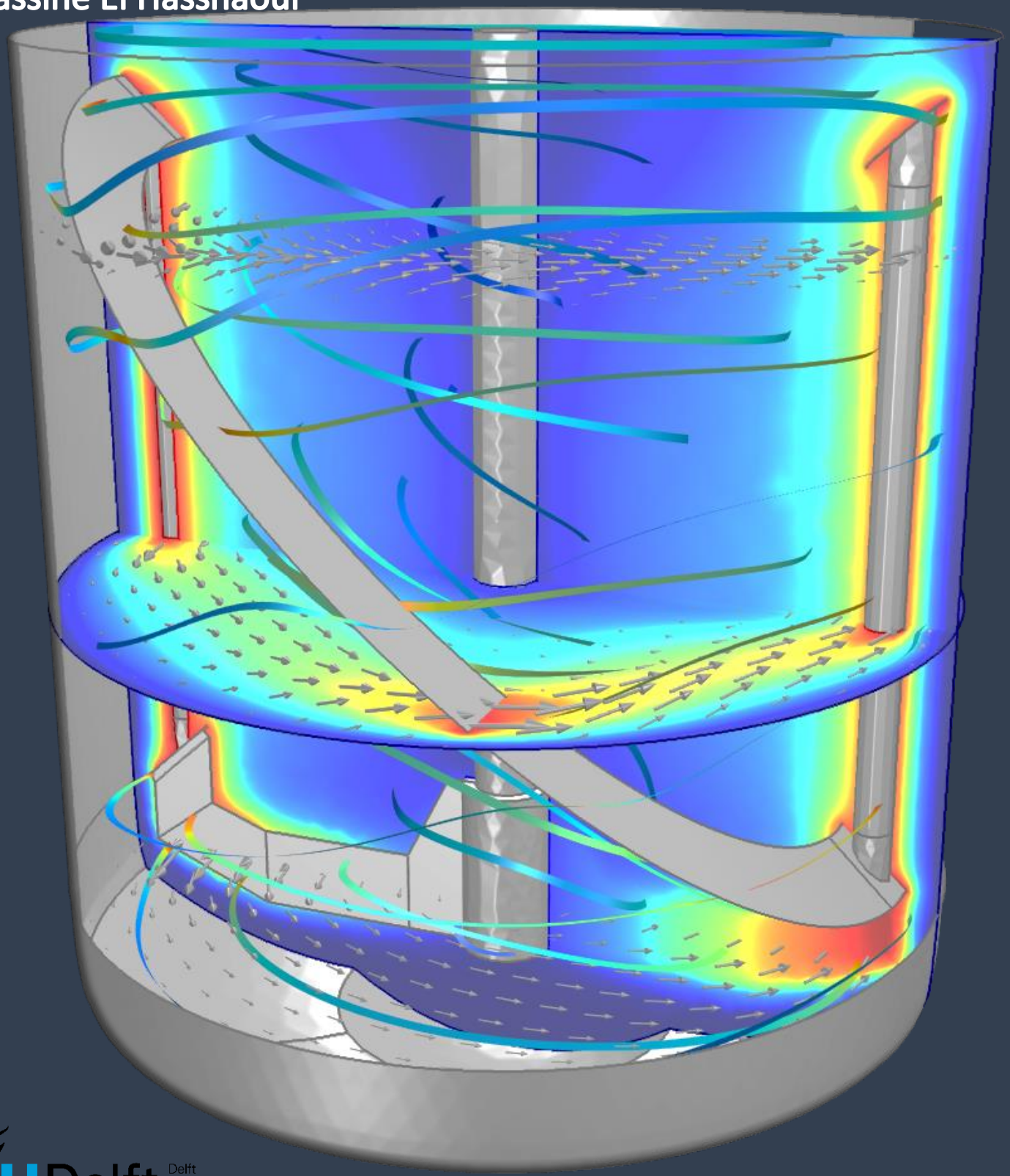
The computational power problem: a compartment model solution

Automating the dynamic compartmenting of a stirred-tank reactor

Master Thesis Project

Yassine El Hassnaoui



TU Delft Delft University of Technology

The computational power problem: a compartment model solution

Automating the dynamic compartmenting of a stirred-tank reactor

By

Yassine El Hassnaoui

In partial fulfilment of the requirements for the degree of

Master of Science

in Life Sciences and Technology with an emphasis in Biochemical Engineering
at the Delft University of Technology
To be defended publicly on Wednesday May 11, 2022 at 14:30 PM

Student number:	4282019
Supervisor:	MSc. L. Puiman
Thesis committee:	Dr. C. Haringa
	Prof. A. Straathof
	Dr. R. Gonzalez Cabaleiro
Project duration:	August, 2021 – May, 2022
Faculty:	Faculty of Applied Sciences, Delft

Cover: taken from ace-chn.com

Table of Contents

List of symbols

$\boldsymbol{\psi}$	Convective flux	$m^3 s^{-1}$
$\bar{\varepsilon}_L$	Average liquid holdup	
$\vec{v}$	Velocity vector	
$\vec{\alpha}_f$	Face area vector	
$\dot{m}$	Mass flow rate	$kg s^{-1}$
ρ	Density	$kg m^{-3}$
α_{phase}	Phase area	m^2
$\lvert\vec{\alpha}_f\rvert$	Magnitude of face area vector	m^2
$E_{k,turb}$	Turbulent kinetic energy	$m^2 s^{-2}$
$\bar{E}_{k,turb}$	Average turbulent kinetic energy	$m^2 s^{-2}$
$\alpha_{fraction}$	Phase fraction	

