

Documentation for “multiBreath.py” Python Multiscale Coupling Code

Andrew Kuprat, Pacific Northwest National Laboratory, October 30, 2024.

Applicability: This code was used in the papers [[Kuprat 2021](#)] [[Kuprat 2023](#)] to couple the OpenFOAM 3D Computational Fluid Particle Dynamics code [[OpenFOAM v2112](#)] to the 1D Multiple-Path Particle Dosimetry Model (MPPD) [[Anjilvel 1995](#)] [[Asgharian 2022](#)].

Contents: The gzipped directory **multiBreath.10.30.2024.tgz** contains python code, input files, and mesh for a demonstration computation for a coarse mesh version of the geometry in [[Kuprat 2021](#)]. (In addition to the python code, the 1D portion of the computation requires java MPPD code which is not the standard available online version MPPD V 3.04 [[MPPD V 3.04](#)], but is a modified version available at [[MPPD SimTK](#)]. This modified version does 1D MPPD deposition modeling and also generates and reads disk files for multiscale coupling. For the 3D portion of the computation, the OpenFOAM driver MPPICFoam is used.)

The python contents in the top directory (“run directory”) include:

multiBreath.py—python driver that is submitted as a SLURM job on linux. I.e., ‘sbatch multiBreath.py’

Other python files:

call1D.py --- code that calls MPPD.jar which is the java JAR file that contains the 1D MPPD code

comm3D.py --- code that generates OpenFOAM input files.

data.py --- code that contains data for the run. This is essentially the input file for the multiscale run.

post3D.py --- code called after a full 3D OpenFOAM breath phase (i.e., after inhale or after exhale)

pre3D.py --- code called before the next 3D OpenFOAM breath phase.

simclass.py --- definition of the class which holds simulation data.

multiBreathExcel.py --- code that generates an Excel spreadsheet from multiscale simulation results.

How to use this code:

0. We assume computation is done on a linux platform such as exists at the time of this writing on the PNNL cluster constance.pnl.gov with **slurm** parallel job scheduler.
1. Set up environment:
 - a. Set the simulation environment in **appenv.sh** (See example in run directory and adjust for your version of gcc, openmpi, java, MPPICFoam)
 - b. source **appenv.sh**
2. Set up MPPD:
 - a. Download ‘MPPD3D1D Source 28 Apr 2022.zip’ located at [[MPPD SimTK](#)] which contains the modified version of MPPD for multiscale computation. Access permission required.
 - b. Create MPPD.jar file from ‘MPPD3D1D Source 28 Apr 2022.zip’ by typing ‘source **makeMPPD**’.

- c. Include *.dat files of the form 'pnnl<name1D item>.dat' (ie pnnloutlet1.dat, pnnloutlet2.dat,..) These files have MPPD subtree data for each subtree. For demonstration example, *.dat files are already in run directory. (*Note: alteration of topology from provided demonstration example requires alteration of custom MPPD code due to some hard-coded assumptions in the java code.*)
3. Edit data.py to give run parameters:
 - a. **reuse3D** is set to Boolean True or False as follows:
 - i. **False**: do 3D computation for inhale/exhale along with 1D computation
 - ii. **True**: Assume last 3D computation is converged and simply repeat 1D computation for future breaths.
(In the following, we assume reuse3D=False.)
 - b. **dtio** is assigned output timestep interval. dtio=0.01 is standard
 - c. **nbreaths** is assigned total number of breaths to be simulated (including both joint 3D/1D computation and 1D-only computation).
 - d. **inhales** is a list of inhalation time intervals used in joint 3D/1D computation
 - e. **exhales** is a list of exhalation time intervals used in joint 3D/1D computation
 - f. **inhaleMasses** is a list of masses inhaled during each inhalation in joint 3D/1D computation. If reuse3D=True, the final value in this array will be amount injected in 1D-only breath computation.
 - g. **tstart** is start of current job submission. If everything goes perfectly, this is tstart=0.00 and then the multiscale job is executed in one fell swoop. Unfortunately, in joint 3D/1D computation, something may crash and then multiscale job may have to be resubmitted with tstart increased to the last successfully computed output time. If reuse3D=True, the value of tstart should equal the time of end exhale in the last breath produced by joint 3D/1D computation or the end exhale time of a subsequent 1D-only breath.
 - h. **tend** is end target of current job submission. In joint 3D/1D computation this is usually the end time of the last exhale, but it could be the end time of any inhale or exhale. (In the latter case, a future submission would carry on with tstart set to the previous tend value, and tend incremented to a newer time target.) If reuse3D=True, tend should be an end exhale time greater than tstart.
 - i. **VolFRCinMI** is a list of volumes of each MPPD subtree in milliliters
 - j. **particleDiameter** is diameter of injected particles in MKS units.
 - k. **particleDensity** is density of injected particles in MKS units.
 - l. **params1D** is a dictionary which defines MPPD parameters:
 - i. **segmentsPerAirway** which is segments per airway in the MPPD simulation
 - ii. **MixingModel** which is: (0=alveolar mixing model off; 1=alveolar mixing model on)
 - m. **params3D** is a dictionary that defines 3D OpenFOAM parameters:
 - i. **inhaleParcelsPerSecond** is number of parcels per second injected by MPPICFoam during inhale.
 - ii. **maxCoParticle** is maximum particle Courant number. This should be less than one. It has been observed that on occasion the 3D code will freeze due to a particle getting 'stuck'. If the multiscale job is resubmitted with tstart increased to a time less than when the 'bad particle' may have been released, altering

