parallel geometry is negligibly small compared to the mesh generation and the computation of the flow. The parallel geometry file is larger than the original *STL* file since triangles may be listed several times due to the aforementioned multiple cell affiliations. The file size ratio of the *parallel NetCDF* and the *STL* file

$$\sigma = \sigma_{NetCDF}/\sigma_{STL} \tag{3}$$

is on the one hand determined by the location of the triangle vertices and on the other hand by the size ratio of the triangles and the grid distance δ_x. Note that in current simulation setups as they are used, e.g., in the simulation of the flow in the human nasal cavity [6], the file size of both, the *STL* and the *NetCDF* file, is of the order of $\mathcal{O}(1\ GB)$. The spatial resolution δ_x and the geometry resolution usually defines ratios σ of the order of $\mathcal{O}(10)$ for such simulations, i.e., the geometrical resolution is often only slightly higher than the mesh resolution. The results presented in Sec. 3.3 will show that a ratio of $\sigma = 1.92$ is achieved for a highly resolved respiratory tract geometry and a large-scale computational mesh.

Simulation initialization using a parallel geometry Due to the parallel nature of the geometry, some special treatment is required for the initialization of the simulation. This treatment is covered in three major steps:

Step a - parallel reading of the geometry file
Step b - triangle exchange for the halo cells
Step c - boundary condition treatment

In ***Step a***, all *MPI*-ranks open the parallel geometry and the corresponding sub-grid file. From the grid file, the workload array is read and used to find the optimal partition of the cells in $\mathcal{L}$ (cf. [14]). This partition is then used to subdivide the triangles stored in the parallel geometry file. Subsequently, each process reads the triangles based on the number of triangles per cell $\Xi(c_{\mathcal{L}}) = |o(c_{\mathcal{L}})|$ in $\mathcal{L}$.
During the initialization of the mesh, the window cells, which are the neighbors to cells on a neighboring *MPI*-rank, are identified and a copy of these cells is created on the neighbor domains as halo cells. The triangles in each such window cell in $\mathcal{L}$ need to be transferred to the respective process neighbors to ensure triangle consistency among all processes. Therefore, in ***Step b*** the original triangle ids $o(c_{\mathcal{L},r})$ for the neighboring rank r are first transferred for all cells $\mathcal{L}_r \subseteq \mathcal{L}$ to r. Then, on process r it is checked which elements of $o(c_{\mathcal{L},r})$ are already present on