Abstract

Syngas fermentation modelling is complex to achieve: from hydrodynamics and kinetics to particle tracking and metabolic analyses, a lot of computational calculations are necessary in order to emulate real-world situations as closely as possible. The computational workload is generally such that simulations run weeks at a time, thereby greatly reducing the work efficiency of users. A contemporary solution is the compartmentalizing of the bioreactor into volumetric regions of assumed homogeneous hydrodynamic parameters, i.e. the axial liquid velocity.

This work builds on existing compartment strategies by proposing two compartmenting methods, the sorting method and the allocation method. The sorting method approach relies on extracting CFD data and subsequently compartmenting it, but did not produce satisfying results. On the contrary, the dynamic compartmenting allocation method compartments the model in-CFD. Complexity is increased by adding the ability to automatically choose the relevant hydrodynamic parameter and the automatic determining of the homogeneity factor. The dynamic compartments are formed with geometric compartments as baseline, comparing hydrodynamic parameters of the geometric compartments with the range set automatically by the homogeneity factor. Merging of geometric compartments is thereby achieved in a two-step automatic process: the most significant hydrodynamic parameter is determined automatically and the homogeneity factor is the result of automatic calculations. The dynamic compartments indicate that indeed a more detailed look has to be taken into the regions directly around the impellers of the bioreactor and regions at the boundary walls.

The allocation method was capable of reducing the number of geometric compartments by factors of 7.63 (in the case of 5000 initial geometric compartments), 12.05 (in the case of 10000 initial geometric compartments) and 19.49 (in the case of 50000 initial geometric compartments). The great reduction in numbers of compartments in the system translates to improvements of the computational workload. Furthermore, the geometric compartmenting model was successfully modularized through the use of nested functions in order to improve the usability and user friendliness. The nested functions script was subsequently validated through mixing time studies, in which both the base model and the modularized model showed a mixing time of 60.2 s for the A36R6T1 system and 54.8 s for the A18R2T1 system. Both mixing times are underestimations of the real-world mixing time, by respectively 16.4 % and 23.9 %.

Future research can improve on the dynamic compartmenting allocation method by firstly confirming that small volume compartments are not present, but if they are then the elimination of the compartments should be done. Secondly, the homogeneity factor should be fine-tuned so the number of dynamic compartments obtained by the model can be a direct user input. Thirdly, it is imperative that the allocation method is validated through mixing time studies. And lastly, the gas-phase is very important for modelling syngas fermentation and its implementation should, therefore, be prioritized in future work.

Introduction

Demand for energy

Since the 18th century industrial revolution mankind has endeavored to innovate and expand the usage of power-driven machinery. Various sectors of the economy and society were impacted profoundly, from increased agricultural production to increased demand for different essential and non-essential items. The technological and industrial progress hereby ushered in an era of great economic growth and the appearance of the capitalistic way of thinking (Lloyd-Jones, R. & Lewis, M., 1998). The environmental impact of such growth began to be apparent when oil and gas were discovered and extensively used worldwide. From that moment onward greenhouse gas emissions skyrocketed to unprecedented heights and its detrimental effects on the environment became apparent (Roser, M. & Ritchie, H., 2020). After a downtick in demand for oil and gas and their respective derivatives during the covid-19 pandemic the global demand for energy is recovering and even surpassing previous figures (IEA, 2021). Hereby, the amount of CO_2 that is expected to be released into the atmosphere during the year 2021 also increases, indicating that fossil fuels are filling in the difference between the 2020 supply and the new demand (IEA, 2021). Even though fossil fuels will still be around for many more decades (Kuo, G., 2019), their impact on the environment and negative influence on climate change necessitates research into alternative solutions. Energy sources like wind, solar, hydro and geothermal have their benefits but equally share a big disadvantage: an intermittent nature (Nanda, S. *et al*. 2014). Furthermore, the above-mentioned renewable resources do not affect the environment in an exclusively positive way (Nat. Ac. Press, 2007; USEIA, 2020; DiPippo, R., 2016).

Syngas fermentation

Another alternative would be to convert carbon-rich compounds, like lignocellulosic wastes, but also directly CO_2, into synthetic gas (or syngas). Advantages of syngas as an energy carrier compared to electricity are that especially H_2 has a very large calorific value; HHV = 141.7 MJ/kg and LHV = 120.0 MJ/kg (Eng. Toolbox, 2007). Moreover, syngas can subsequently be converted into ethanol (Istiqomah, N.A., 2021), which is a liquid fuel and thus more easily managed, transported and distributed. Syngas fermentation has, compared with similar processes in the chemical industry branch, the unique advantages that irrespective of the quality the whole biomass can be used, no pre-treatment and enzymes are needed, higher specificity of the biocatalyst, the H_2:CO ratio does not have an effect on the bioconversion rate, no metal poisoning happens and the bioreactor can operate at ambient, non-extreme conditions (Munasinghe, P.C. & Khanal, S.K., 2010). There are two major challenges, however, the principal one being the insignificant gas-to-liquid mass transfer properties of the individual syngas component gasses, which subsequently leads to the other challenge: low ethanol yields (Monir, M.U. *et al.* 2020). Several attempts have been made in order to improve the bioethanol yield of the syngas fermentation process, from optimizing the cell growth media (Benevenuti, C. *et al.* 2020) to the addition of secondary chemicals (Xiang, Y. *et al.* 2022). Improvements to the bioreactor design have also been considered in order to increase the mass transfer of the substrates (Bredwell, M.D. *et al.* 2008).