- maxCoParticle (by decreasing or increasing but staying less than one) may allow a slightly different particle path which will move the simulation forward.
- iii. **maxCoFlow** is maximum Courant number of Navier-Stokes flow timestep. This can be greater than one, since OpenFOAM 'pimplefoam' (merged PISO-SIMPLE) algorithm is used in MPPICFoam. It should be as big as possible, but no greater than that. Typical values are 5-20. If set too large, the 3D integration will become unstable and will crash. If set too small, the 3D code will take too long to compute on a large geometry.
 - iv. **maxDeltaT** is maximum delta T for integration. Typically timestep will be limited by the maxCoFlow condition, but if it is not, then having a reasonably small maxDeltaT will allow accuracy to be maintained.
 - n. **output3Dfile** is name of output file for whole multibreath simulation. Best to keep this equal to 'Lung.out'. **Lung.out should not be altered during the multibreath simulation, since it is analyzed to compute amounts injected.**
 - o. **error3Dfile** is error output generated by slurm during the multiscale simulation. Typically 'Lung.err'
 - p. **ngen** is maximum number of generations in each subtree including 3D portion. E.g. if longest path addresses generation 0,...,23, then ngen=24.
 - q. **name3D** is list of outlet subtree names used by 3D code. These are descriptive form, e.g., 'outRB5'
 - r. **name1D** is list of subtree names used by 1D MPPD code. These are standard form 'outletXX', where XX is a 1-based index.
 - s. **depPatch** is a list of wall deposition patches in the 3D geometry.
4. Set up the 3D input files:
 - a. The 3D geometry **constant/polyMesh** must have wall patches given in depPatch (above), single inlet patch 'inlet', and outlet patches as in name3D (above).
 - b. For the first time '0', two files '**p**' and '**U.air**' are set as initial conditions for the MPPICFoam solver for pressure and flow. Set them in the format of the example.
 - i. For p, wall patches (in depPatch), are 'type zeroGradient'
 - ii. For p, outlet patches (in name3D) are 'type zeroGradient'
 - iii. For p, inlet patch is 'uniformTotalPressure' with p0=0
 - iv. For p, internalField is set to 'uniform 0'
 - v. For U.air, wall patches are 'type fixedValue; value uniform (0 0 0);'
 - vi. For U.air, 'inlet' patch is 'type pressureInletOutletVelocity;'
 - vii. For U.air, outlet patches are 'type flowRateInletVelocity; volumetricFlowRate tableFile; file "flow_*.txt"; outOfBounds 'clamp'; (Note: on inhale multiBreath.py will change this boundary condition to 'flowRateOutletVelocity' on all processor copies. This causes outlet flows to not be plug flows but to have faster core flow as is appropriate for an exit boundary condition. Then on exhale, 'flowRateOutletVelocity' will be changed back to 'flowRateInletVelocity' on all processor copies, so that flows into domain are plug flows.)
 - viii. For U.air, internalField is "uniform (0 0 0)"
 - c. Define for each outlet a file "**flow_*.txt**" which specifies the volumetric flow at the outlets as specified in 0/U.air. Note from the sample that we currently have a 0.01s

ramp at the beginning and end of inhale/exhale phases. The end of the ramp is not zero, but some tiny value (currently 1.e-9), because MPPICFoam can crash if instructed to end at exactly zero velocity.