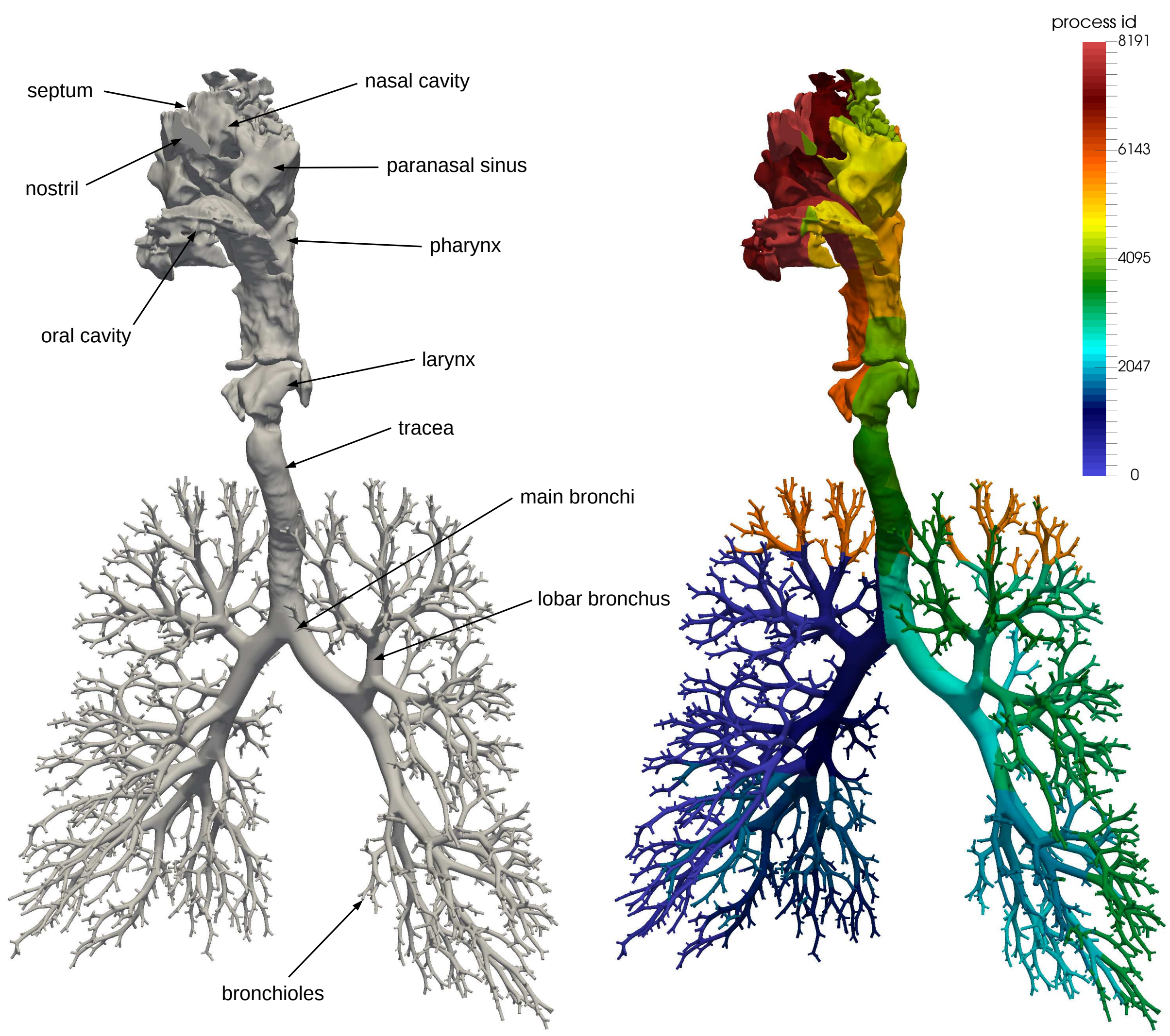


Figure 6: On the left, the geometry of the respiratory tract used for the memory and performance measurements is shown. The right side shows the parallel geometry. Each colored patch corresponds to a single process-id. In total, 8192 cores have been used for the parallel geometry in this case, i.e., there exist at maximum 8192 colored patches.

this domain and which are missing. Subsequently, the originating process is notified about the missing triangles and sends them to r. To this end, all processes have all triangles available for later geometry processing. Fig. 5(b) shows the situation for two neighboring domains.
Some boundary conditions of the flow simulation need special treatment when applying a parallel geometry. For example, the hydraulic diameter $D_h = 4A/C$, based on the area A and the circumference C of a surface segment Ω_s, e.g., of an inlet geometry, is often used as reference length in the REYNOLDS number $Re = (u \cdot D_h)/\nu$, where u is a reference velocity and ν is the kinematic fluid viscosity. Furthermore, the minimal distances of boundary cell centers to the boundary of Ω_s are required to prescribe boundary conditions like parabolic *Dirichlet* inflow profiles for the velocity. The calculation of these values hence necessitates the process-locality of the whole geometry segment, however, Ω_s may be shared between different processes. To overcome these problems, the required values need to be transferred between the different processes in ***Step c***. That is, for a set of shared processes $S = \{p_a, \ldots, p_b\}, 0 \leq a < b \leq n$, where n is the total number of *MPI*-ranks and $p_a, \ldots, p_b$ define a *MPI* sub-communicator, the area is

calculated by

$$A = \sum_{i=p_a}^{p_b} \sum_{k=0}^{l} A_{i,k}, \tag{4}$$

where $A_{i,k}$ is the area of the k^{th} triangle on process i in Ω_s. To obtain the circumference the length of all boundaries of Ω_s is summed

$$C = \sum_{i=p_a}^{p_b} C_i, \tag{5}$$

where C_i is calculated by the sum of the lengths of the edges of the triangles in Ω_s that are incident to only one triangle (boundary edges). However, to calculate the distances it is necessary to have the whole boundary of Ω_s available on $p_a, \ldots, p_b$. Therefore, the boundary edges are gathered on all processes of the corresponding *MPI* sub-communicator.

Note that due to the reduced number of triangles in the process-local *ADT*s, searches, as they are performed for the wall distance calculation of the *LBM BFL*-rule, are at average reduced from

$$\log_2\left(\mathcal{O}(|T_s|)\right) \quad \text{to} \quad \log_2\left(\frac{\mathcal{O}(|T_p|)}{n}\right), \tag{6}$$

where $|T_s|$ is the number of global triangles in the serial case, $|T_p|$ is the number of triangles in the parallel case, and n is the number of *MPI*-ranks. The number of triangles in T_p is usually of the order of $o(|T_s|)$.

3 RESULTS

In this section, the total and the per-process memory consumption for a simulation Ψ of the flow in the complete human respiratory system down to the 12th lung generation using the two configurations of a parallel and a non-parallel geometry are investigated in Sec. 3.1. In addition to the memory consumption, the performance of the simulation preprocessing for the computations is juxtaposed for both, the non-parallel and the parallel geometry version, in Sec. 3.2. Finally, in Sec. 3.3 a brief analysis of the flow field obtained from the *LBM* simulation using the parallel geometry is presented.
The respiratory system geometry consists of $7 \cdot 10^6$ triangles and consumes 645 MB in *NetCDF* format and 334 MB in 32-bit binary *STL* format on disk. The upper respiratory tract down to the end of the trachea is reconstructed from *CT* image data using the methods described in [28]. The image has a spatial transverse, coronal, and saggital resolution of 0.49 mm, 0.49 mm, and 0.7 mm and was recorded by a Siemens *CT* scanner. The lower respiratory tract is generically generated by using the software *Lung4Cer* [29] and has been glued to the upper respiratory tract surface. The geometry of Ψ is stored in 1,747 files (two inlets, 1,744 outlets, and the wall). The corresponding parallel geometry file has a size of $1,239$ MB, i.e., the respective *STL* in *NetCDF* format to parallel geometry *NetCDF* file size ratio is $\sigma = 1.92$. The information on the parallel geometry is stored in a single file. The geometry of the respiratory tract is shown on the left side in Fig. 6.
All measurements were performed on the IBM BlueGene/Q *JUQUEEN* [21] system at the *JSC*. The *JUQUEEN* 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. It is capable of running a maximum number of 4 *OpenMP* threads per core.

