DOI: <https://doi.org/10.14311/TPFM.2022.013>

A PARALLEL ALGORITHM FOR FLUX-BASED BOUNDED SCALAR REDISTRIBUTION

Martin Isoz^{1,2}, Marie Plachá³

¹ Czech Academy of Sciences, Institute of Thermomechanics, Dolejškova 5, Prague 182 00, Czech Republic

² Department of Mathematics, Faculty of Chemical Engineering, University of Chemistry and Technology, Technická 5, Prague 166 28, Czech Republic

³ Department of Chemical Engineering, Faculty of Chemical Engineering, University of Chemistry and Technology, Technická 5, Prague 166 28, Czech Republic

Abstract

Let us assume a bounded scalar function $\alpha : Q = I \times \Omega \rightarrow \langle 0, 1 \rangle$, $I \subset \mathbb{R}$, $\Omega \subset \mathbb{R}^3$, where Q is an open bounded domain; and its discrete counterpart α^h defined on a computational mesh $Q^h = I^h \times \Omega^h$. The problem of redistribution of α^h over Ω^h ensuring the scalar boundedness while maintaining the invariance of $\int_{\Omega^h} \alpha^h dV$ is surprisingly frequent within the field of computational fluid dynamics (CFD). The present contribution is motivated by the case arising from coupling Lagrangian particle tracking and particle deposition within Ω^h with Eulerian CFD computation. We propose an algorithm for α^h redistribution that is (i) based on fluxes over the computational cells faces, i.e. suitable for finite volume (FV) computations; (ii) localized, meaning that a cell Ω_P^h with $\alpha_P^h > 1$ affects only its closest neighbors with $\alpha^h < 1$; and (iii) designed for parallel computations leveraging the standard domain decomposition methods.

Keywords: CFD, OpenFOAM, scalar redistribution.

1 Introduction

Solid material deposition and build-up is of importance in numerous industrial and natural processes ranging from sedimentation [1] via coal ash deposition [2] or air clean up towards biomass burning [3]. Hand in hand with the widespread presence and importance of solid material deposition comes the necessity to simulate it. Two main approaches to the solid phase deposition simulations seem to be prevalent. In the first one, the dispersed solid and the continuous fluid phases are approximated as interpenetrating continua giving rise to the class of Eulerian-Eulerian models, see e.g. the SedFoam solver developed for sediment transport [1]. In the second, Lagrangian-Eulerian approach, each solid particle is traced separately within the Lagrangian framework and the fluid is simulated as an Eulerian continuum, see e.g. [3].

Within the Eulerian-Eulerian approach, the effects of the solid phase on the flow are easily computable as both the phases are simulated via the same principles – Navier-Stokes equations extended by tracking the volume fraction of each phase in each cell (or element) of the computational mesh [1, 4]. On the other hand, if the effects of solids on the flow are of interest in the Lagrangian-Eulerian framework, the solids need to be somehow projected into the fluid computational domain. Such a projection is usually, similarly to the Eulerian-Eulerian approach, based on computing the volumetric fraction of solid phase in each computational cell.

In both the Eulerian-Eulerian and Lagrangian-Eulerian approaches to modeling of solid material deposition, it is of utmost importance to correctly compute the volumetric fraction of solid phase in each cell of the computational mesh. At the same time, in both the approaches, the solid phase volumetric fraction may become unbounded. In the Eulerian-Eulerian approach, the standard culprit is the numerical solution of advection equation used for the phase tracking [4]. In the Lagrangian-Eulerian approach, volume fraction of particles deposited in a mesh cell may exceed one if the solids are not re-projected into the fluid domain in each step of the Lagrangian solid phase tracking, which is usually computationally unfeasible.

In this contribution, we present a physics-motivated parallel algorithm for redistribution of a scalar function representing the volumetric fraction of a solid phase in a computational cell over the computational mesh to ensure its boundedness. In the following, we will first outline the algorithm fundamentals and show its defining properties. Next, we will discuss the algorithm

Figure 1: Illustration of the sought *localized* solid phase redistribution. (a) Computational mesh with inscribed non-zero values of α and the cell from which the solid phased has to be redistributed highlighted. (b) Unwanted algorithm behavior. (c) Required *localized* algorithm behavior.

behavior on unstructured non-orthogonal meshes and decomposed computational domain. The paper is concluded by a showcase usage of the algorithm in simulations of soot deposition inside monolithic catalytic filters in automotive exhaust gas aftertreatment system.