- d. Set the inlet injection profile '**particle_dmdt_inlet_profile.txt**' as in the example. The example has a 0.01s ramp that goes from 0=no injection to 1=full injection at the beginning and end of each inhale. It is zero during the entire exhale.
 - e. Set up '**constant**' directory
 - i. Use example files in constant directory.
 - ii. In '**constant/kinematicCloudProperties**',
 1. Define inlet, outlet, and wall patches to match the **polyMesh** geometry. Wall patches have 'type stick' and inlet/outlets patches have 'type escape'.
 2. At bottom, include all inlet/outlets/wall patches in cloudFunctions/patchPostProcessing1/patches subdictionary
 - f. Set up '**system**' directory
 - i. In **decomposeParDict**, set '**numberOfSubdomains**' to number of processors that will be used to run the problem.
 - ii. In **controlDict**, at bottom of file, define system/controlDict/functions dictionary to contain '#includeFunc flowRatePatch' functions as in example. Wall patches get


```
#includeFunc flowRatePatch(name=<WallPatchName>,kinematicCloud:massStick,executeControl=writeTime,writeControl=writeTime)
for each distinct <WallPatchName>
For 'inlet' there is
#includeFunc flowRatePatch(name=inlet,phi.air,executeControl=writeTime,writeControl=writeTime)
#includeFunc
flowRatePatch(name=inlet,kinematicCloud:massEscape,executeControl=writeTime,writeControl=writeTime)
For each outlet patch <outlet patch name>, there is
#includeFunc flowRatePatch(name=<outlet patch
name>,phi.air,executeControl=writeTime,writeControl=writeTime)
#includeFunc flowRatePatch(name=<outlet patch
name>,kinematicCloud:massEscape,executeControl=writeTime,writeControl=writeTime)
```
5. Edit **multiBreath.py** lines starting with #SBATCH to give relevant SLURM parameters for your job submission. See example. Edit so that number of nodes multiplied by number of tasks per node is equal to numberOfSubdomains specified in system/decomposeParDict. Also in multiBreath.py define python variable **numcores** to equal numberOfSubdomains specified in system/decomposeParDict.
 6. Perform parallel decomposition at time=0 with 'decomposePar -time 0'.
 7. Run multiBreath in parallel with SLURM using 'sbatch multiBreath.py'
 8. Observe run. Then,
 - a. If particle gets stuck, resubmit with tstart adjusted to be just prior to when particle may have been introduced into 3D domain. Slightly alter 'maxCoParticle' as described above. Do not alter Lung.out logfile.
 - b. If timeout occurs, resubmit at last output time that was achieved. Do not alter Lung.out logfile.

9. Run outputs suitable for conversion to excel for each breath phase are in **multiResults/**<breath number>_inh and multiResults/<breath number>_exh. <breath number> is zero-based, so for 3 breaths the directories are **0_inh, 0_exh, 1_inh, 1_exh, 2_inh, 2_exh**.
10. Copy to Microsoft windows, for last achieved breath phase, '**sim.yaml**' file in breath phase output directory. E.g., for exhale of 3rd breath output, copy the file multiResults/2_exh to windows. Also copy the files data.py and simclass.py to windows from run directory.
11. Customize excel output to be created by editing **multiBreathExcel.py**:
 - a. **SubtreeClusters** contains desired list of subtree clusters. E.g., cluster subtrees by lobes. Subtree numbers are 1-based.
 - b. **SubtreeClusterNames** gives corresponding names for subtree clusters. E.g. lobe names
 - c. **GenClusters** contains desired list of generation clusters. E.g. cluster conducting, proximal respiratory, and distal respiratory airways.
 - d. **GenClusterNames** gives corresponding names for generation clusters. E.g. 'Gen4-10'.
 - e. **SubtreeGenCoClusters** gives list of order pairs of (subtree, generation) clusters. E.g. [[[3,9,11],[20]]] is a list containing the single ordered pair [[3,9,11], [20]] which corresponds to desired graph showing data occurring in cluster of subtrees 3, 9, and 11, occurring in generation 20. This is optional and allows fine scale inspection of results.
 - f. **SubtreeGenCoClusterNames** gives corresponding names for SubtreeGenCoClusters.
12. On Microsoft windows machine (with WSL linux prompt), run 'python multiBreathExcel.py'. This creates excel output.