Computer models

Augmenting the ability of researchers to improve bioreactor designs through the introduction of computer models allows for a more detailed and focused approach. By modeling the bioreactor, certain parameters, like the superficial gas velocity and the gas holdup, can be simulated and tweaked in order to find an optimum which would result in higher fermentation rates. Moreover, getting insights into "the concentration changes that cells experience while flowing" through the bioreactor (Siebler, F. *et al.* 2019) is useful especially in tackling the issue of the limited mass transfer across the gas-liquid interface. In attempting to model syngas fermentation in a bubble column, a thermodynamics-based blackbox model has been proposed (Belacazar, E.A. *et al.* 2020) as well as a kinetic model for syngas in a stirred-tank reactor (Ruggiero, G. *et al*. 2022). Moreover, inclusion of microbial metabolics into computer models has been attempted (Li, X. *et al*. 2018). Furthermore, attempts have been made to produce a hydrodynamic model of the bioreactor (Heindel, T.J. & Kadic, E. 2014) and models for flow characteristics and particle tracking (Delafosse, A. *et al.* 2015). In conclusion, modeling all aspects of syngas fermentation in a bioreactor has not been achieved yet.

Computational power

Simulation techniques based on Computational Fluid Dynamics (CFD) are the main tools for mathematically and computationally simulating flows within a bioreactor. CFD has allowed researchers to simulate real-world scenarios in order to preempt possible problems and tackle challenges before the scaling up of the process (Haringa, C. 2022). Combining CFD with relevant physical and kinetic models makes it possible to understand and finetune input parameters like the gas bubble size and initial gas flow velocity, which have an impact on the mixing performance inside the bioreactor. Different interfaces are commercially available, like Ansys Fluent and COMSOL Multiphysics, which have easy-to-use graphical user interfaces (GUI's). Simulating flows and kinetics in a non-steady-state situation, while also simulating impellers and their effects on the flow. Using a fine mesh requires great computational power and takes a lot of time to complete. Moreover, power/energy consumption by computer hardware which runs iterative CFD for extended lengths of time is reaching the extent that running the hardware will be costlier than owning it (Mittal, S. 2014). The past decades have seen a great increase in computational power needs as well as great improvements in high-performance computing (figure 1). The trend is losing steam, however, which necessitates alternative, more creative ways of using the available computing power.

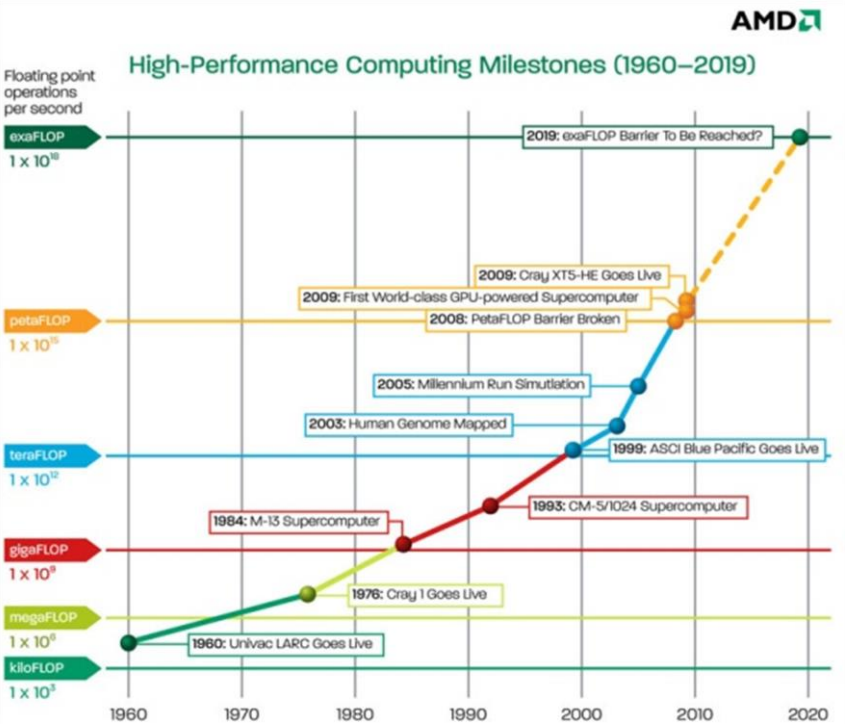


***Figure 1. High-performance computing milestones**. The past decades have shown an exponential increase in computing power, especially until 2009 when the first GPU-powered supercomputer was unveiled. The increasing trend is, however, losing its exponential trait, which suggests that alternative methods are needed which instead make creative use of the available computing power. [Taken from Mims, C. (2012).]*