2 Redistribution algorithm

To describe the algorithm, let us first introduce the necessary notation. Let $\alpha : Q = I \times \Omega \rightarrow \langle 0, 1 \rangle$, $I \subset \mathbb{R}$, $\Omega \subset \mathbb{R}^3$ be a bounded scalar function and Q an open bounded domain. Furthermore, let us denote α^h the discrete counterpart of α defined on a computational mesh $Q^h = I^h \times \Omega^h$, where Ω^h is a finite volume spatial discretization of Ω . Finally, let Ω_P^h be a single cell of Ω^h , $\{f\}_P$ the faces of Ω_P^h and $\{\Omega_N^h\}_P$ its face-neighbors, i.e. the cells of Ω^h that share a face with Ω_P^h . Note that hereafter, we will concentrate solely on the discrete setting and the superscript h will be omitted.

The scalar field α stands for the volumetric fraction of a solid phase (SP) in each cell of Ω . Therefore, it represents the SP distribution inside the computational domain. The task of the solid phase redistribution is threefold, (i) SP needs to be redistributed in a physically-defensible manner; (ii) the algorithm needs to be computationally efficient and robust; and (iii) it needs to be easily parallelized within the domain decomposition framework, which is a standard for the current CFD codes.

First, SP redistribution is considered physically acceptable if it preserves the total mass of the solids in the system, i.e. if $m_s = \rho_s \int_{\Omega} \alpha dV$, where m_s is the SP total mass in Ω and ρ_s its density, is not in any way affected by the algorithm. Furthermore, the SP redistribution needs to be performed in a *localized* manner, see Fig. 1.

The second and third requirements on the algorithm behavior are strongly inter-connected. Furthermore, they are linked to the framework used for the numerical solution of the system governing partial differential equations. In this contribution, we assume that the system at hand is solved via a cell-centered variant of the finite volume method, which is used for example in OpenFOAM [5]. Furthermore, the code parallelization is assumed to be carried out utilizing the domain decomposition. Consequently, during a parallel computation, each processor solves a part of the overall problem defined on a subset of Ω and communicates with other processors via artificial processor-processor boundaries.

In particular, for the domain decomposition $\Omega = \bigcup_{i=1}^{N_p} \Omega^i$, where N_p is the number of available processor cores and Ω^i are non-overlapping subsets of Ω , the field α originally defined on Ω is decomposed in N_p independent fields α^i , $i = 1, \dots, N_p$, defined on the corresponding sub-meshes. Because of the α field decomposition, the information transfer between α^i and α^j defined on adjacent sub-meshes needs to be handled via special boundary conditions and no easy general access to α^j values from processors $i = 1, \dots, j-1, j+1, N_p$ is available.

Furthermore, due to the robustness, the code needs to take into account the possibility of $\alpha > 1$ in a cell Ω_P surrounded by cells with $\alpha \geq 1$ already, see Fig. 2a. In such a case, it is necessary to redistribute the solid phase through the already full neighbors of the cell Ω_P into its closest neighbor-neighbors as shown in Fig. 2b. However, in the given setting (FVM combined with domain decomposition), such a task is not straightforward. First, the standard finite volume formulation is based on the so-called compact stencil [6], i.e. all the utilized approximations regarding the cell

Figure 2: Illustration of the algorithm on a decomposed mesh. (a) The solid phase from the cell with $\alpha > 1$ needs to be redistributed through already full cells. (b) Access from Ω^i to the cell on Ω^j , outlined in red, is required but not available in the standard domain decomposition methods.

Ω_P stem from the values stored in Ω_P and in its face-neighbors $\{\Omega_N\}_P$. This is reflected in the computational codes that allow for easy and efficient access to Ω_P , $\{\Omega_N\}_P$, and the faces $\{f\}_P$ shared between Ω_P and $\{\Omega_N\}_P$. However, access from Ω_P to any cell not included in $\{\Omega_N\}_P$ is usually not readily available. Furthermore, if the mesh Ω is decomposed for a parallel computation, only the data in the cells directly adjacent to the shared processor-processor boundary $\partial\Omega^{ij}$ may be accessed from two neighboring sub-meshes Ω^i and Ω^j . In particular, the cell outlined in red in Fig. 2b cannot be accessed from Ω^i as it is in Ω^j and not adjacent to $\partial\Omega^{ij}$.