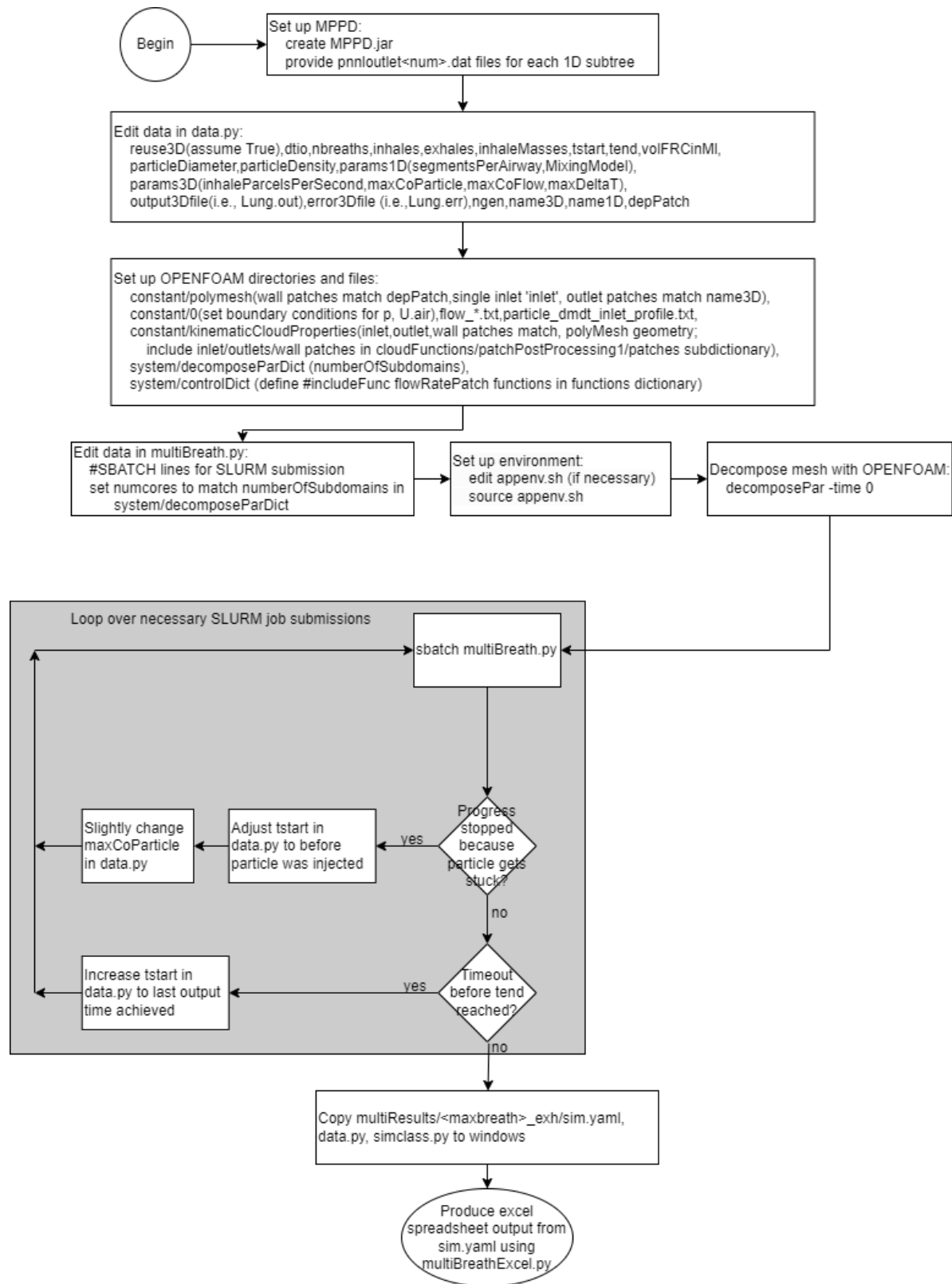


Figure 1: Flowchart of user actions required for running multiBreath 3D/1D coupling simulation.

Structure of the multiBreath code

The python driver is **multiBreath.py**. It is presumed to run as a parallel job under **slurm**:

sbatch multiBreath.py

data.py contains SOME of the user-settable parameters. Ideally all the parameters could be set in this file, but the decision was made to have some 3D parameters set in FOAM directories and associated text files. In particular, you need to set the time dependent flows in the subtrees in text files in the run directory. For the current set-up with 11 subtrees, these files are called **flow_RB1.txt**, **flow_RB2.txt**, etc. These are read in by the FOAM dictionaries in **0/U.air** to set inlet/outlet air flows. Also, a relative profile of particle injection **particle_dmdt_profile.txt** should be supplied for particle injection into the whole model. For the current case, it ramps from 0 to 1 in the first 0.01s of inhalation and ramps down from 1 to 0 in the last 0.01s of inhalation. It remains zero during exhalation. (This text file contains absolute injection times which the python code uses to create a text file **particle_dmdt_inlet.txt** which is relative profile of injection relative to start-of-injection (**SOI**) time of current inhalation.)

Also, the following information is supplied in the FOAM directories:

0/p --- pressure BC

0/U.air --- airflow BC

constant/g --- gravity force vector

constant/kinematicCloudProperties ---- particle BC

constant/transportProperties --- air properties

constant/turbulenceProperties --- (Was set to laminar in current set-up. If turbulence were used, BC for turbulence fields would have to be supplied in '0' directory.)

constant/polyMesh --- The 3D mesh directory containing the **boundary**, **faces**, **neighbour**, **owner**, and **points** files. In particular the names of boundary surfaces in the **boundary** file must match the names used in **data.py**

system/controlDict --- most of this is set by **data.py**, but '**#includeFunc**' statements as detailed in other document must be inserted. These have been inserted in current set-up.

system/decomposeParDict --- **numberOfSubdomains** must match **numcores** in **multiBreath.py**

system/fvSchemes --- standard discretization schemes for pimplefoam incompressible CFD solution

system/fvSolution --- choice of solution algorithms for pimplefoam incompressible CFD solution and Lagrangian particle tracking.

These OpenFOAM parameters, once set, do not need to be changed. The parameters that are often changed due to restarts, extra breaths, change to 1D only computation, are specified in **data.py** and have been previously described in this document.

The MPPD executable **MPPD.jar** is made by executing the **makeMPPD** file. **pnnloutlet<num>.dat** files corresponding to all subtrees are required.