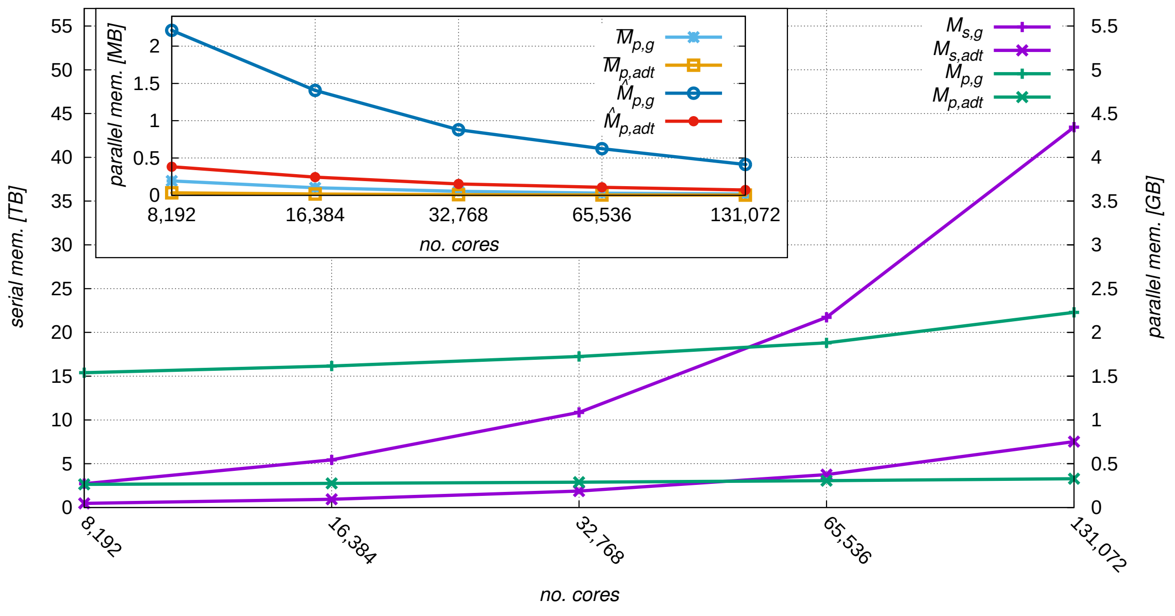


Figure 7: Total memory consumption of the simulations Ψ_s and Ψ_p on $8,192,\ldots,131,072$ cores. The graphs for the serial geometry and the serial *ADT* (purple) use the left ordinate scale counting in TB ($\mathcal{M}_{s,g}$ and $\mathcal{M}_{s,adt}$), while the graphs for parallel geometry and parallel *ADT* ($\mathcal{M}_{p,g}$ and $\mathcal{M}_{p,adt}$, green) use the right ordinate counting in GB. The inset shows the per-process averaged ($\overline{\mathcal{M}}_{p,g}$ and $\overline{\mathcal{M}}_{p,adt}$) and maximum ($\hat{\mathcal{M}}_{p,g}$ and $\hat{\mathcal{M}}_{p,adt}$) memory allocations for the parallel geometry and the *ADT*.

3.1 Memory consumption

For the simulation of the respiratory tract, memory measurements using the non-parallel geometry (simulation Ψ_s) and the parallel geometry (simulation Ψ_p) are performed. For both simulation cases, jobs using 8,192, 16,384, 32,768, 65,536, and 131,072 cores are submitted. Since the geometry consumes approx. $1,390$ MB in memory, *MPI*-processes are started on only every fourth core of the *JUQUEEN* nodes using 16 *OpenMP* threads for Ψ_s. This allows having enough memory available for the simulation. Note that in addition to the variables stored in the *parallel NetCDF* file, further variables are allocated per triangle, i.e., the bounding box, and the boundary condition id (see Tab. 1). For Ψ_p 16 *MPI*-ranks per node and 4 *OpenMP* threads are used for the simulations. The right image in Fig. 6 shows the corresponding triangle distribution on 8,192 cores. Fig. 7 and Tab. 2 show the results of the memory measurements for Ψ, where the left ordinate in Fig. 7 represents the memory consumption $\mathcal{M}_s$ in TB for the computations using the serial geometry and the right ordinate the consumption $\mathcal{M}_p$ of the parallel geometry approach in GB. While the amount of memory consumed by the serial geometry stays constant across all domains, the total allocated amounts of memory are $\mathcal{M}_{s,g}(8,192) \approx 2780.7$ GB and $\mathcal{M}_{s,adt}(8,192) \approx 481.3$ GB for the geometry itself and the *ADT* (left ordinate in Fig. 7), in sum $\mathcal{M}_{s,\Sigma}(8,192) = 3261$ GB ≈ 3.19 TB. Doubling the number of cores results in a doubling of the allocated memory leading to a total amount of memory on 131,072 cores of $\mathcal{M}_{s,\Sigma}(131,072) = 52191.3$ GB ≈ 51 TB for Ψ_s. In contrast, the total amount of consumed memory of the parallel geometry increases slightly from $\mathcal{M}_{p,g}(8,192) = 1.54$ GB for 8,192 cores to $\mathcal{M}_{p,g}(131,072) = 2.28$ GB for 131,072 cores (right ordinate in Fig. 7). The same trend is visible for the *ADT*, with an increase from $\mathcal{M}_{p,adt}(8,192) = 0.27$ GB to $\mathcal{M}_{p,adt}(131,072) \approx 0.34$ GB. Hence, the sum $\mathcal{M}_{p,\Sigma}$ for all core number also increases from $\mathcal{M}_{p,\Sigma}(8,192) = 1.81$ GB to $\mathcal{M}_{p,\Sigma}(131.072) \approx 2.62$ GB. The overall slight upward tendency is caused by the increase of the local triangle count

