ParCFD2024
35th International Conference on Parallel Computational FluidDynamics
Sep. 02-04, 2024, Bonn, Germany

# GPU Accelerated FEM-Based Lagrangian Particle Tracking Framework for Human Air Pathway[†]

Thivin Anandh[a,*], Sashikumaar Ganesan[a]

[a]*Indian Institute of Science, Department of Computational and Data Sciences, Bangalore, 560012, India*

ARTICLE INFO[†]

*Keywords*:
Finite Element Method
GPU Accelerated Particle Tracking
Flow Modelling in Human Air Pathway
Lagrangian Particle Tracking

ABSTRACT

Efficient particle deposition modeling is crucial for understanding regional particle deposition effects on human airways. Lagrangian particle tracking in large, complex domains presents two primary challenges: high computational costs and the requirement for a robust framework with effective interpolation routines and comprehensive mesh information. Finite Element Method (FEM) routines, utilized for fluid modeling, are ideal due to their interpolation capabilities and data structures, which store mesh and geometric information essential for tracking and identifying particle deposition status. However, FEM data structures are complex and usually reside on CPUs, making it challenging to port them to GPUs. Despite this, GPUs offer significant parallelization potential for particle tracking if these data structures can be efficiently managed. In this work, we introduce a GPU-accelerated Lagrangian particle deposition framework utilizing FEM-based routines. Our approach focuses on efficient transfer and simplification of FEM data structures from CPU to GPU, facilitating the implementation of zonal-based particle searching and inertial deposition techniques directly on the GPU. Our model demonstrated lower sequential execution time than ANSYS Fluent's particle tracking module and achieved a 100x speedup over the Sequential CPU implementation and a 4x speedup over the OpenMP implementation for number of particles up to 5 Million. The GPU-accelerated framework reduces execution time from days to hours for complex geometries, enabling deeper exploration of particle deposition in human airways.

## 1. Introduction

Modeling aerosol deposition in the human airway is crucial for understanding regional particle deposition, aiding the development of targeted drug delivery systems [1, 2, 3, 4]. In-vivo methods are limited by safety and regulatory issues, especially with radioactive aerosols [5, 6, 7]. In-vitro studies show that slight variations in geometry and surface irregularities from fabrication significantly impact aerosol deposition [8, 9, 10]. A wide range of fluid flow solvers, including the finite volume method (FVM) [11], finite element method (FEM) [12], and lattice Boltzman method (LBM) [13] have been employed to simulate fluid flow in bounded domains. These solvers incorporate turbulence models such as Reynolds–averaged Navier–Stokes (RANS) [14, 12] and large eddy simulation (LES) [14, 12]. In a study by [13], the authors aimed to enhance particle simulations using GPUs, focusing on the lattice Boltzmann method (LBM). However, this approach lacked particle modeling and experimental validation, limiting comprehensive evaluation and practical applicability. In another work by [15], efforts were made to parallelize particle deposition within the FEM framework. However, this parallelization was achieved exclusively using CPU resources.

## 2. Methodology

### 2.1. Fluid Modeling

The framework was implemented using our in-house FEM package, ParMooN [16]. We solved the unsteady 3D Navier-Stokes equations to simulate fluid flow in the human airway. Turbulence modeling based on the Variational Multiscale Method (VMS) was applied for the Reynolds number 3725. For further details on VMS, see [17]. The computational mesh used in this study, as shown in Fig. 1, consists of 300,000 tetrahedral cells.

[†]This paper is part of the ParCFD 2024 Proceedings. A recording of the presentation is available on YouTube. The DOI of this document is doi and of the Proceedings doi.

*Corresponding author

thivinanandh@iisc.ac.in (T. Anandh); sashi@iisc.ac.in (S. Ganesan)

ORCID(s): 0000-0003-4969-3242 (T. Anandh); 0000-0003-1858-3972 (S. Ganesan)