The proposed algorithm for solid phase redistribution is based on the concept of *face fluxes* used to alter the α values in both the face-neighbors of any given face $f \in \Omega$. This approach is natural for any FVM computations, can be easily generalized for unstructured non-orthogonal meshes, and preserves the compact stencil of the standard finite volume method in the sense that for a $\alpha_P > 1$ in Ω_P , only the face-neighbors of Ω_P ($\{\Omega_N\}_P$) are used for SP redistribution. Thus, if $\alpha_P > 1$ and $\alpha_N \geq 1$ in any of the cells in $\{\Omega_N\}_P$ and a bigger algorithm stencil is required (see Fig. 2), the computation is performed in an iterative manner.

Algorithm 1 Preparation of $\mathcal{L}_p$ and $\mathfrak{X}$ (performed on each sub-mesh separately).

Require: list of cells to be tested $\mathcal{L}_t$, tick = true, tolerance ε

```

1: for each  $\Omega_P \in \mathcal{L}_t$  do
2:   compute SP surplus:  $\varphi_P = \max\{0, 1 - \alpha_P\}$ ;
3:   if  $\varphi_P > 0$  then
4:     append  $\Omega_P$  to  $\mathcal{L}_p$ :  $\mathcal{L}_p \leftarrow \Omega_P$ ;
5:     if tick then
6:       for each  $\Omega_N \in \{\Omega_N\}_P$  do (only the cells on the current sub-mesh are tested)
7:         if  $\alpha_N > 1 - \varepsilon$  then
8:           append the face  $f_{PN}$  between  $\Omega_P$  and  $\Omega_N$  to  $\mathfrak{X}$ :  $\mathfrak{X} \leftarrow f_{PN}$ ;
9:         end if
10:      end for
11:    end if
12:  end if
13: end for
14: tick :=  $\neg$ tick;
15: Shuffle  $\mathcal{L}_p$  to suppress algorithm directionality:  $\mathcal{L}_p := \text{shuffle}(\mathcal{L}_p)$ ;
    return problem cells  $\mathcal{L}_p$  and no-flux faces  $\mathfrak{X}$ 

```

The algorithm itself requires a list $\mathcal{L}_t$ of cells in which $\alpha > 1$ can occur and may be split into three distinct parts, (i) preparation of a list $\mathcal{L}_p$ containing cells with $\alpha > 1$ and of a list $\mathfrak{X}$ with faces blocked-off from the SP redistribution due to being placed between two cells with $\alpha \geq 1$; (ii) generation of solid phase face fluxes Φ^s and update of data across the decomposed domain; (iii) application of Φ^s to redistribute the α field over the computational mesh Ω .

The first part of the algorithm dealing with the generation of lists $\mathcal{L}_p$ and $\mathfrak{X}$ is given in Alg. 1.

We go through the cells in $\mathcal{L}_t$. If $\alpha_P > 1$ in any $\Omega_P \in \mathcal{L}_t$, Ω_P is appended to a list of cells that need to be treated ($\mathcal{L}_p$). Simultaneously, α in the face-neighbors of Ω_P ($\{N\}_P$) is checked and if $\alpha \geq 1$ in any of the neighbors, the shared face is blocked-off for any solid phase transfer, see the line 8 of Alg. 1. Note that this blocking-off is performed only every other algorithm iteration (*tick*), which allows for SP transfer from cells surrounded by already full cells.

Algorithm 2 Preparation of Φ^s and data synchronization.