simulation	$\mathcal{M}_g$ [GB]	$\mathcal{M}_{adt}$ [GB]	$\mathcal{M}_\Sigma$ [GB]	$\overline{\mathcal{M}}_g$ [MB]	$\overline{\mathcal{M}}_{adt}$ [MB]	$\hat{\mathcal{M}}_g$ [MB]	$\hat{\mathcal{M}}_{adt}$ [MB]
$\Psi_s(8,192)$	2,780.684	481.271	3,261.955	-	-	-	-
$\Psi_s(16,384)$	5,561.368	962.542	6,523.910	-	-	-	-
$\Psi_s(32,768)$	11,122.736	1,925.084	13,047.820	-	-	-	-
$\Psi_s(65,536)$	22,245.472	3,850.168	26,095.640	-	-	-	-
$\Psi_s(131,072)$	44,490.944	7,700.336	52,191.280	-	-	-	-
$\Psi_p(8,192)$	1.540	0.270	1.810	0.193	0.033	2.212	0.383
$\Psi_p(16,384)$	1.655	0.282	1.937	0.101	0.017	1.406	0.243
$\Psi_p(32,768)$	1.766	0.296	2.062	0.054	0.009	0.879	0.152
$\Psi_p(65,536)$	1.925	0.314	2.239	0.029	0.005	0.625	0.108
$\Psi_p(131,072)$	2.282	0.336	2.618	0.017	0.003	0.413	0.071

Table 2: Absolute geometry memory consumptions of the simulations Ψ_s and Ψ_p performed on $8,192,\dots,131,072$ cores of the *JUQUEEN* system. The table shows the total global memory allocation for the geometry and the *ADT* $\mathcal{M}_g$, $\mathcal{M}_{adt}$ and the accumulated memory allocation $\mathcal{M}_\Sigma$ in GB. Furthermore, the process-averaged $\overline{\mathcal{M}}_g$ and $\overline{\mathcal{M}}_{adt}$ and the maxima $\hat{\mathcal{M}}_g$ and $\hat{\mathcal{M}}_{adt}$ are given in MB.

for higher core numbers as a consequence of the increased triangle exchange in the halo cells. The inset of Fig. 7 and Tab. 2 additionally give the process-averaged $\overline{\mathcal{M}}_{\{g,adt\}}$ and maximum $\hat{\mathcal{M}}_{\{g,adt\}}$ memory allocation of the geometry and the *ADT*. It is clearly visible that due to the scattering of the triangles across all domains, the local memory usage is strongly decreased. To this end, the global memory consumption of the simulation Ψ is reduced by a factor of

$$R_{\mathcal{M}}\left[\Psi(8,192)\right] = \frac{\mathcal{M}_{s,\Sigma}(8,192)}{\mathcal{M}_{p,\Sigma}(8,192)} \approx 1,802 \tag{7}$$

using the same number of cores of 8,192 for the serial and the parallel geometry simulation. This factor is with

$$R_{\mathcal{M}}\left[\Psi(131,072)\right] = \frac{\mathcal{M}_{s,\Sigma}(131,072)}{\mathcal{M}_{p,\Sigma}(131,072)} = 19,936 \tag{8}$$

even higher for 131,072 cores.

3.2 Preprocessing performance

The same simulations as analyzed in Sec. 3.1 are used to juxtapose the preprocessing performance of the serial and parallel geometry version, i.e., for Ψ_s and Ψ_p on 8,192 up to 131,072 cores on the *JUQUEEN* system. The analysis considers the performance of the code sections that are associated with the geometry. That is, the performance of the geometry I/O, the calculation of the bounding box for each triangle, the setup of the *ADT*, the calculation of the normal vectors for inlets and outlets, and the *LBM* wall-distance calculation for wall-boundary cells required for the *BFL*-rule [19], is measured. Note that due to the equal processing time for the serial geometry, only the results obtained on 8,192 cores for Ψ_s are juxtaposed to the parallel geometry results. Fig. 8 and Tab. 3 show the results of the performance measurements for Ψ_s and Ψ_p. It is clearly visible that in $\Psi_s(8,192)$ the reading of the geometry consumes with $\mathcal{T}_{s,rdg}(8192) \approx 763.7$ s most of the time. This is caused by the serial reading from disk of

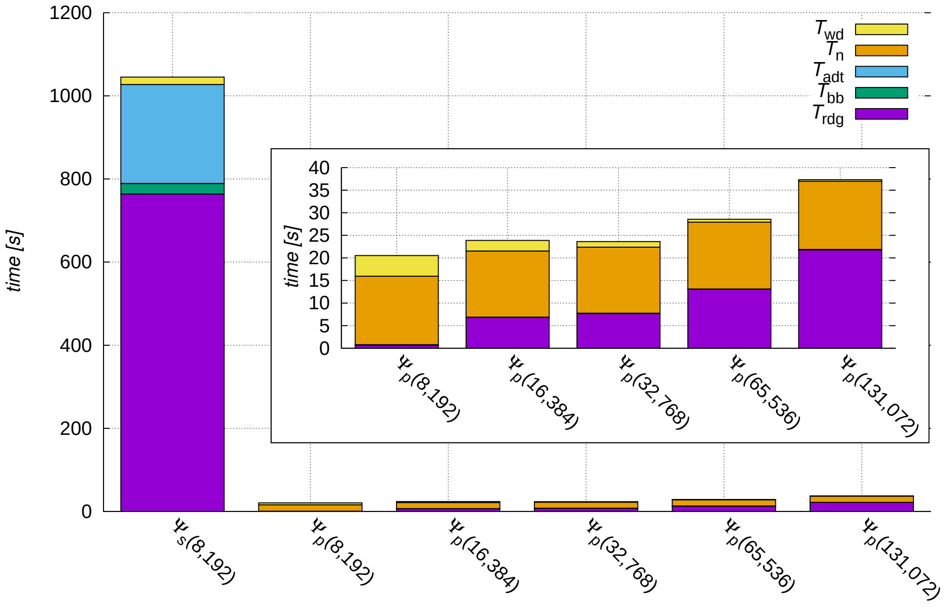


Figure 8: Performance of different code sections involving geometry treatment during the processing of simulation Ψ on different core numbers $8, 192, \ldots, 131, 072$ using a serial and parallel geometry. The code section times are recorded for the wall-distance calculation $\mathcal{T}_{wd}$, normal calculation $\mathcal{T}_n$, setting up the *ADT* $\mathcal{T}_{adt}$, calculating the bounding boxes of all triangles $\mathcal{T}_{bb}$, and reading and distributing the geometry $\mathcal{T}_{rdg}$. The inset shows a zoom of the parallel code performance.