Compartment models

One such alternative is the compartmentalization method (CM), whereby the bioreactor is assumed to be divided into homogeneous sub-volumes (Oosterhuis, N.M.G. 1984). Homogeneity is especially assumed in terms of concentrations and liquid/gas flow velocities. Using sub-volumes, or compartments, instead of individual mesh grid cells drastically reduces the number of computations needed to perform a simulation. Therefore, a great increase in performance in terms of energy consumption and time are observed (Jourdan, N. *et al.* 2019). The homogeneous compartments represent a simplification while they reduce the computational demand, thus leading to a trade-off between simplification and ease of computation. The balance between the two sides of the trade-off depends on the research question and what is sought to be achieved. Delafosse *et al*. were the first to achieve a comprehensive CFD-based compartment model, whereby a single-phase stirred-tank reactor (STR) was subdivided into a network-of-zones (Delafosse, A. *et al.* 2014). The network-of-zones is type of compartmentalization in which the bioreactor is divided into geometric compartments which are equally spread along the radial, tangential and axial directions (figure 2). Each geometric compartment is assumed homogeneous and is viewed as an individual entity with which calculations are performed. Other research groups, like Weber *et al*. also managed to successfully implement their interpretation of the network-of-zones strategy (Weber, B. *et al*. 2019), see Table 1 for an overview of some of the most recent compartment models. Since 2019 a strategy to automatically determine the compartmentalization of the bioreactor was shown by Tajsoleiman *et al.* (Tajsoleiman, T. *et al.* 2019), which builds on the network-of-zones method. The automatic method subdivides the bioreactor into many geometric compartments, called the geometric working space, which are used as the basic

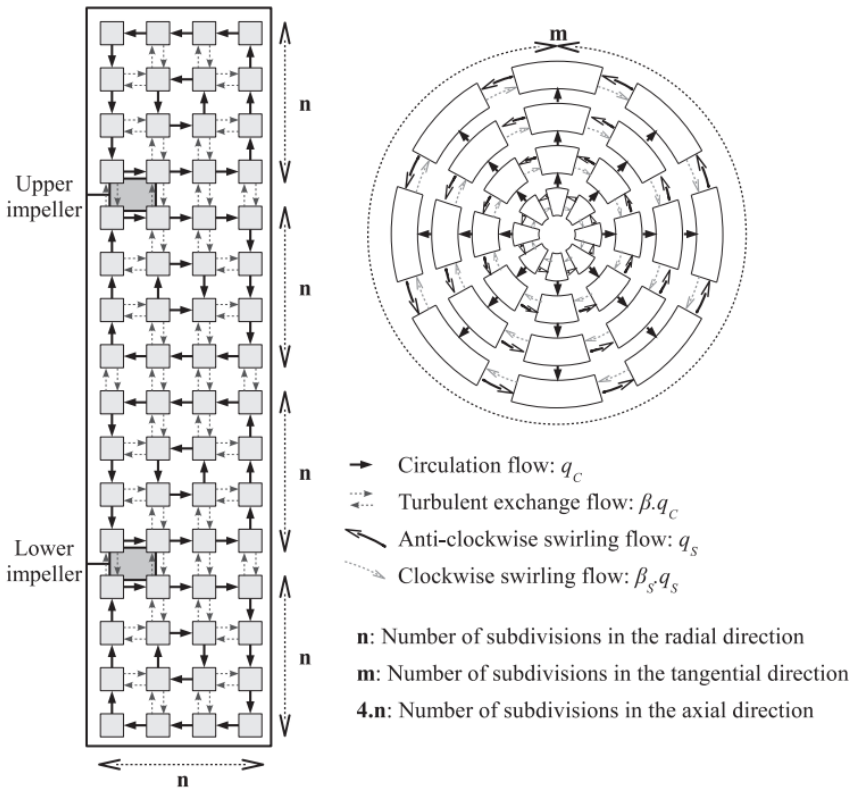


Figure 2. Network-of-zones compartment model*. The basic premise of the network-of-zones model is that the bioreactor is divided into equal numbers of parts in radial, tangential and axial directions. These equal parts, the zones, are subsequently treated as individual units with which calculations can be performed. Taken from Delafosse, et al. (2014).*

elements for calculations. From the geometric compartments the radial and axial velocities are derived and compared with a homogeneity factor. If the velocities of two neighboring geometric compartments lie within the range set by the homogeneity factor, the two neighbors are merged. The merging is based on the assumption that neighboring regions with approximately the same magnitude for the hydrodynamic parameter (i.e. the axial and radial velocities) are homogeneous in nature. The homogeneity factor is, therefore, the main tool to optimize for the number of final compartments, since the smaller the range to which the homogeneity factor caters, the fewer neighboring regions are merged and therefore the larger the number of compartments in the final result.