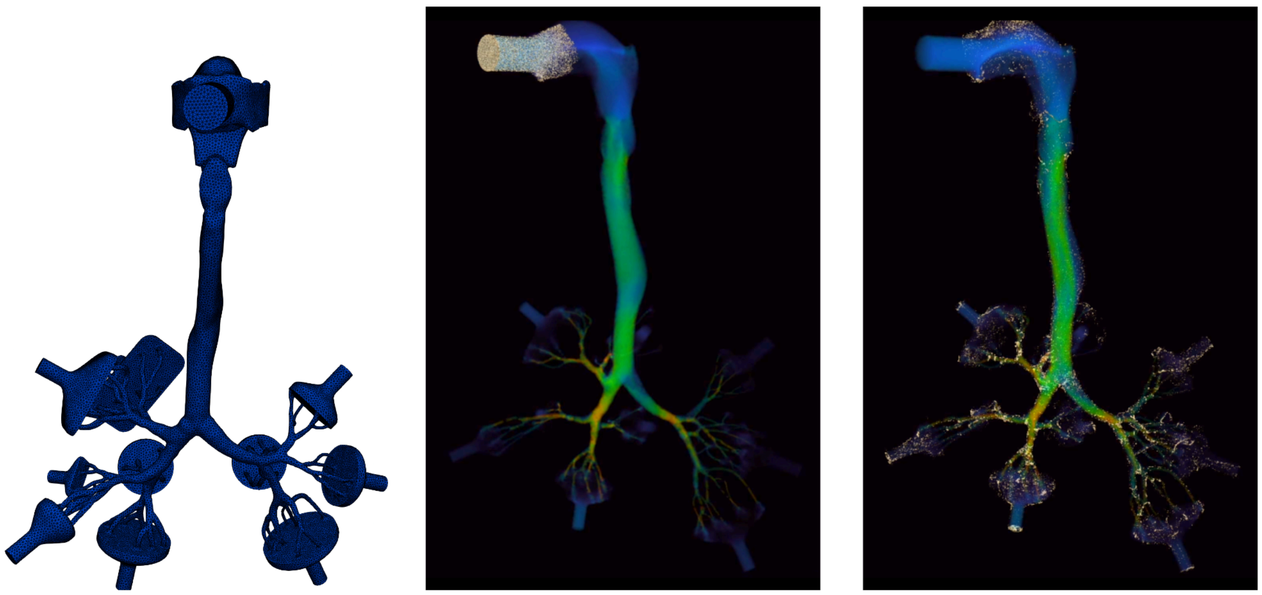

**Figure 1:** Computational mesh, Particle deposition at 3k and 6k timestep with velocity heat maps.

## 2.2. Particle Modelling

The equations of motion for the particles [12] are given by

$$\begin{aligned} m_p \frac{d\mathbf{u}_p}{dt} =& \frac{3}{4}\frac{\rho_f}{\rho_p}\frac{m_p}{d_p}\frac{C_D}{C_C}\left|\mathbf{u}_f - \mathbf{u}_p\right|\left(\mathbf{u}_f - \mathbf{u}_p\right) \\ &+ m_p\mathbf{g}\frac{\rho_p - \rho_f}{\rho_p} + \mathbf{F}_B, \end{aligned} \tag{1}$$

where $\mathbf{u}_p$, $\mathbf{u}_f$, $\rho_p$, $\rho_f$, represent the particle velocity, fluid velocity, particle density, and fluid density, respectively. The terms in the right-hand side are drag force, gravitational force and Brownian force, respectively.

## 3. GPU Implementation

The main concept and parallelization scheme for GPU-accelerated particle tracking are illustrated in Fig. 2. Before the simulation begins, all essential data, including simplified FEM data structures and mesh information, are transferred from the CPU to the GPU and stored there. The particle information is initialized on the CPU and then transferred to the GPU. At each timestep, the corresponding fluid solution is read from the stored file and transferred to the GPU for computation. In the GPU, each thread is assigned to track a particle within the domain. This thread is responsible for solving the particle equations of motion, interpolating velocity values at the particle's current position, performing zonal searches to compute the updated particle position within the domain, and calculating the particle deposition parameters. After all threads complete the computation, a cuda-synchronization call is made on the CPU side. If necessary, particle details can be transferred back to the CPU from GPU for visualization or logging purposes. The fluid solution for the next timestep is then read, and the process is repeated until all particles are deposited or have escaped.

### 3.1. Adaptation of FEM Data Structures for GPU

FEM utilizes complex nested data structures; for example, a cell class may contain edges, and an edge class may contain vertices. Copying these structures to GPUs is complex and introduces computational overhead, especially in multithreaded GPU environments constrained by memory. The core idea of our implementation was to transfer essential data as primitive arrays from CPU to GPU, enabling the implementation of FEM-based algorithms for deposition, interpolation, and tracking routines.