	$\mathcal{T}_{wd}$ [s]	$\mathcal{T}_{n}$ [s]	$\mathcal{T}_{adt}$ [s]	$\mathcal{T}_{bb}$ [s]	$\mathcal{T}_{rdg}$ [s]	$\mathcal{T}_{\Sigma}$ [s]
$\Psi_s(8, 192)$	17.882	0.011	238.175	25.447	763.673	1,045.188
$\Psi_p(8, 192)$	4.584	15.168	0.020	0.077	0.685	20.534
$\Psi_p(16, 384)$	2.350	14.628	0.007	0.075	6.820	23.880
$\Psi_p(32, 768)$	1.217	14.633	0.002	0.073	7.684	23.609
$\Psi_p(65, 536)$	0.637	14.784	0.001	0.077	13.057	28.556
$\Psi_p(131, 072)$	0.337	15.110	0.0003	0.077	21.794	37.318

Table 3: Absolute runtime in seconds for preprocessing algorithms of the simulations using a serial geometry Ψ_s performed on 8192 cores and an the parallel geometry Ψ_p executed on $8, 192, \ldots, 131, 072$ cores of the *JUQUEEN* system. The total runtime $\mathcal{T}_\Sigma$ is subdivided into the time required for the wall-distance calculation $\mathcal{T}_{wd}$, normal calculation $\mathcal{T}_n$, setting up the *ADT* $\mathcal{T}_{adt}$, calculating the bounding boxes of all triangles $\mathcal{T}_{bb}$, and reading and distributing the geometry $\mathcal{T}_{rdg}$.

all the 1747 files containing the $7 \cdot 10^6$ triangles on all processes. The second most expensive algorithm is with $\mathcal{T}_{s,adt}(8, 192) \approx 238.2$ s the setup of the *ADT*. Compared to these values, the durations of the wall-distance calculation $\mathcal{T}_{s,wd}(8, 192)$, normal calculation $\mathcal{T}_{s,n}(8, 192)$, and bounding box calculation $\mathcal{T}_{s,bb}(8, 192)$ are rather small. The total required time is given by $\mathcal{T}_{s,\Sigma}(8, 192) \approx 1045.2$ s. Using the parallel geometry reduces the preprocessing time of Ψ drastically, i.e., using the same amount of cores the total runtime is reduced from $\mathcal{T}_{s,\Sigma}(8, 192)$ to $\mathcal{T}_{p,\Sigma}(8, 192) \approx 20.6$ s by a factor of

$$R_{\mathcal{T}}\left[\Psi(8, 192)\right] = \frac{\mathcal{T}_{s,\Sigma}(8, 192)}{\mathcal{T}_{p,\Sigma}(8, 192)} \approx 51. \tag{9}$$

The inset in Fig. 8 shows a zoom of the parallel geometry runtimes. Due to the increase of the number of computational subdomains using more cores the reading process of the *parallel NetCDF* library slows down (see also Tab. 3). Furthermore, an increased number of triangles needs to be exchanged between *MPI*-ranks due to the growth of the global amount of halo cells for higher core numbers. As it is evident from Tab. 8, the normal calculation stays almost constant for all core counts. This effect can be explained by the fact that the total number of geometry segments that require a normal computation (1746 inlets/outlets) stays constant. Most of the computation is performed locally and hence the influence of the augmentation of the number of sharing processes is small. A comparison of $\mathcal{T}_{s,n}(8192) = 0.011$ s with those of the parallel version shows that the serial normal calculation is much faster since it does not require any communication and an inside/outside detection can directly be performed locally. Finally, looking at the other times, it is obvious that due to the decreasing number of triangles per process under an increasing number of cores the setup of the *ADT* and the bounding box calculations are negligible. Furthermore, the time for the wall-distance calculation continuously decreases from $\mathcal{T}_{p,wd}(8,192) \approx 4.58$ s to $\mathcal{T}_{p,wd}(131,072) \approx 0.34$ s.