A multitude of CFD-based compartment models have been developed over the years, each with its niche of usage and limitations (table 1). Most of the models are on a stirred-tank reactor in single-phase, while using (parts of) the velocity field as tactic to make compartments. Even though most models are about the network-of-zones type of compartments, already some groups managed to improve into a more dynamic type of compartmenting. The dynamic compartment models are recognizable by their employment of a homogeneity tolerance factor, which determines whether two subcompartments are to be merged or not. Furthermore, the most occurring limitation in the models is the lack of a multiphase component. Although most models make compartments using the velocity field, not all models manage to take into account the third dimension of the velocity, i.e. the circumferential or z-direction velocity. Furthermore, the most occurring limitation in the models is the lack of a multiphase component, which is required for modelling syngas fermentation.

Table 1. Overview of compartment models devised by different research groups.

Authors	Type of Reactor	Phase	Type of Compartments	How Compartments are made	Limitation(s)
Delafosse, A., et al. (2014)	STR	Single-phase	Network-of-zones	Using velocity field from CFD	Lacks complexity and detail. Not multiphase. No circumferential velocity.
Tajsoleiman, T. et al. (2019)	STR	Single-phase	Acceptable homogeneity tolerance	Using axial and radial velocities	Not multiphase. No circumferential velocity.
Oner, M. et al. (2019)	STR	Single-phase	Hypothesis-driven: using experimental data	Using axial and radial velocities. Boundaries are where flow direction changes	Requires a lot of experimental data, thus very specific usage. Not multiphase.
Norregaard, A. et al. (2019)	STR	Single-phase	Hypothesis-driven: using experimental data	Using axial, radial and circumferential velocities	Requires a lot of experimental data, thus very specific usage. Not multiphase.
Spann, R. et al. (2019)	STR	Single-phase	Recirculation loop in velocity profile	Boundaries are where flow direction changes	Not multi-phase. Compartments are assumed homogeneous, but are not. Only axial flows are mentioned.
Weber, B. et al. (2019)	Bubble column	Multi-phase, but no gas-phase variables	Network-of-zones	Estimated using axial and radial velocities	No circumferential velocity.
Nadal-Rey, G. et al. (2021)	STR fed-batch	Multi-phase, but no gas-phase variables	Acceptable homogeneity tolerance, taking into account volume changes	Using axial and radial velocities and step-wise increase of volume. Hereby ignoring the gas-phase.	Assumption that flow does not change if volume increases with steps of ≤ 10 %. No circumferential velocity taken into account.
This research	STR	Single-phase, but with gas-phase variables	Acceptable homogeneity tolerance	Using the average 3D velocity field from CFD	Not multiphase, but gas-phase is present in system.

Research question

Relying exclusively on geometric compartments of the network-of-zones type has, however, an important trade-off between accuracy of results and speed of computation (Haringa, C., 2022). It is namely the case that the fewer geometric compartments are implemented in the bioreactor model, the more local details are lost and not considered. Therefore, information is lost and the accuracy of the end result is diminished. To counteract the reduced accuracy, more geometric compartments can be implemented, but the number of compartments is limited by the processing time needed to perform all calculations. Dynamic compartmenting does not have the same trade-off, since compartments are constructed based on hydrodynamic parameter(s). Because of the use of hydrodynamic parameters, it can be assumed that the dynamic compartments are homogeneous in nature and thus the model is inherently more accurate at less compartments. Tajsoleiman *et al.* have already shown a dynamic compartmenting method using Matlab without taking into account multiphase possibility and, by working only with the axial and radial velocities, leave out the third velocity field dimension.

This research will continue to improve on the concept of dynamic compartments, by adding the possibility to implement gas-phase variables and including the entire velocity field in determining how compartments should be formed.

In order to achieve the above-mentioned goal, the geometric compartment model provided by Dr. Haringa, C. is used and is henceforth called the base script. Because the base script is blunt in terms of coding, it will be functionalized in order to improve readability and modularity, while allowing the possibility for multi-phase implementation. Subsequently, the validity of the now functionalized script will be tested through mixing time calculations. It is hypothesized that the mixing time results will be identical between both scripts.

Additionally, the base script geometric compartment model will be expanded by adding dynamic compartmenting capabilities. Two approaches will be attempted in order to do the expansion. Firstly, the sorting method comparable to what Tajsoleiman *et al.* have proposed and secondly, a so-called allocation method, in which compartments are dynamically allocated in-CFD. Instead of first exporting the CFD data and subsequently perform dynamic compartmenting in Matlab, this research will do the compartmenting inside the CFD simulation. This disadvantage of exporting CFD data to external software is that not all CFD data is exported, only what the user perceives as useful. This research bypasses the above-mentioned disadvantage by actually doing the compartmenting in-CFD, so no simulation data is lost. In addition, in-CFD compartmenting has the benefit that the compartmenting script can run parallel to the simulation. Hereby

Both models will use a steady-state stirred-tank reactor of 7.7 m in height and 3.0 m in diameter. Syngas fermentation is usually performed in a bubble column, but for validation purposes a stirred-tank reactor is used for proof of concept, since stirred-tank reactor simulations are readily available. The gas-phase is enabled and will be considered, but the simulation will only be single-phase in this work.