Python structure of multiBreath code

multiBreath.py is called using **slurm** as '**sbatch multiBreath.py**'. **multiBreath.py** contains **slurm** **#SBATCH** statements that control the **slurm** submission.

data.py contains user settable parameters as detailed previously.

multiBreath.py creates a '**sim**' object that contains simulation data. The **sim** class is defined in **simclass.py**. For an initial run, array storage is allocated for number of breaths required. For restart runs, a **sim.yaml** file is read from last computed breath and then array storage for additional requested breaths is allocated.

multiBreath.py contains a loop that executes each breath. The following applies to the case of 3D/1D hybrid computation (**reuse3D=False**). Since arbitrary restarts are allowed, integration in the loop can start at the beginning or the middle of the 3D inhalation, it can start at the end of inhalation where the MPPD code is called, or it can start in the middle of the exhalation. (The end of inhalation is assumed to be the beginning of exhalation and for this start time, the MPPD code will always be called.) The very first time **multiBreath** is invoked (i.e., time=0), it is assumed that breathing starts at beginning of inhalation. An end time can be specified for end of an inhalation, but it is usually set to end of an exhalation as this would be the end of a complete breath.

In the breathing loop, for 3D inhalation computation, **pre3D.pre3Dinhale** method is called. This calls **comm3D.genparticle_dmdt_inlet** to generate particle injection data relative to start-of-injection (**SOI**) at beginning of inhale. It also calls **comm3D.edientry** to edit the 3D FOAM file **system/controlDict** to set 3D FOAM parameters for current inhale. It calls **comm3D.edientry** to edit the 3D FOAM file to edit particle parameters in **constant/kinematicCloudProperties**, including specifying that include file **inletInjections.H** should be read. It also edits each **processor*/<time>/U.air** file to have 3D/1D interface boundary condition '**flowRateOutletVelocity**'. Then MPI is called in parallel on **numcores** to run MPPICFoam. The **slurm** output file (usually called **Lung.out**) contains important injection data and should not be edited.

post3D.post3Dinhale calls **comm3d.readinjection** to get **sim.massInhaled** by scanning through **Lung.out** to get record of particle injections from inlet. (This is an awkward aspect of the OpenFOAM code. Particle hits on outlets are easily read from files in the **postProcessing** directory that is created. However, particle injection data is not available in this way and must be deduced from reading text data generated in the log file **Lung.out**.) **post3D.post3Dinhale** obtains airflow exit data and particle exit data from the **postProcessing** directory and inserts it into the '**sim**' object using the **post3D.simupdate** routine. Due to possibility of multiple restarts, **post3D.simupdate** orders files in **postProcessing** in inverse chronological order and pieces together the most recent computations of exit airflow and exit particle mass to be inserted into the '**sim**' object. **post3D.post3Dinhale** writes **airvol_pmass_outlet<num>.csv** files giving recorded exit airflows and exit particle masses that will be read by MPPD code. **post3D.post3Dinhale** then calls **call1D.call1D** which in turn calls the MPPD code for each subtree. **call1D.call1D** then reads output files from MPPD code and inserts 1D exit particle masses and 1D generational suspended and deposited masses back into '**sim**' object.

In the breathing loop, for 3D exhalation computation, **pre3D.pre3Dexhale** method is called. This calls **comm3D.genparticle_dmdt_outlets** to generate particle injection data relative to start-of-injection (SOI) at beginning of exhale. It also calls **comm3D.edientry** to edit the 3D FOAM file **system/controlDict** to set 3D FOAM parameters for current exhale. It calls **comm3D.edientry** to edit the 3D FOAM file to edit particle parameters in **constant/kinematicCloudProperties**, including specifying that include file **outletInjections.H** should be read. It also edits each **processor*/<time>/U.air** file to have 3D/1D interface boundary condition '**flowRateInletVelocity**'. Then MPI is called in parallel on **numcores** to run MPPICFoam. Then **post3D.post3Dexhale** method is called. This reads particle deposition and particle and airflow exit data (through original inlet) from **postProcessing** directory. It then uses **post3D.simupdate** to update the '**sim**' object with this data. Then **post3D.post3Dexhale** calls **yaml.dump** to write a **sim.yaml** file in the directory **multiResults/<num>_exh**, where <num> is number of breath, with '0' being the first breath.

Structure of 'sim' object in sim class

A sim object contains arrays **sim.x**, where x are the following arrays that give data computed or used by the simulation and which have entries for each output time t.

time[t] --- starting from 0 and increasing by dtio (output interval, currently 0.01) until end of simulation.

massInhaled[t] --- particle mass inhaled into 3D inlet, obtained by parsing 3D log file.

mass3Ddeposited[patch,t] --- particle mass deposited on each 3D surface patch.

mass3Dsuspended[t] --- particle mass suspended in 3D domain

specflowrate[outlet,t] --- flow rate that is user-specified for each outlet

specairvols[outlet,t] --- time integral of specflowrate which cycles between 0 and tidal volume.

recflowrate3D[outlet,t] --- air flow rate actually recorded exiting each outlet

mass3Dto1D[outlet,t] --- particle mass that has flowed from 3D into 1D domain

mass1Dto3D[outlet,t] --- particle mass that has flowed back out from 1D domain to 3D domain

mass1Ddeposited[outlet,gen,t] --- particle mass that has deposited in outlet subtree in generation gen.

mass1Dsuspended[outlet,gen,t] --- particle mass suspended in outlet subtree in generation gen.

massExhaled[t] --- particle mass that has been exhaled from 3D inlet.