### 3.2. Inertial Deposition Algorithm

The deposition logic triggers when a particle moves outside the computational domain. We determine the deposition point by analyzing the boundary face of the preceding cell and constructing a vector between the previous and current particle positions. The intersection of this vector with the current boundary surface identifies the deposition point, as shown in Fig. 2. Using the simplified FEM data structures, we compute the intersection and orientation of the boundary face in 3D using the simplified boundary face and cell connectivity information transferred from CPU to GPU.

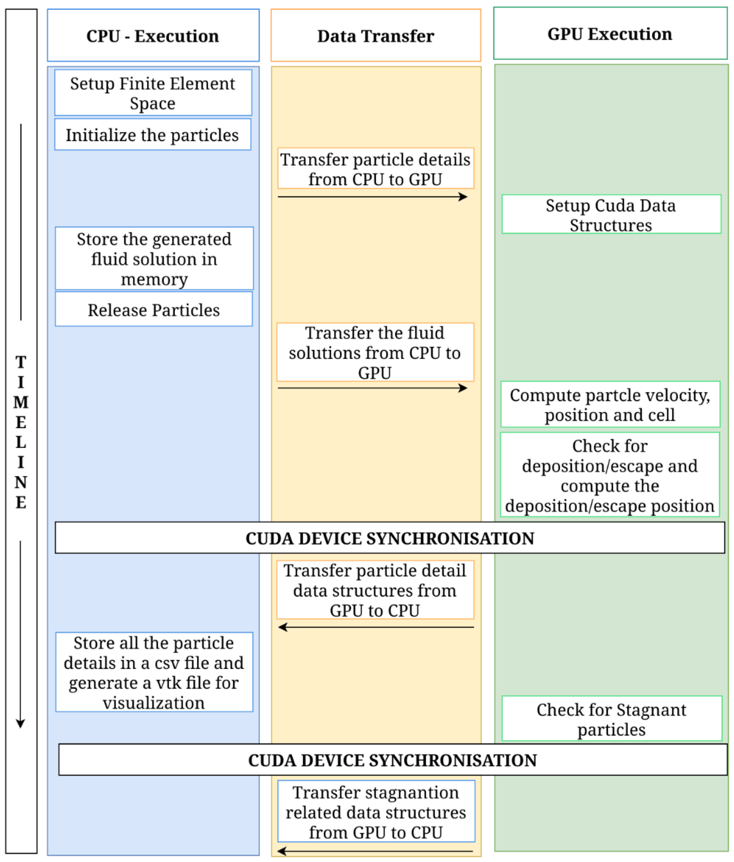
CPU - Execution
Data Transfer
GPU Execution
TIMELINE
Setup Finite Element Space
Initialize the particles
Transfer particle details from CPU to GPU
Setup Cuda Data Structures
Store the generated fluid solution in memory
Release Particles
Transfer the fluid solutions from CPU to GPU
Compute partcle velocity, position and cell
Check for deposition/escape and compute the deposition/escape position
CUDA DEVICE SYNCHRONISATION
Transfer particle detail data structures from GPU to CPU
Store all the particle details in a csv file and generate a vtk file for visualization
Check for Stagnant particles
CUDA DEVICE SYNCHRONISATION
Transfer stagnantion related data structures from GPU to CPU