The CFD software that is used is Ansys Fluent (Ansys Inc., Canonsburg, Pennsylvania, USA), especially the built-in C-compiler will be used for in-CFD compartmenting. The use of C language has the advantage that it provides for efficient management of memory in terms of allocation and releasing (Klemens, B., 2014). The drawback is a direct consequence of the age of the language: in order to perform current-age tasks, creative solutions and usages of C are needed. Furthermore, post-

processing, i.e. mixing time calculations, will be done in Julia programming language instead of Matlab. Julia combines the speed of C, while remaining as easy to use as Python. Furthermore, it is faster than MATLAB because of its use of an LLVM-based JIT compiler and has as multitude of options for performance improvements.

CFD software used is Ansys Fluent version ANSYS 2020 R1 which is run on the servers of Delft University of Technology, whereby a maximum of 24 CPU cores are used at any one time. The desktop on which the programming and postprocessing are performed has the following specs:
Intel Core i5-9600K @3.70 GHz (6 CPU's)
16 GB RAM
NVIDEA GeForce GTX 1650 SUPER 4 GB VRAM
Windows 10 Education 64-bit

Methods

In this section, the scripts that are used are described as well as the functionalization reasoning and validation. Furthermore, the compartmentalization methods and their respective validations are described.

The scripts

The averaging script – see Appendix 1.2

The averaging script is a UDF-type script which calculates the averages of CFD-derived parameters, namely the x, y and z direction liquid velocity, the x, y and z direction gas velocity, the turbulent kinetic energy, the dissipation of the turbulent kinetic energy and the gas fraction in the bioreactor. If the use of average data is desired by the user, the averaging script should be executed first in the CFD.

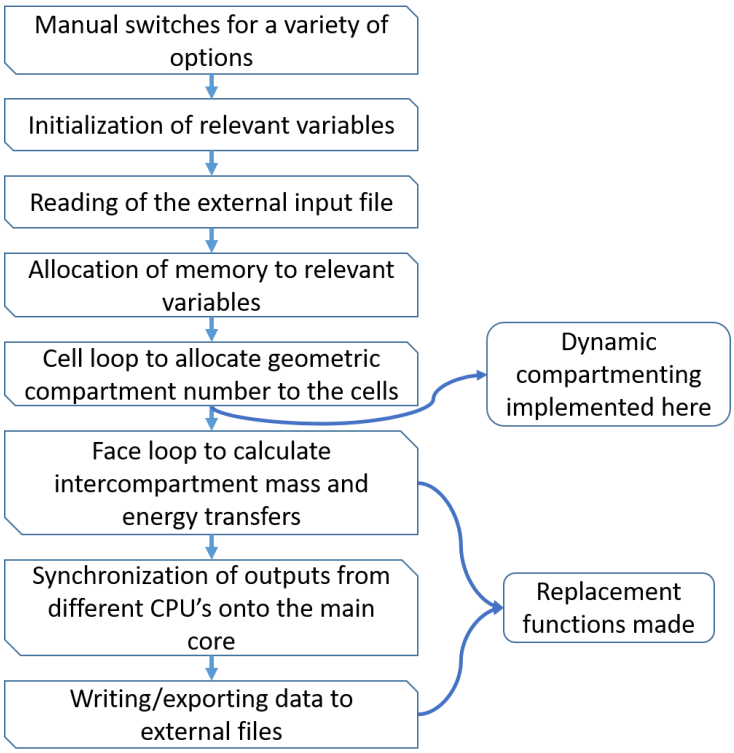


Figure 3. General overview of the base case script and the parts which were functionalized or where additions were made. *The base script, upon execution in the Ansys simulator, reads through manually implemented switches and input files before constructing geometric compartments between which convective flux and energy transfers are calculated. Then, data calculated by different CPU cores has to be synchronized on the host core in order to acquire a united data set. Finally, the obtained data is exported to external files for further usage. One of the focus points of this research was on improving the readability and modularity of the script by converting parts of the script into UDF-nested functions. The other part of this research was on adding a dynamic dimension to the geometric compartment model. See Appendix 1.3 for thorough explanation.*

The base script - see Appendix 1.3

This work relies on the base script for the pre-formation of geometric compartments (figure 3). The base script is provided by Dr. Haringa, C. and is of the UDF type of function. The base script derives the user-desired number of axial, radial and tangential compartments from an external inputs file, constructs geometric comparts and calculates the intercompartmental fluxes and transfers of turbulent kinetic energies. Furthermore, the base script contains code which exports the calculated data to external files.

The functionalized scripts – see Appendix 1.4

The functionalized script converts parts of the base script into nested functions, which basically makes those parts of code individual entities, which can be called when required with the desired arguments. In essence, the nested functions introduce modularity to the base script and thus improves the useability of the base script. The parts of the base script which are functionalized are the calculations of the convective flux and the turbulent kinetic energy and the data writing to external files.

The compartmentalization scripts – see Appendices 1.5

The compartmentalization script has as input the geometric division of the bioreactor, which is converted into a dynamic compartmenting. The script automatically calculates the homogeneity factor, which is an added complexity step in the model.

The mixing time calculation – see Appendix 1.6

The mixing time is calculated using a script provided by Dr. Haringa, C. and is written in the Julia language. This script determines the covariance in the mixing time of a tracer and provides the actual mixing time of the model.