3.3 Flow in the respiratory tract

To test the new algorithm, a simulation using 32,768 cores and 4 OpenMP threads per *MPI*-rank is performed on the *JUQUEEN* system, i.e., 2,048 nodes are allocated. The computational mesh has been created by the parallel mesh generator [14] on 4,096 cores in 339.2 s. The coarse-level mesh used for the domain decomposition is on level $l_\alpha = 9$ and the final level is $l_\gamma = 12$ resulting in a total number of cells of $266.5 \cdot 10^6$ and a grid spacing of $\delta_x = 0.11$ mm on l_γ. Note that in Lintermann et al. [14] a mesh $\mathcal{K}$ consisting of $9.82 \cdot 10^9$ could be generated within ≈ 406.8 s. However, the coarse-level mesh of $\mathcal{K}$ is at level $l_{\alpha,\mathcal{K}} = 6$ and the used geometry was a simple cube. That is, serial refinement in the current simulation setup and the inside/outside determination are caused by the large level difference of $l_\Delta = l_\alpha - l_{\alpha,\mathcal{K}} = 3$ and the complex geometry more time consuming. The share of the parallel geometry creation in the mesh generation is with 9.1 s or with 2.7% of the meshing time negligible.
The simulation uses the boundary conditions described in Sec. 2.1 for inspiratory flow and a REYNOLDS number of $Re = (u \cdot D_h)/\nu = 1,515$. Therefore, a bulk velocity magnitude u based on a volume flux of 360 ml/s, the hydraulic diameter D_h of the trachea (see Fig. 6), and the kinematic viscosity of air are applied. To obtain a quasi-steady flow field, approx. $3 \cdot 10^6$ *LBM* iterations have been performed which required around 4 days wall-clock time for the computation on 32,768 cores, i.e., around $3.2 \cdot 10^6$ core-hours. The computations are restarted from checkpoints every 12 hours. All variables introduced in this section are given in non-dimensional *LBM* units.
Fig. 9 shows a cut-plane of the velocity magnitude at inspiration reaching from the left nasal cavity to the first bifurcation in the lung. It is clearly visible that a jet emerges at the epiglottis and continues towards the larynx and below. Further upstream, the flow is laminar while downstream of the larynx the flow becomes transitional and laminarizes again when reaching the first bifurcation of the lung. The left zoom of Fig. 9 shows contours of the Δ-*criterion* [30] given by

$$\Delta_c = \left(\frac{Q}{3}\right)^3 + \left[\frac{\det\left(\nabla \otimes \vec{v}\right)}{2}\right]^2 > 0, \tag{10}$$

which is based on the Q-*criterion* [31]

$$Q = \frac{1}{2}\left(|\Omega|^2 - |S|^2\right) > 0, \tag{11}$$

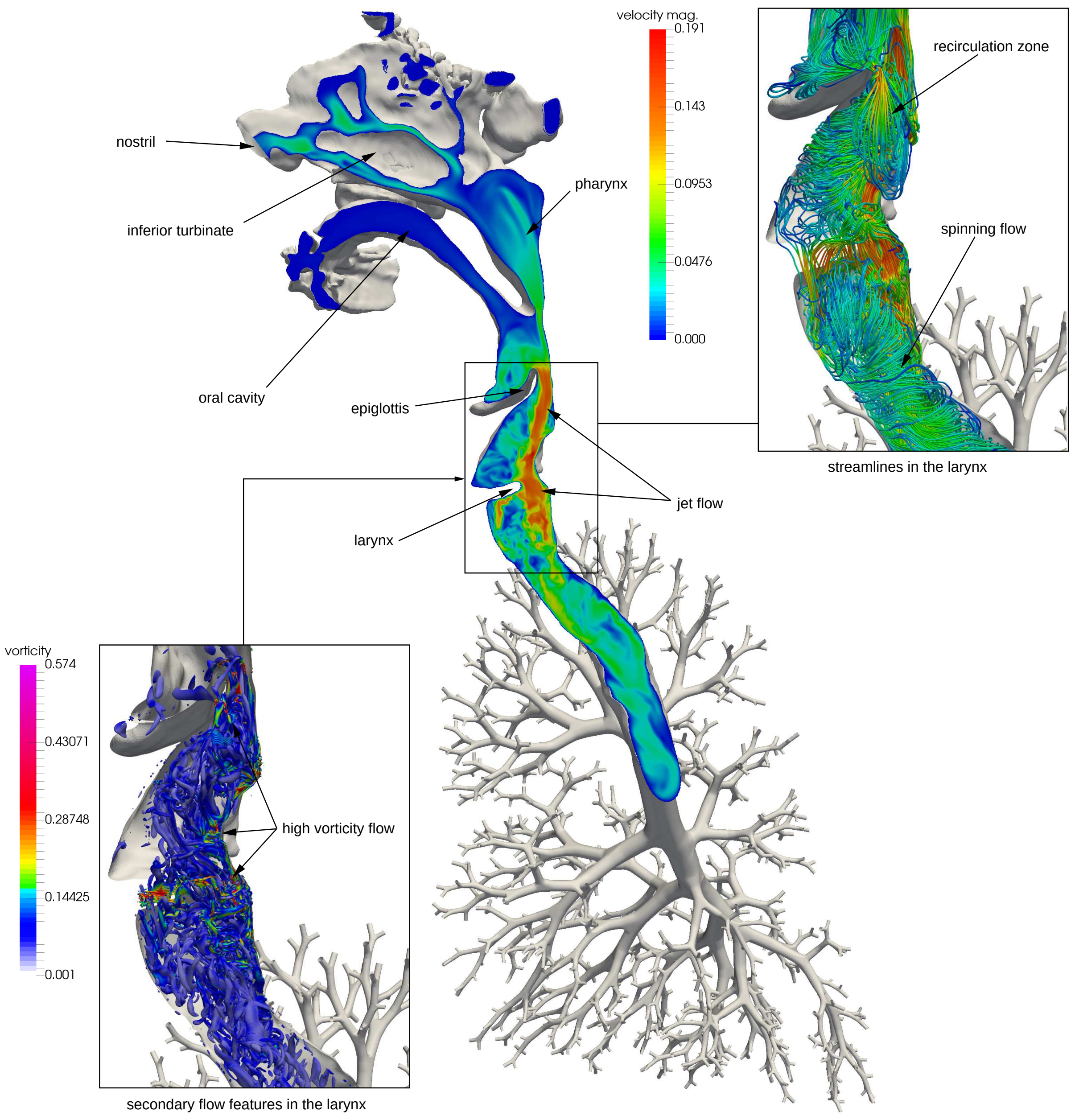


Figure 9: Flow in the human respiratory system at inspiration. In the center, a cut-plane colored by the velocity magnitude is shown. The zooms in the left and right corner show contours of the Δ-*criterion* colored by the vorticity magnitude and streamlines colored by the velocity magnitude.