**Figure 2:** Timeline of GPU accelerated particle deposition.

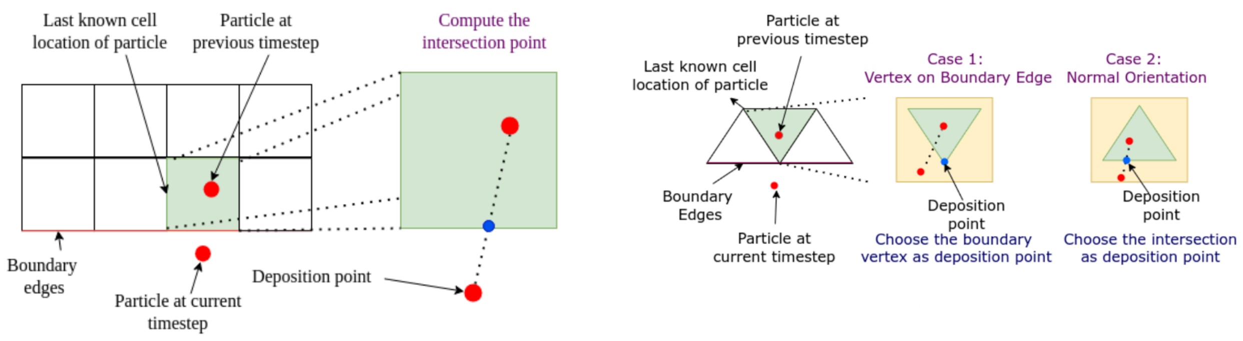
Last known cell location of particle
Particle at previous timestep
Compute the intersection point
Boundary edges
Particle at current timestep
Deposition point
Particle at previous timestep
Last known cell location of particle
Case 1: Vertex on Boundary Edge
Case 2: Normal Orientation
Boundary Edges
Particle at current timestep
Deposition point
Choose the boundary vertex as deposition point
Deposition point
Choose the intersection as deposition point

**Figure 3:** Deposition logic for structured and unstructured cells.

## 3.3. Zonal Particle Search

In a zonal search approach, we need the data structures of the mesh to compute the neighboring cells of the current cell. This results in searching for the particle within the zonal neighborhood rather than the entire domain. This information is necessary to interpolate the underlying velocity value at the current particle position, which is required for solving the particle equations of motion. The neighboring information can be obtained from the FEM data structures available within the CPU, which hold information such as cells, their faces, vertices, and neighbor information. However, using these data structures to compute the neighboring cells in the GPU presents two main challenges. First, these data structures are complex and will add unnecessary memory load when copied to the GPU. Second, computing these

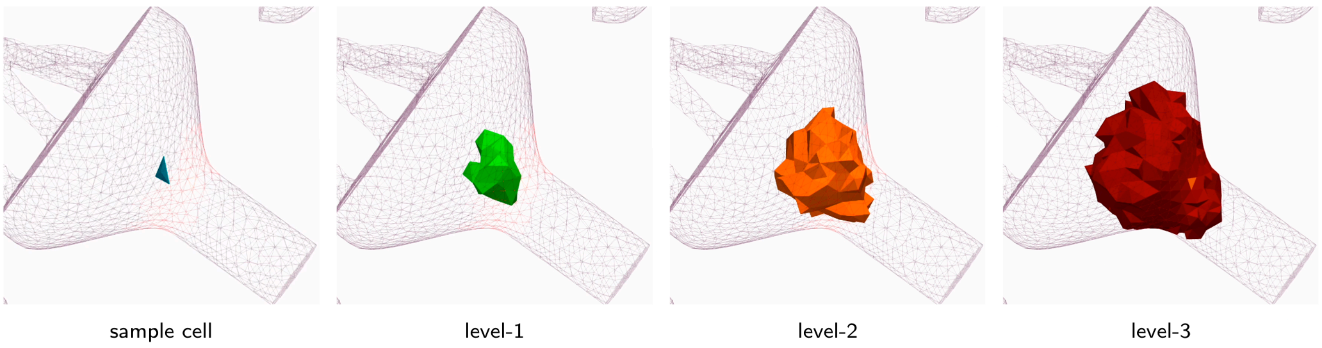

**Figure 4:** Zonal neighbors visualized at various zonal levels for a sample cell within the domain.

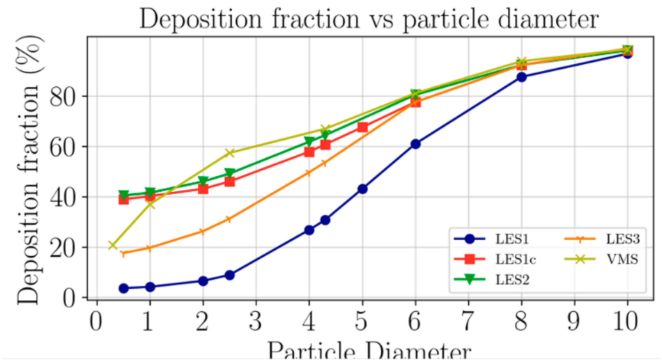

**Figure 5:** Deposition fraction for various particle sizes. VMS is the deposition fraction from our GPU accelerated model. All other models are from [12]

| Zonal Levels | Min | Median | Max |
|---|---|---|---|
| 1 | 14 | 61 | 114 |
| 2 | 30 | 210 | 558 |
| 3 | 31 | 447 | 1376 |

**Table 1**
Statistics of number of neighbouring cells for a given cell in the domain up to that zonal level.

neighbors on-the-fly within the GPU using these complex data structures is computationally expensive, as each GPU thread responsible for updating a particle's position must traverse numerous data structures to calculate the neighboring cells.