In addition the sim object contains the entries **sim.x**, where x does not change during the course of an inhale or exhale:

multiTimeCompleted --- Initially set to 0 or restart time, this is incremented at end of 3D inhale or 3D exhale to end-inhale or end-exhale time.

multTimePartial --- Initially set to 0 or restart time, this is incremented at end of 3D inhale (with MPPD call) to future end-exhale time, reflecting the fact that partial data on the upcoming exhale is known

because MPPD computes both inhale and exhale phases at once. At end of 3D exhale, multiTimePartial is also set to end-exhale time.

output3Dfile --- name of slurm log file (usually Lung.out). It is critical to not edit or delete this file, as injection data can only be obtained by scanning this file.

currbreath --- current breath number. Starts with 0.

loadedhistoricsimfile --- In case of restart after at least one phase (inhale or exhale) of breath is completed, a restart will load the sim object with the latest sim.yaml file. The name of this file is stored in this variable.

tstart --- the start time for run. 0 at beginning, or tstart > 0 in case of restart.

tend --- the requested end time for run. In general can be set to any time > tstart, but best practice to assure output is to set it to end-exhale time of current or future breath.

dtio --- the time interval between 3D output dumps as well as time granularity of data transfer between 3D and 1D codes. Currently set to 0.01s

inhales --- List of [tbegin,tend] pairs giving start & end times for inhalation.

exhales --- List of [tbegin,tend] pairs giving start & end times for exhalation.

ndist --- number of distal outlets.

ngen --- number of airway generations used by MPPD code.

nDepPatch --- number of deposition surface patches used in 3D code.

params1D --- dictionary of parameters relevant to 1D MPPD code

params3D --- dictionary of parameters relevant to 3D CFPD code

particleDiameter --- diameter of particles (MKS units)

particleDensity --- density of particles

inhaleMasses --- List of injection masses for each inhale breath

parcelsPerKilo --- Derived parameter giving number of computational parcels per kilogram of particles. Derived using user-specified inhaleParcelsPerSecond.