where $S = \frac{1}{2}\left[\nabla \otimes \vec{v} + (\nabla \otimes \vec{v})^T\right]$ is the strain tensor, $\Omega = \frac{1}{2}\left[\nabla \otimes \vec{v} - (\nabla \otimes \vec{v})^T\right]$ is the vorticity tensor and $|\cdot|$ denotes the *Frobenius norm*. The contours are colored by the vorticity magnitude and evidence the existence of secondary flow structures emerging from the epiglottis. Considering the streamlines shown in the zoom on the right of Fig. 9 a recirculation zone is visible between the epiglottis and the larynx. Downstream of the larynx a spinning wall-bound flow appears that helps to laminarize the flow towards the first bifurcation of the lung.
Note that the simulation of the intrinsic flow in such a complex and highly-resolved geometry is not possible using 16 *MPI*-ranks per *JUQUEEN* node and 4 *OpenMP* threads per task when loading the geometry in serial. In such a case, 4 *MPI*-ranks per node have to be used at maximum and 16 *OpenMP* threads need to be spawned per task. However, in addition to the

waste of memory, *OpenMP* scalability tests on one *JUQUEEN* node have shown that the sweet spot is actually at 16 *MPI*-ranks and 4 *OpenMP* threads and the corresponding computations of the *LBM* are about 1.43 times faster than using 4 *MPI*-ranks and 16 *OpenMP* threads. For the benchmark simulation that means that instead of running 8 batch jobs à 12 hours on the *JUQUEEN* a computation with the serial geometry requires about 11.4 batch jobs à 12 hours to perform the same amount of *LBM* iterations as in the parallel geometry case. Therefore, applying the parallel geometry distribution method not only reduces the allocated memory and increases the preprocessing speed, but also allows for faster simulations.

4 CONCLUSION

In this paper a new and efficient algorithm to generate parallel geometries during the mesh generation has been presented. The algorithm takes advantage of the ordering of coarse-level cells that are output by the parallel grid generator and that determine a domain decomposition of the flow field using a *Hilbert* curve. This leads to a subdivision of the geometry that goes along with the mesh decomposition, i.e., the mesh generator allows for simulations on a quasi-arbitrary amount of cores using distributed geometry elements that are process-local. Instead of loading the geometry on each process in serial and hence consuming a large memory share that could otherwise be used for the simulation, the new method reduces the share to a small fraction. That is, for the simulation of the flow in a respiratory tract on 8,192 cores the total amount of memory allocated by the geometry could be reduced by a factor of 1802.2 from 3.19 TB to 1.8 GB and for 131,072 cores even by a factor of 19,936 from ≈ 51 TB to 2.6 GB. Furthermore, the algorithm allowed reducing the preprocessing time by a factor of about 51 from 1045.2 s to 20.5 s for the simulation on 8,192 cores. The test simulations with higher core numbers of up to 131,072 showed that the global memory and the preprocessing time increases only slightly by using the new method. Finally, some results of the flow field computation have been presented to underline the validity of the algorithm and show that the parallel geometry allows using the sweet spot with respect to a combination of distributed and shared memory parallelization. This leads to a 1.43 times faster computation compared to the serial geometry case using the same number of nodes.
Further investigations will cover the performance increase for other solvers using the same meshing technique, i.e., an analysis of the performance of a *Finite-Volume* solver using *level-set* methods [2, 17] for the representation of moving boundaries and a *discontinuous Galerkin* method for the computation of aeroacoustics [32] will be performed.

5 Acknowledgments

The author gratefully acknowledges the computing time granted by the *JARA-HPC* Vergabegremium and provided on the *JARA-HPC* Partition, part of the supercomputer *JUQUEEN* [21] at Forschungszentrum Jülich. Furthermore, the author would like to thank the Institute of Aerodynamics, RWTH Aachen University, for the provision of their code *ZFS*.

REFERENCES

[1] T. Nakashima, M. Tsubokura, M. Vázquez, H. C. Owen, Y. Doi, Coupled Analysis of Unsteady Aerodynamics and Vehicle Motion of a Heavy-Duty Truck in Wind Gusts, Computers & Fluids 80 (2012) 1–9. doi:10.1016/j.compfluid.2012.09.028.

[2] M. Meinke, L. Schneiders, C. Günther, W. Schröder, A cut-cell method for sharp moving boundaries in Cartesian grids, Computers & Fluids 85 (2013) 135–142. doi:10.1016/j.compfluid.2012.11.010.

[3] A. Pogorelov, M. Meinke, W. Schröder, Cut-cell method based large-eddy simulation of tip-leakage flow, Physics of Fluids 27 (7) (2015) 075106. doi:10.1063/1.4926515.

[4] G. J. M. Garcia, N. Bailie, D. a. Martins, J. S. Kimbell, Atrophic rhinitis: a CFD study of air conditioning in the nasal cavity, Journal of applied physiology (Bethesda, Md. : 1985) 103 (3) (2007) 1082–92. doi:10.1152/japplphysiol.01118.2006.

[5] S. K. Kim, Y. Na, J.-I. Kim, S.-K. Chung, Patient specific CFD models of nasal airflow: overview of methods and challenges, Journal of Biomechanics 46 (2) (2013) 299–306. doi:10.1016/j.jbiomech.2012.11.022.