Require: list of problem cells $\mathcal{L}_p$ and no-flux faces $\mathcal{X}$
preparation of SP face-fluxes (done on each sub-mesh separately)

- 1: **for each** $\Omega_P \in \mathcal{L}_p$ **do**
- 2: initialize cell surface area available for SP transfer: $A_t := 0$;
- 3: initialize a list $\mathcal{F}_P$ of cell faces available for SP transfer;
- 4: **for each** $f \in \{f\}_P$ **do**
- 5: **if** $f \notin \mathcal{X}$ **then**
- 6: $A_t += \text{Area}(f)$;
- 7: append the face f to the list $\mathcal{F}_P$: $\mathcal{F}_P \leftarrow f$;
- 8: **end if**
- 9: **end for**
- 10: **if** $A_t > 0$ **then**
- 11: correct α_P : $\alpha_P -= 2\varphi_P$;
- 12: prepare SP redistribution on unstructured meshes: $\varphi_P *= \text{Volume}(\Omega_P)/A_t$;
- 13: **end if**
- 14: **for each** $f \in \mathcal{F}_P$ **do**
- 15: prepare SP flux through f : $\Phi_f^s := \varphi_P \text{Area}(f)$;
- 16: append the used face to no-flux faces: $\mathcal{X} \leftarrow f$;
- 17: **end for**
- 18: **end for**

data synchronization across the decomposed domain

- 19: **for each** processor-processor patch $\partial\Omega^{ij} \in \text{sub-mesh boundary } \partial\Omega^i$ **do**
- 20: **for each** $f \in \partial\Omega^{ij}$ **do**
- 21: assign Φ_f^s based on the data in Ω_N , f face-neighbor of Ω_P : $\Phi_f^s := \varphi_N \text{Area}(f)$;
- 22: **end for**
- 23: **end for**

return solid phase face fluxes Φ^s and updated no-flux faces $\mathcal{X}$

The solid phase face fluxes Φ^s are prepared in the second part of the algorithm, listed in Alg. 2. In each cell of $\mathcal{L}_p$, the algorithm walks through the cell faces and if the face is not blocked-off from SP transfer, it adds the face area to the total cell surface available for SP transfer (A_t) and appends the face itself to the list of cell transfer faces $\mathcal{F}_P$. Afterwards, the surplus solid phase φ_P is removed from the cells $\Omega_P \in \mathcal{L}_p$ and converted from volumetric fractions into absolute amounts. This conversion followed by dividing ϕ_P by A_t allows for computations on unstructured meshes, see the line 12 of Alg. 2.

The converted φ_P is used to compute the solid phase fluxes Φ_f^s through all the faces $f \in \mathcal{F}_P$ necessary to obtain $\alpha_P = 1$ (line 15 in Alg. 2). Furthermore, all the faces with non-zero Φ_f^s are blocked-off from the SP transfer in the next algorithm iteration, i.e. included in the list $\mathcal{X}$. This disallows for any SP back-fluxes through an already used face and, combined with the tick-tock nature of Alg. 1, ensures the required physically acceptable SP redistribution as illustrated in Figures 1 and 2.

At this moment, the data necessary to update the α -field in each sub-mesh Ω^i of Ω are stored both on cell faces as Φ^s and in cell centers as φ . This enables efficient data transfer between individual neighboring sub-meshes as for a face f on a processor-processor boundary $\partial\Omega^{ij}$, the finite volume method itself needs access to both its face-neighbors, despite the fact that they both lie in different sub-meshes. This access is leveraged to transfer information on the SP fluxes from one sub-mesh to another, see lines 19–23 in Alg. 2.

In the algorithm final part, given in Alg. 3, the previously prepared solid phase face fluxes Φ^s

Algorithm 3 Application of Φ^s to modify the α field (performed on each sub-mesh separately).

Require: solid phase face fluxes Φ^s

```

1: for each  $\Phi_f^s \in \Phi^s$  do
2:   update  $\alpha$  in the first  $f$  face-neighbor:  $\alpha_P += \Phi_f^s / \text{Volume}(\Omega_P)$ ;
3:   append the first  $f$  face-neighbor to the cells to be tested:  $\mathcal{L}_t \leftarrow \Omega_P$ ;
4:   if  $f$  is internal face then
5:     update  $\alpha$  in the second  $f$  face neighbor:  $\alpha_N += \Phi_f^s / \text{Volume}(\Omega_N)$ ;
6:     append the second  $f$  face-neighbor to the cells to be tested:  $\mathcal{L}_t \leftarrow \Omega_N$ ;
7:   end if
8: end for
9: make the list of cells to be tested unique:  $\mathcal{L}_t := \text{makeUnique}(\mathcal{L}_t)$ ;
return updated list of cells to be tested  $\mathcal{L}_t$ 

```

are used to alternate the α field. Let us recall, that the value of α in the cells in $\mathcal{L}_p$ was already modified on the line 11 of Alg. 2, where two times the surplus solid phase volumetric fraction was subtracted from it. Now, because the face flux Φ_f^s is used to update α in both the face-neighbors of f , one time φ_P is added back to each cell $\Omega_P \in \mathcal{L}_p$. Such an arrangement is adopted because it is actually faster to apply Φ^s to all the face-neighbors of the faces it is defined on than to identify which of the cells were originally in $\mathcal{L}_p$.