Functionalization

Improving script readability

The Ansys compiler always runs a main function, which contains most C-language libraries already imported. Users are then allowed to program a user-defined function, or UDF, which can be executed at any desired moment during the CFD simulation. Strikingly, the UDF is comparable in its usage to regular C main functions. Subsequently, UDF's can have nested-functions, which are a regular occurrence in C (figure 4). The base script used as the baseline for this research falls under the category of a UDF, which is executed after the simulations have passed. However, since the main Ansys script does not import every C library, a header file containing the remaining user-needed libraries is needed in addition to the UDF. This research makes use of both the UDF and nested function parts of the function nesting order.

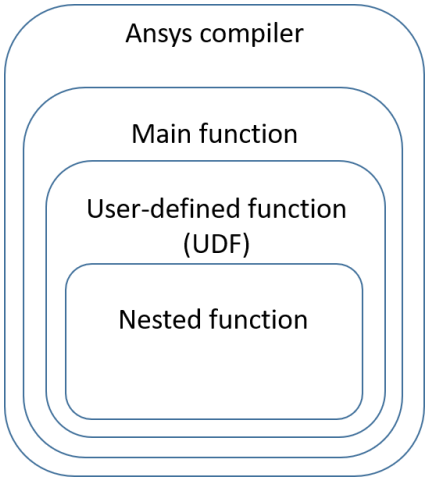


Figure 4. Function nesting in the Ansys compiler. *The Ansys compiler always runs a main function, in which a user-defined function can be nested and executed. Additionally, the UDF is able to accept a nested-function of itself.*

In order to improve the useability and readability of the base script, a modularity approach was considered. Hereby, parts of the script, which contained repeating snippets of code or are whose need is user-dependent, were converted into modular functions. The main effort was put into the pieces of code which contained the calculations for the convective flux and energy transfers between compartments: the fluxes. Users can switch the use of the flux functions on or off at the initial ("switch") part of the code (figure 3). Three functions were written, namely PhaseFluxIO();, PhaseFluxIntC(); and TurbulenceIntC(); (figure 5). The above-mentioned functions are successively called inside a cell face loop, whereby the PhaseFluxIO(); and PhaseFluxIntC(); functions are used to calculate the convective flux between compartments, while the TurbulenceIntC(); function is used for calculating the energy transfer. The bioreactor mesh contains three different types of grid cells, namely inlet cells, outlet cells and internal

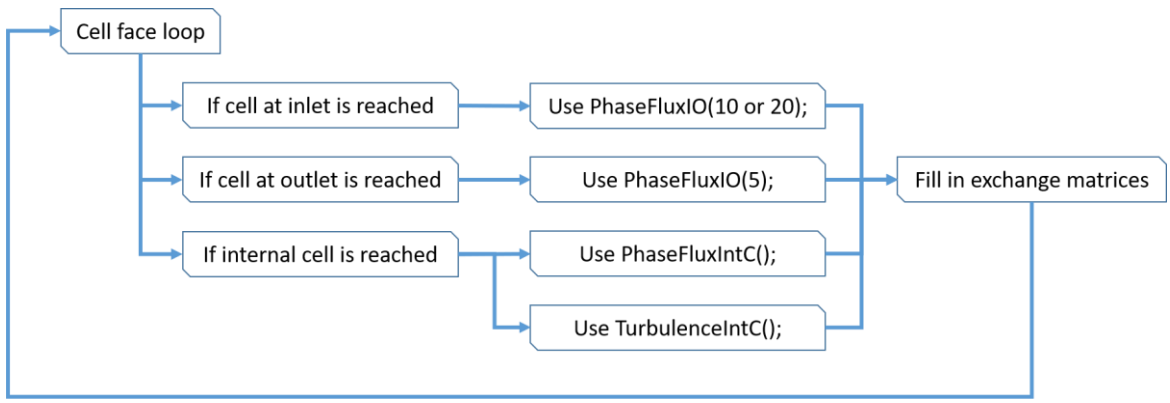


Figure 5. Overview of usage of nested functions inside the UDF*. See Appendix 1.4 for thorough explanations of each function.*

cells. The main difference between inlet and outlet cells on one side and internal cells on the other, is that inlet and outlet cells do not have neighboring cells. Therefore, no intercompartmental fluxes can be calculated, but instead the fluxes from the outside of the bioreactor into the reactor (in the case of inlet cells) and the fluxes out of the bioreactor (in the case of outlet cells) are determined. This last fact also points to the difference between inlet and outlet cells, namely the direction of the flux. Moreover, in Ansys inlet cells are stored in subthreads identified with numbers 10 and 20, whereas outlet cells are stored in subthreads with 5 as identification number. Contrary to the convective flux functions, the turbulent kinetic energy function is only applied to internal cells, since energy is assumed to be conserved inside the bioreactor.

Furthermore, every function allows for the implementation of both single-phase and multi-phase as well as allows the usage of time-averaged data. Implementation of the multi-phase and averaged data are outside of the scope of this project, however. An overview of the functions is provided in figure 6, while the detailed approach is available in Appendix 1 and the calculations in Appendix 2.