[6] A. Lintermann, M. Meinke, W. Schröder, Fluid mechanics based classification of the respiratory efficiency of several nasal cavities, Computers in Biology and Medicine 43 (11) (2013) 1833–1852. doi:10.1016/j.compbiomed.2013.09.003.

[7] I. Hörschler, M. Meinke, W. Schröder, Numerical simulation of the flow field in a model of the nasal cavity, Computers & Fluids 32 (1) (2003) 39–45. doi:10.1016/S0045-7930(01)00097-4.

[8] M. Konopka, M. Meinke, W. Schröder, Large-eddy simulation of shock-cooling-film interaction at helium and hydrogen injection, Physics of Fluids 25 (10) (2013) 106101. doi:10.1063/1.4823745.

[9] B. Roidl, M. Meinke, W. Schröder, A reformulated synthetic turbulence generation method for a zonal RANS-LES method and its application to zero-pressure gradient boundary layers, International Journal of Heat and Fluid Flow 44 (2013) 28–40. doi:10.1016/j.ijheatfluidflow.2013.03.017.

[10] B. Roidl, M. Meinke, W. Schröder, Boundary layers affected by different pressure gradients investigated computationally by a zonal RANS-LES method, International Journal of Heat and Fluid Flow 45 (2014) 1–13. doi:10.1016/j.ijheatfluidflow.2013.11.004.

[11] V. Statnikov, T. Sayadi, M. Meinke, P. Schmid, W. Schröder, Analysis of pressure perturbation sources on a generic space launcher after-body in supersonic flow using zonal turbulence modeling and dynamic mode decomposition, Physics of Fluids 27 (1) (2015) 016103. doi:10.1063/1.4906219.

[12] K. Nakahashi, L. Kim, Building-Cube Method for Large-Scale High Resolution Flow Computations, in: 42nd AIAA Aerospace Sciences Meeting and Exhibit, no. January, Reno, Nevada, 2004, AIAA 2004–434. doi:10.2514/6.2004-434.

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

[14] A. Lintermann, S. Schlimpert, J. Grimmen, C. Günther, M. Meinke, W. Schröder, Massively parallel grid generation on HPC systems, Computer Methods in Applied Mechanics and Engineering 277 (2014) 131–153. doi:10.1016/j.cma.2014.04.009.

[15] C. Günther, M. Meinke, W. Schröder, A flexible level-set approach for tracking multiple interacting interfaces in embedded boundary methods, Computers & Fluids 102 (2014) 182–202. doi:10.1016/j.compfluid.2014.06.023.

[16] R. K. Freitas, W. Schröder, Numerical investigation of the three-dimensional flow in a human lung model, Journal of Biomechanics 41 (11) (2008) 2446–2457. doi:10.1016/j.jbiomech.2008.05.016.

[17] L. Schneiders, C. Günther, J. Grimmen, M. Meinke, W. Schröder, Sharp resolution of complex moving geometries using a multi-cut-cell viscous flow solver, in: 22nd AIAA Computational Fluid Dynamics Conference, American Institute of Aeronautics and Astronautics, Reston, Virginia, 2015, AIAA 2015–3427. doi:10.2514/6.2015-3427.

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

[19] M. Bouzidi, M. Firdaouss, P. Lallemand, Momentum transfer of a Boltzmann-lattice fluid with boundaries, Physics of Fluids 13 (11) (2001) 3452–3459. doi:10.1063/1.1399290.

[20] D. Hartmann, M. Meinke, W. Schröder, A strictly conservative Cartesian cut-cell method for compressible viscous flows on adaptive grids, Computer Methods in Applied Mechanics and Engineering 200 (9-12) (2011) 1038–1052. doi:10.1016/j.cma.2010.05.015.

[21] M. Stephan, J. Docter, JUQUEEN: IBM Blue Gene/Q Supercomputer System at the Jülich Supercomputing Centre, Journal of large-scale research facilities JLSRF 1 (2015) A1. doi:10.17815/jlsrf-1-18.

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

[23] D. Hänel, Molekulare Gasdynamik, Einführung in die kinetische Theorie der Gase und Lattice-Boltzmann-Methoden, Springer-Verlag, 2004.

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

[25] J. Li, M. Zingale, W.-k. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, Parallel netCDF: A High-Performance Scientific I/O Interface, in: Proceedings of the 2003 ACM/IEEE conference on Supercomputing - SC '03, ACM Press, New York, New York, USA, 2003, p. 39. doi:10.1145/1048935.1050189.

[26] J. Bonet, J. Peraire, An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems, International Journal for Numerical Methods in Engineering 31 (1) (1991) 1–17. doi:10.1002/nme.1620310102.

[27] H. Sagan, Space-Filling Curves, 1st Edition, Springer, 2007.

[28] G. Eitel, R. K. Freitas, A. Lintermann, M. Meinke, W. Schröder, 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, Springer Berlin / Heidelberg, 2010, pp. 513–520.

[29] H. Kitaoka, A 4D Model Generator of the Human Lung, Forma 26 (2011) 19–24.

[30] M. S. Chong, A. E. Perry, B. J. Cantwell, A general classification of three-dimensional flow fields, Physics of Fluids 2 (1990) 765–777.

[31] J. C. R. Hunt, A. A. Wray, P. Moin, Eddies, Stream, and Convergence Zones in Turbulent Flows, Center For Turbulence Research Report CTR.

[32] M. A. Schlottke, H.-J. Cheng, A. Lintermann, M. H. Meinke, W. Schröder, A direct-hybrid method for computational aeroacoustics, in: 21st AIAA/CEAS Aeroacoustics Conference, Dallas, Texas, USA, 2015, AIAA 2015–3133. doi:10.2514/6.2015-3133.