Algorithm 4 The overall solid phase redistribution algorithm for decomposed domain .

Require: initial list of cells to be tested $\mathcal{L}_t$, tick = true, tolerance ε

```

1: initialize the field  $\varphi := 0$  defined in each cell of  $\Omega^i \subset \Omega$ ;
2: initialize the field  $\Phi^s := 0$  defined on each face of  $\Omega^i \subset \Omega$ ;
3: set iterate = true;
4: while iterate do
5:   prepare the lists  $\mathcal{L}_p$  and  $\mathfrak{X}$  on each  $\Omega^i$  separately via Alg. 1;
6:   prepare the SP fluxes  $\Phi^s$  on each  $\Omega^i$  separately, synchronize data between the sub-meshes via Alg. 2;
7:   apply the prepared SP fluxes  $\Phi^s$  to update the SP volume fractions  $\alpha$  on each  $\Omega^i$  separately via Alg. 3;
8:   clear-out auxiliary variables:  $\Phi^s *= 0$  and  $\varphi *= 0$ ;
9:   if  $\mathcal{L}_p = \emptyset$  or maximum number of iterations is exceeded set iterate = false; endif
10: end while
11: discard the non-physical solid phase:  $\alpha = \max\{1, \alpha\}$ ;
return updated SP volume fractions  $\alpha$ 

```

Finally, the overall computational scheme is given in Alg. 4, where we embed the Algorithms 1–3 into an outer iteration loop. Because of the approach to the $\mathfrak{X}$ list construction, approximately each two outer iterations increase the stencil available for SP redistribution by one layer of cells (cell-neighbors→cell-neighbors-neighbors...). In an ideal case, the iterations should be repeated until there is no cell with $\alpha > 0$. However, in practice this may prove computationally unfeasible and it is necessary to limit the maximum number of algorithm iterations.

3 Algorithm behavior

The proposed algorithm behavior is illustrated on a number of proof-of-concept cases. First, in Fig. 3, we illustrate that for a uniform hexahedral mesh decomposed into four sub-meshes, the algorithm behaves as depicted in Figures 1 and 2. In particular, the tests shown in Fig. 3 were constructed in a way that in each time-step, an amount of solid phase corresponding to two-times the source cell volume (see Fig. 3) was introduced into it. Afterwards, the iterative loop given in Alg. 4 is repeated until all the solid phase is redistributed, which is followed by proceeding to a new time level and reintroducing new SP into the source cell.

In the case shown in Fig. 3a, the source cell was placed directly on the geometry wall and located one cell away from a processor-processor boundary. In the case of Fig. 3b, the source cell

Figure 3: Algorithm behavior on a uniform hexahedral mesh and a decomposed domain comprising four sub-meshes. (a) Solid phase deposition on a geometry wall, (b) SP deposition on a pre-existing solid layer. The deposited SP shapes are indicated via wireframe.

had the same position with respect to the processor-processor boundary but was located on a layer of cells with $\alpha = 1$. The depicted data correspond to the situation after four time steps, i.e. after introduction of eight times the cell volume of SP into the source cell. Note that (i) the algorithm is forced to re-distribute the solid phase across the processor-processor boundary, and (ii) problems with the algorithm *localized* behavior may be identified in Fig. 3b. These are caused by the fact that during the preparation of the list $\mathcal{X}$ listed in Alg. 1, the algorithm on the sub-mesh Ω^i does not have access to the data on its neighbors (lines 6-10).

Figure 4: Algorithm behavior on a (a) uniform hexahedral mesh, (b) unstructured polyhedral mesh, and (c) unstructured tetrahedral mesh. The amount of solid phase in all the three domains corresponds to $\int_{\Omega} \alpha \, dV = 0.078$.