We compared our sequential implementation with an OpenMP-based multi threaded implementation and our GPU-based implementation. As shown in Tab. 3 and Fig. 6, the GPU implementation achieves a speedup of 100 times compared to the sequential implementation.

Precomputing this neighboring information externally and transferring it to the GPU is also memory-intensive. As seen from Table 1, each cell may have close to 1000 level-3 neighbors, resulting in storing thousands of cell IDs as neighboring information for each cell in a domain with millions of cells. This can create an array that uses a large amount of memory, making it challenging to transfer to the GPU. Therefore, we need a simple data structure that can be transferred to the GPU and used to perform minimal calculations in computing the zonal neighbors of the cell.

We observed that the raw mesh file, which provides information about which cells and vertices are connected, can be transformed into an adjacency matrix through proper calculations on the CPU. In this matrix, each row represents a cell, and the column values represent connectivity to other cells in the domain. Furthermore, the resulting adjacency matrix is sparse, allowing it to be stored in a Compressed Sparse Row (CSR) format. This format significantly reduces the memory footprint, making it easier to transfer to the GPU and use to calculate neighboring cells according to the algorithm shown in Algorithm 1. Each GPU thread will perform Algorithm 1 to compute the zonal neighbors of its respective particle's current cell.

| Reference | Mesh Size | Number of Particles | Time taken(s) per timestep |
|---|---|---|---|
| ANSYS Fluent | 0.2 Million | 100K | 0.6 |
| Our Implementation | 0.3 Million | 100K | **0.159** |

**Table 2**
Comparison of sequential timings.

| N_Particles | Sequential | OMP | CUDA | OMP Speedup | GPU Speedup |
|---|---|---|---|---|---|
| 100K | 0.219 | 0.01 | 0.002 | 21 | 88 |
| 1M | 2.166 | 0.139 | 0.022 | 16 | 98 |
| 2M | 4.445 | 0.18 | 0.044 | 25 | 102 |
| 5M | 11.101 | 0.433 | 0.101 | 26 | **110** |

**Table 3**
Time taken and speedup for various number of particles.

## 4. Results

To validate the efficiency of our sequential implementation, we compared the time our code takes to track particles (sequential implementation with one OpenMP thread) with ANSYS Fluent on a single CPU. This helps show that our code's baseline version is optimal when compared with the existing implementations. The table 2 shows the time taken for sequential execution of the ANSYS fluent module and our sequential implementation. It's important to note that ANSYS Fluent has its own way of creating meshes (meshing routines), So a mesh created within ANSYS with 200K elements was used for calculating the time taken for particle tracking. However, our in-house solver cannot work with meshes exported from ANSYS. Because of this, we used our own mesh (with 300,000 elements) generated externally and measured the particle tracking time. Both our code and ANSYS Fluent used the same number of particles. Even though ANSYS Fluent has fewer cells (elements) in its mesh, it still took longer to track particles than our framework. Additionally, we compared our deposition fraction results with those from [12], showing a strong correlation for larger particle sizes, as depicted in Fig. 5. For smaller particle sizes, a more refined mesh is necessary to accurately capture the fluid dynamics.

## Acknowledgements

The authors acknowledge Lokesh Mohanty (Indian Institute of Science, Bangalore) and Varun Vaidyanathan (Indian Institute of Technology, Delhi) for their assistance in conducting the experiments for this study.