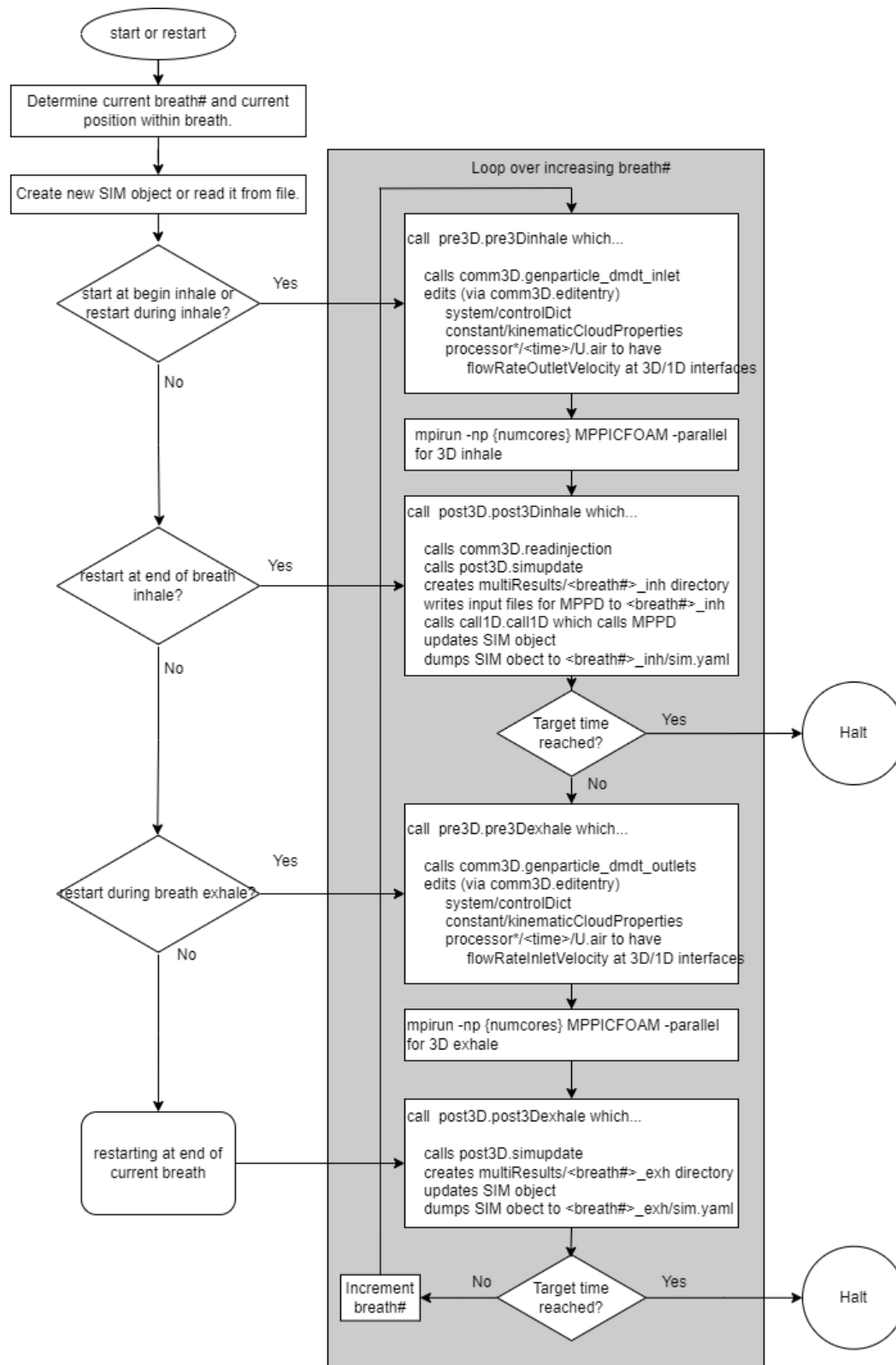


Figure 2: Flowchart of multiBreath 3D/1D coupling algorithm.

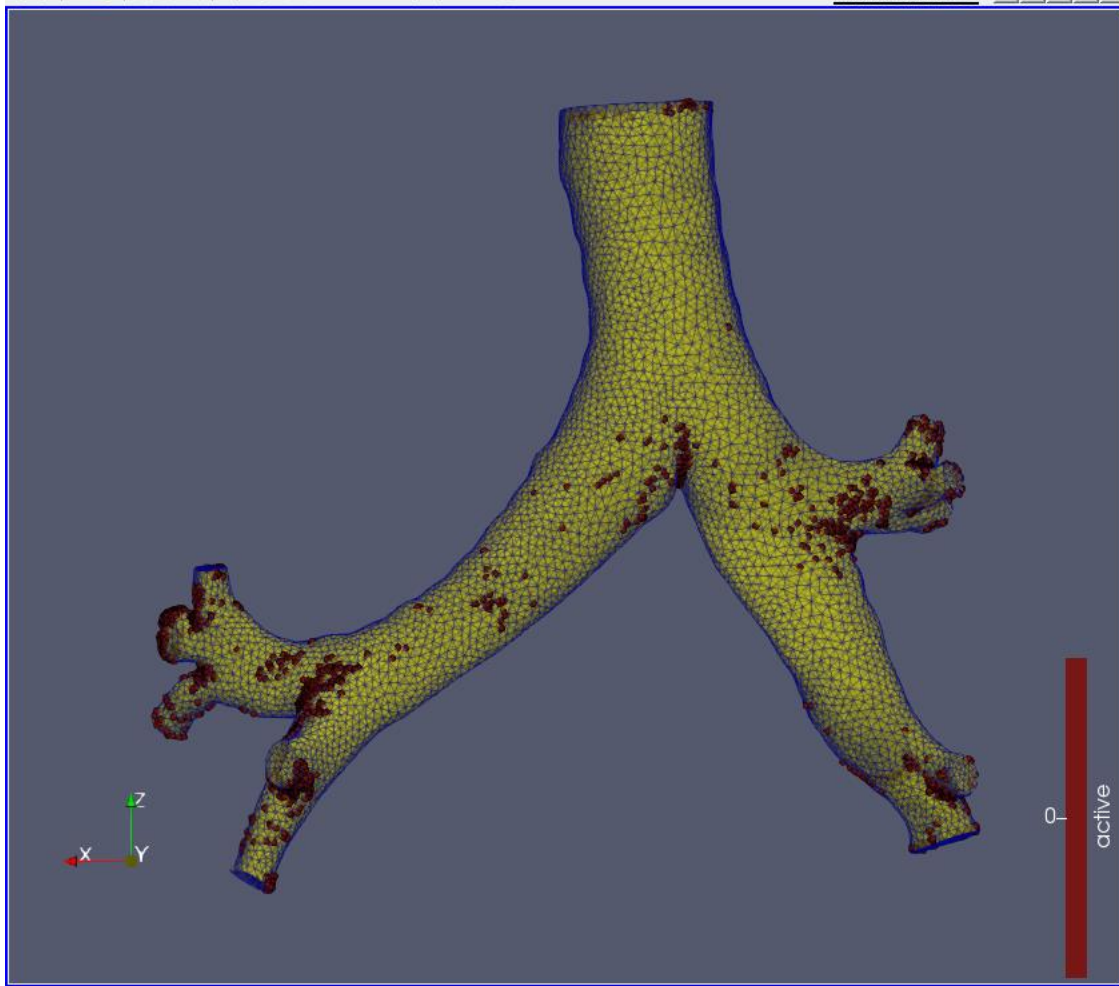


Figure 3. Low resolution mesh used for demonstration computation. Excessive particle deposition (red) at entrance and exits of 3D domain due to insufficient resolution of boundary layer. Visualization using ParaView 5.6.0.