The next test was performed on three different meshes. The first mesh was the same as in the first test, while the second and the third mesh were unstructured. The second mesh comprised general polyhedra. The third mesh consisted of tetrahedra and was polydual to the second one. Once more, all the meshes were decomposed into four sub-meshes and a SP volume corresponding to two times the volume of the hexahedral cell in the mesh one was added to a source cell located close to the geometry center.

The test results are given in Fig. 4a, b and c, respectively. Note that the overall topology of the domain with $\alpha > 0$ (Ω_{α}) is, up to the differences caused by the used meshes, similar for all the three cases. Furthermore, note that all the three displayed Ω_{α} span all four submeshes comprising Ω . Together, the results shown in Fig. 4 illustrate the algorithm usability for computations on decomposed unstructured meshes, which is a crucial property for its practical usage.

As the final note on the proposed algorithm behavior, let us comment on its computational cost. The authors are aware of the existence of an algorithm similar to the proposed one implemented as `packingLimiter.H` function in the `sedFoam` solver [1]. However, the existing `packingLimiter.H`

algorithm is not capable of redistributing the solid phase over a decomposed domain. Furthermore, it sees only the immediate face-neighbors of the cell with $\alpha > 1$ and does not check the amount of solid phase in these neighbors. Because of the additional controls in the new algorithm, each of its outer iterations (see the while loop in Alg. 4) takes approximately twice the time compared to the `packingLimiter.H`. Furthermore, each iteration in the aforementioned while loop further increases the algorithm computational cost but at the same time, it enlarges its stencil available for the SP redistribution. In practical applications, it is usually desirable to keep the maximum number of the outer algorithm iterations below five.

4 Application in a pore-scale soot accumulation model for automotive particulate filters

One of the illustrative applications of the proposed algorithm lies within the field of modeling of automotive exhaust gas aftertreatment. Combustion engine generates an exhaust gas comprising a number of pollutants, one of which is the particulate matter (PM) produced by non-ideal fuel oxidation. The particulate matter particles are removed from the exhaust gas in PM filters (PF), whose filtration performance is tightly linked with the structure of its porous walls. Recently a number of three-dimensional computational models of filter porous walls was proposed, see e.g. [7, 8].

The presented algorithm allows for using the Lagrangian tracking of fine PM particles inside the computational domain to generate a semi-permeable zone representing the PM in the Eulerian CFD mesh. The PM deposition in this case corresponds with Figure 3. In the real application, the computational domain Ω represents a small section of the PF wall and is generated based on processed XRT images of real PF samples as described in detail by Koci et al. [9]. The domain Ω is divided as $\Omega = \Omega_p \cup \Omega_c \cup \Omega_s$, where Ω_p are the free pores, Ω_c is the catalytic layer and Ω_s represents the deposited particulate matter, i.e. $\alpha > 0$ in Ω_s . Note that $\Omega_p \cap \Omega_c \cap \Omega_s = \emptyset$.

Figure 5: Simulated evolution of a catalytic filter filtration efficiency during its operating time. (a) Image of the clean filter wall with the catalytic material shown in yellow and flow streamlines colored according to the velocity magnitude, (b) detail of the filter pore at $t = 0$ h, (c) detail of the filter pore at $t = 3.1$ h with the deposited soot shown in red.

In order to obtain physically relevant simulation results, Ω needs to represent a significant portion of the PF porous wall. Consequently, the resulting FVM meshes comprised roughly $2 \cdot 10^8$ computational cells and the computations had to be performed in parallel. To be computationally feasible, the calculations were performed as follows: (i) The flow inside Ω was computed via FVM-based CFD with Ω_c and Ω_s acting as a porous zones, which results in a velocity field $\mathbf{u}$. For details see [9]. (ii) The velocity field $\mathbf{u}$ was used in the Lagrangian particle tracking to simulate the PM movement inside the PF, where Ω_c and Ω_w acted as an instantaneous particle traps [7] and Ω_s acted as a probabilistic particle trap with the probability of capture in a cell $\Omega_P \in \Omega_s$ equal to α_P . The values of $\alpha \in \Omega$ were updated based on the number of captured particles. (iii) After a significant PM deposition inside Ω , the flow was recomputed and $\mathbf{u}$ was updated.