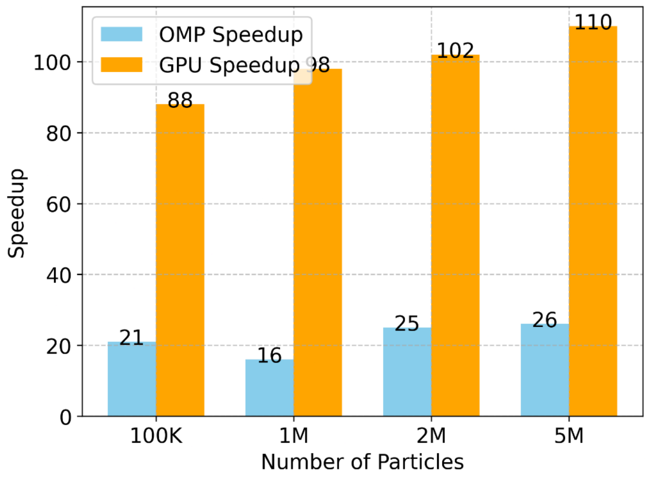

**Figure 6:** GPU vs OMP speedup for various number of particles.

**Algorithm 1** Breadth-First Search (BFS) for Neighboring Cells using CSR-based Adjacency Matrix for CUDA implementation

```
 1: // Create local arrays to store cells and depths for the queue
 2: int queue_cell_no[], queue_depth[], searched_cells[], queue_start_cursor ← 0, queue_end_cursor ← -1, searched_cells_counter ← 0
 3: // Add the current cell to the queue
 4: queue_end_cursor ← queue_end_cursor + 1, queue_cell_no[queue_end_cursor] ← cell_no, queue_depth[queue_end_cursor] ← 0
 5: while queue_start_cursor ≤ queue_end_cursor not inside_domain do
 6:     // Pop the cell from the queue
 7:     current_cell ← queue_cell_no[queue_start_cursor]
 8:     current_depth ← queue_depth[queue_start_cursor]
 9:     queue_start_cursor ← queue_start_cursor + 1
10:     // Loop through neighbour cells at current level
11:     start_index ← d_m_row_pointer[current_cell]
12:     end_index ← d_m_row_pointer[current_cell + 1]
13:     // For cells in current level
14:     for index ← start_index to end_index - 1 do
15:         neighbour_cell ← d_m_col_index[index] // Get the neighbor cell
16:         is_searched ← false
17:         // Check if the neighbor cell is already searched
18:         for k ← 0 to searched_cells_counter - 1
19:             if neighbour_cell = searched_cells[k]
20:                 is_searched ← true
21:                 break
22:         if is_searched then
23:             continue
24:         // Check if particle is inside current neighbour cell
25:         bool inside_neighbour_cell ← Is_Point_In_Cell_CUDA()
26:         searched_cells_counter ← searched_cells_counter + 1
27:         searched_cells[searched_cells_counter] ← neighbour_cell
28:         if inside_neighbour_cell
29:             inside_domain ← true // to get out of while loop
30:             d_m_previous_cell[tid] ← d_m_current_cell[tid] // update cell information
31:             d_m_current_cell[tid] ← neighbour_cell
32:             break
33:         if current_depth + 1 < search_depth
34:             // Add the neighbour cell to the queue to be searched
35:             queue_end_cursor ← queue_end_cursor + 1
36:             queue_cell_no[queue_end_cursor] ← neighbour_cell
37:             queue_depth[queue_end_cursor] ← current_depth + 1
38:     end for
39: end while
```

## References

[1] K. Kadota, A. Imanaka, M. Shimazaki, T. Takemiya, K. Kubo, H. Uchiyama, Y. Tozuka, Effects of inhalation procedure on particle behavior and deposition in the airways analyzed by numerical simulation, Journal Of The Taiwan Institute Of Chemical Engineers 90 (2018) 44–50. doi:10.1016/j.jtice.2017.11.008.

[2] J. Kimbell, R. Segal, B. Asgharian, B. Wong, J. Schroeter, J. Southall, C. Dickens, G. Brace, F. Miller, Characterization of deposition from nasal spray devices using a computational fluid dynamics model of the human nasal passages, Journal Of Aerosol Medicine 20 (2007) 59–74. doi:10.1089/jam.2006.0531.

[3] K. Inthavong, Z. Tian, J. Tu, W. Yang, C. Xue, Optimising nasal spray parameters for efficient drug delivery using computational fluid dynamics, Computers In Biology And Medicine 38 (2008) 713–726. doi:10.1016/j.compbiomed.2008.03.008.