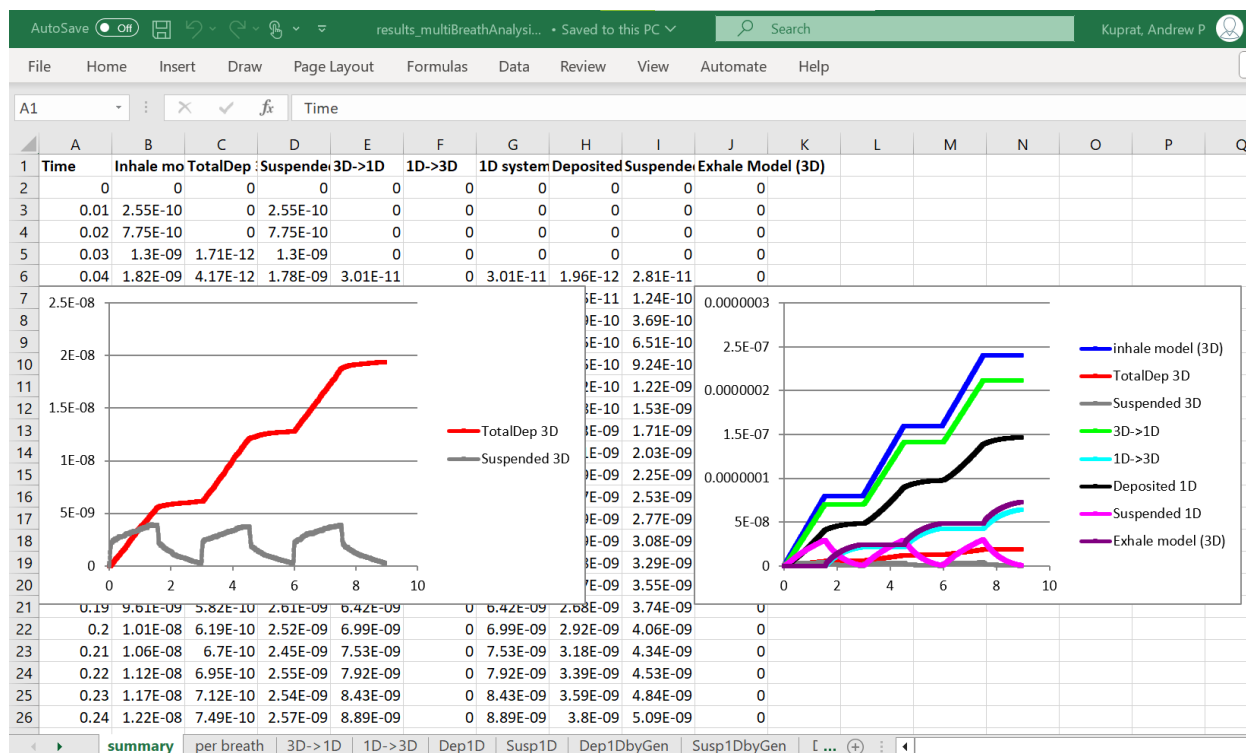


Figure 4. Screenshot of output spreadsheet produced using multiBreathExcel.py showing fate of particles in multiscale 3D/1D domain.

References

1. [Anjilvel 1995] Anjilvel S., Asgharian B. A multiple-path model of particle deposition in the rat lung. *Fundam.Appl.Toxicol.*, 28 (1995), pp. 41-50, <https://doi.org/10.1006/faat.1995.1144>
2. [Asgharian 2022] Asgharian B., Price O., Borojeni A.A.T., Kuprat A.P., Colby S., Singh R.K., ..., Darquenne C. Influence of alveolar mixing and multiple breaths of aerosol intake on particle deposition in the human lungs. *Journal of Aerosol Science*, 166 (2022), Article 106050, <https://doi.org/10.1016/j.jaerosci.2022.106050>
3. [Kuprat 2021] Kuprat A.P., Jalali M., Jan T., Corley R.A., Asgharian B., Price O., Singh R.K., Colby S., Darquenne C. Efficient bi-directional coupling of 3D computational fluid-particle dynamics and 1D Multiple Path Particle Dosimetry lung models for multiscale modeling of aerosol dosimetry. *J Aerosol Sci*, 151 (2021), Article 105647, <https://doi.org/10.1016/j.jaerosci.2020.105647>
4. [Kuprat 2023] Kuprat A.P., Price O., Asgharian B., Singh R.K., Colby S., Yugulis K., Corley R.A., Darquenne C. Automated bidirectional coupling of multiscale models of aerosol dosimetry: validation with subject-specific deposition data. *Journal of Aerosol Science*, 174 (2023), Article 106233, <https://doi.org/10.1016/j.jaerosci.2023.106233>
5. [MPPD SimTK] <https://simtk.org/docman/view.php/2083/13996/MPPD3D1D+Source+28+Apr+2022.zip>
6. [MPPD V 3.04] <https://ara.com/mppd>
7. [OpenFOAM v2112] <https://www.openfoam.com>