Note that the flow and particle movement simulations are actually decoupled. However, such an approach is possible because the PM taken into account represent soot particles generated by gasoline engines [7]. One reconstructed section of a gasoline catalytic filter wall was selected in order to illustrate the PM deposition. The clean filter wall together with flow streamlines can be seen in Fig. 5a. The filtration efficiency dependence on PF operation time presented in Fig. 5 shows a gradual increase from 20 % towards the complete filtration efficiency of 100 % reached at time 3.1 h. The particulate matter build up is depicted in Figure 5b and c. More comprehensive description of this application and comparison with experimental data is under development.

5 Conclusion

In the present contribution, we described an algorithm for solid phase redistribution over a finite volume computational mesh. The algorithm can be used in parallel computations and redistributes the solid phase in a physically defendable manner. The algorithm behavior was illustrated on a number of small cases. The practical usability of the algorithm was demonstrated via its application in a solution of the complex problem of modeling soot filtration and accumulation inside particulate matter filters in automotive exhaust gas aftertreatment systems. The proposed algorithm and the presented examples are available at <https://github.com/MartinIsoz/spRedist>.

Acknowledgment

The work was financially supported by the Czech Science Foundation (GA19-22173S), by the institutional support RVO:61388998, from the grant of Specific university research – grant No A1.FCHI.2021.004 and by the Centre of Excellence for Nonlinear Dynamic Behaviour of Advanced Materials in Engineering CZ.02.1.01/0.0/0.0/15_003/0000493 (Excellent Research Teams) in the framework of Operational Programme Research, Development and Education. Marie Plachá thanks Johnson Matthey Technology Centre in Sonning Common for the support and feedback.

References

- [1] Chauchat, J., Cheng, Z., Nagel, T., Bonamy, C. & Hsu, T.-J.: SedFoam-2.0: a 3-D two-phase flow numerical model for sediment transport. *Geoscientific Model Development*. vol. 10 no. 12: (2017). pp. 4367–4392. doi: 10.5194/gmd-10-4367-2017.
- [2] Brink, A., Lindberg, D., Huppa, M., de Tejada, M. E., Paneru, M., Maier, J., Scaffknecht, G., Prancitelli, A. & Pourakashanian, M.: A temperature-history based model for the sticking probability of impacting pulverized coal ash particles. *Fuel Processing Technology*. vol. 141: (2016). pp. 210–215. doi: 10.1016/j.fuproc.2015.08.039.
- [3] Chapela, S., Porteiro, J., Gomez, M., Patino, D. & Miguez, J.: Comprehensive CFD modeling of the ash deposition in biomass packed bed burner. *Fuel*. vol. 234: (2018). pp. 1099–1122. doi: 10.1016/j.fuel.2018.07.121.
- [4] Rusche, H. *Computational fluid dynamics of dispersed two-phase flows at high phase fractions*. PhD thesis: Imperial College of Science, Technology and Medicine: (12 2002).
- [5] OpenCFD. *OpenFOAM: The Open Source CFD Toolbox. User Guide Version 1.4*, OpenCFD Limited. Reading UK: (2007).
- [6] Moukalled, F., Darwish, M. & Mangani, L. *The finite volume method in computational fluid dynamics: an advanced introduction with OpenFOAM and Matlab*. Springer-Verlag: Berlin, Germany: 1 edition: (2016). ISBN 978-3-319-16874-6.
- [7] Placha, M., Koci, P., Isoz, M., Svoboda, M., Price, E., Thompson, D., Kallis, K. & Tsolakis, A.: Pore-scale filtration model for coated catalytic filters in automotive exhaust gas aftertreatment. *Chemical Engineering Science*. vol. 226: (2020). doi: 10.1016/j.ces.2020.115854.
- [8] Belot, I., Vidal, D., Votsmeier, M., Hayes, R. E. & Bertrand, F.: Numerical investigation of the impact of washcoat distribution on the filtration performance of gasoline particulate filters. *Chemical Engineering Science*. vol. 221: (2020). page 115656. doi: 10.1016/j.ces.2020.115656.
- [9] Koci, P., Isoz, M., Placha, M., Arvajova, A., Vaclavik, M., Svoboda, M., Price, E., Novak, V. & Thompson, D.: 3d reconstruction and pore-scale modeling of coated catalytic filters for automotive exhaust gas aftertreatment. *Catalysis Today*. vol. 320: (2019). pp. 165–174. doi: 10.1016/j.cattod.2017.12.025.