[4] A. Gambaruto, E. Olivares, H. Calmet, G. Houzeaux, A. Bates, D. Doorly, Transport and deposition in the upper human airways during a sniff, in: Computational Engineering And Science For Safety And Environmental Problems COMPSAFE2014, Sendai (Japan), 2014.

[5] K. Cheng, Y. Cheng, H. Yeh, R. Guilmette, S. Simpson, Y. Yang, D. Swift, In vivo measurements of nasal airway dimensions and ultrafine aerosol deposition in the human

nasal and oral airways, Journal Of Aerosol Science 27 (1996) 785–801. doi:10.1016/0021-8502(96)00029-8.

[6] J. Kesavan, R. Bascom, B. Laube, D. Swift, The relationship between particle deposition in the anterior nasal passage and nasal passage characteristics, Journal Of Aerosol Medicine 13 (2000) 17–23. doi:10.1089/jam.2000.13.17.

[7] G. Kanapilly, O. Raabe, C. Goh, R. Chimenti, Measurement of in vitro dissolution of aerosol particles for comparison to in vivo dissolution in the lower respiratory tract after inhalation, Health Physics 24 (1973) 497–507. doi:10.1097/00004032-197305000-00004.

[8] Z. Zhang, C. Kleinstreuer, Computational analysis of airflow and nanoparticle deposition in a combined nasal–oral–tracheobronchial airway model, Journal Of Aerosol Science 42 (2011) 174–194. doi:10.1016/j.jaerosci.2011.01.001.

[9] G. Garcia, E. Tewksbury, B. Wong, J. Kimbell, Interindividual variability in nasal filtration as a function of nasal cavity geometry, Journal Of Aerosol Medicine And Pulmonary Drug Delivery 22 (2009) 139–156. doi:10.1089/jamp.2008.0713.

[10] E. Frederix, A. Kuczaj, M. Nordlund, M. Belka, F. Lizal, J. Jedelsky, J. Elcner, M. Jicha, B. Geurts, Simulation of size-dependent aerosol deposition in a realistic model of the upper human airways, Journal Of Aerosol Science 115 (2018) 29–45. doi:10.1016/j.jaerosci.2017.10.007.

[11] T. Gemci, V. Ponyavin, Y. Chen, H. Chen, R. Collins, Computational model of airflow in upper 17 generations of human respiratory tract, Journal Of Biomechanics 41 (2008) 2047–2054. doi:10.1016/j.jbiomech.2007.12.019.

[12] P. Koullapis, S. Kassinos, J. Muela, C. Perez-Segarra, J. Rigola, O. Lehmkuhl, Y. Cui, M. Sommerfeld, J. Elcner, M. Jicha, I. Saveljic, N. Filipovic, F. Lizal, L. Nicolaou, Regional aerosol deposition in the human airways: The siminhale benchmark case and a critical assessment of in silico methods, European Journal of Pharmaceutical Sciences 113 (2018) 77–94. doi:10.1016/j.ejps.2017.09.003.

[13] T. Miki, X. Wang, T. Aoki, Y. Imai, T. Ishikawa, K. Takase, T. Yamaguchi, Patient-specific modelling of pulmonary airflow using gpu cluster for the application in medical practice, Computer Methods In Biomechanics And Biomedical Engineering 15 (2012) 771–778. doi:10.1080/10255842.2011.560842.

[14] A. Kolanjiyil, C. Kleinstreuer, Computationally efficient analysis of particle transport and deposition in a human whole-lung-airway model. part i: Theory and model validation, Computers In Biology And Medicine 79 (2016) 193–204. doi:10.1016/j.compbiomed.2016.10.020.

[15] E. Olivares Mañas, Parallel lagrangian particle transport: application to respiratory system airways, Ph.D. thesis, Universitat Politècnica de Catalunya (2018).

[16] U. Wilbrandt, C. Bartsch, N. Ahmed, N. Alia, F. Anker, L. Blank, A. Caiazzo, S. Ganesan, S. Giere, G. Matthies, R. Meesala, A. Shamim, J. Venkatesan, V. John, Parmoon – a modernized program package based on mapped finite elements. doi:10.1016/j.camwa.2016.12.020.

[17] B. Pal, S. Ganesan, A finite element variational multiscale method for computations of turbulent flow over an aerofoil, Int. J. Adv. Eng. Sci. Appl. Math. 7 (1–2) (2015) 14–24. doi:10.1007/s12572-015-0126-1.