The functions all output data into an exchange matrix, which contains all fluxes between neighboring compartments (figure 7). The exchange matrices are constructed such that the rows and columns correspond to the compartments and that the diagonals contain the total flux exiting the corresponding compartment. Moreover, the rows contain the flux from the 'row compartment' into the 'column compartment', while the columns show the flux entering the 'column compartment' originating from the 'row compartment'. Compartments which do not border each other have zero flux between them, which is evident from the zero value in the relevant exchange matrix element. Additionally, the same applies for the energy transfer between compartments.

The exchange matrices are subsequently exported to external files for post-processing purposes. Since the CFD produces a multitude of data about the geometric compartments, it was decided to functionalize the writing of data, with the goal of making the export of data modular and easy to use.

Mixing time

In order to test the validity of a CFD model, the mixing time that is needed for a tracer to reach 95 % concentration mixing in the bioreactor has to be calculated. It has been shown by the research group of Hartmann, H. that the coefficient of mixing (the covariance of the mixing time) is linearly related to the mixing concentration as follows (Hartmann, H. *et al*. 2006).:

$$C_{mix} = 0.17\left(\frac{100 - C_{\%}}{30}\right)$$

Utilizing the above equation and assuming that a mixing concentration of 95 % represent a sufficiently mixed bioreactor tank, the covariance of mixing should be 0.0283 or smaller. The determination of the mixing time covariance is done using a post-processing script in Julia language (julialang.org), run in Visual Studio Code (Microsoft, Redmond, Washington, USA).

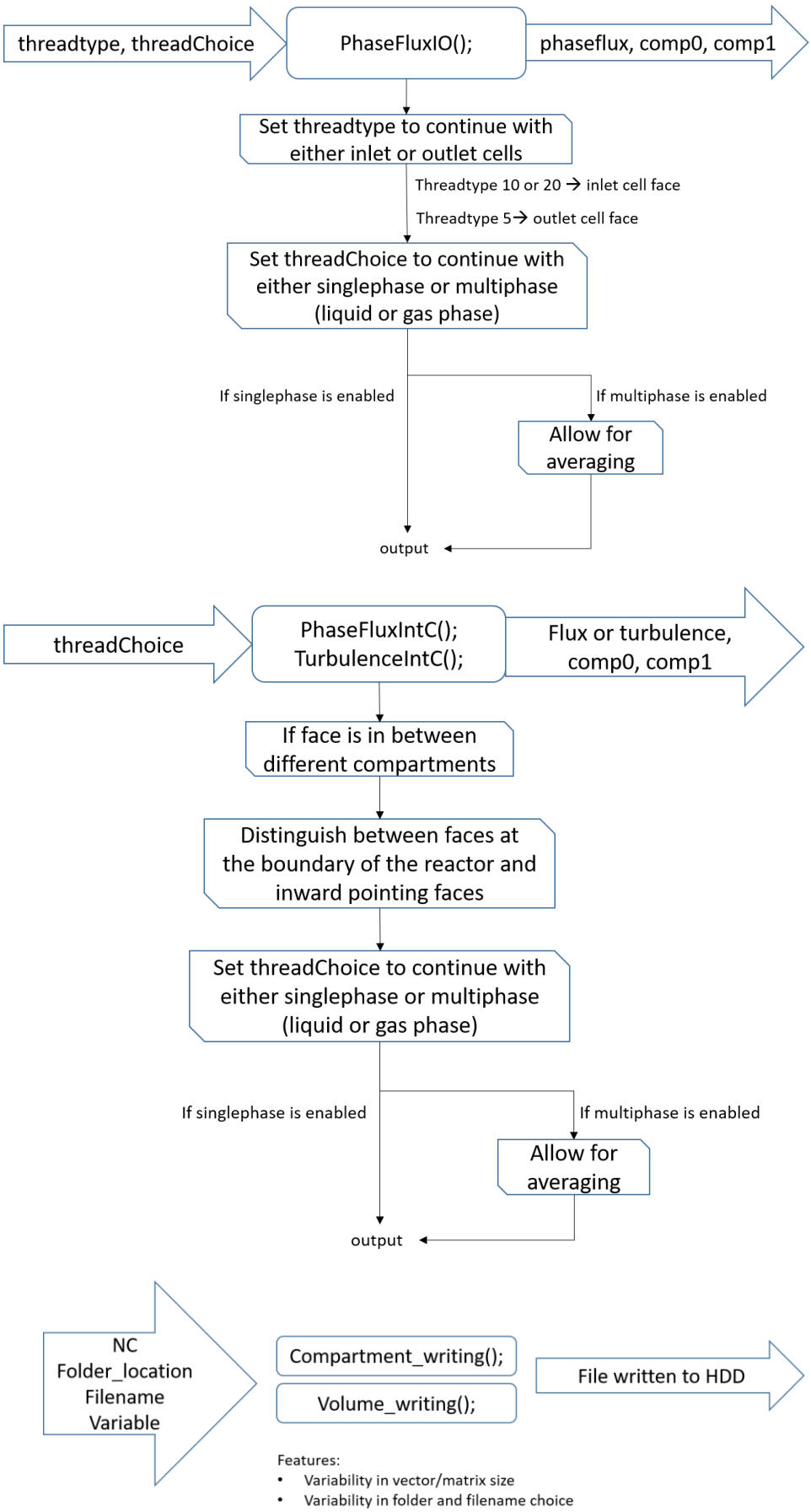
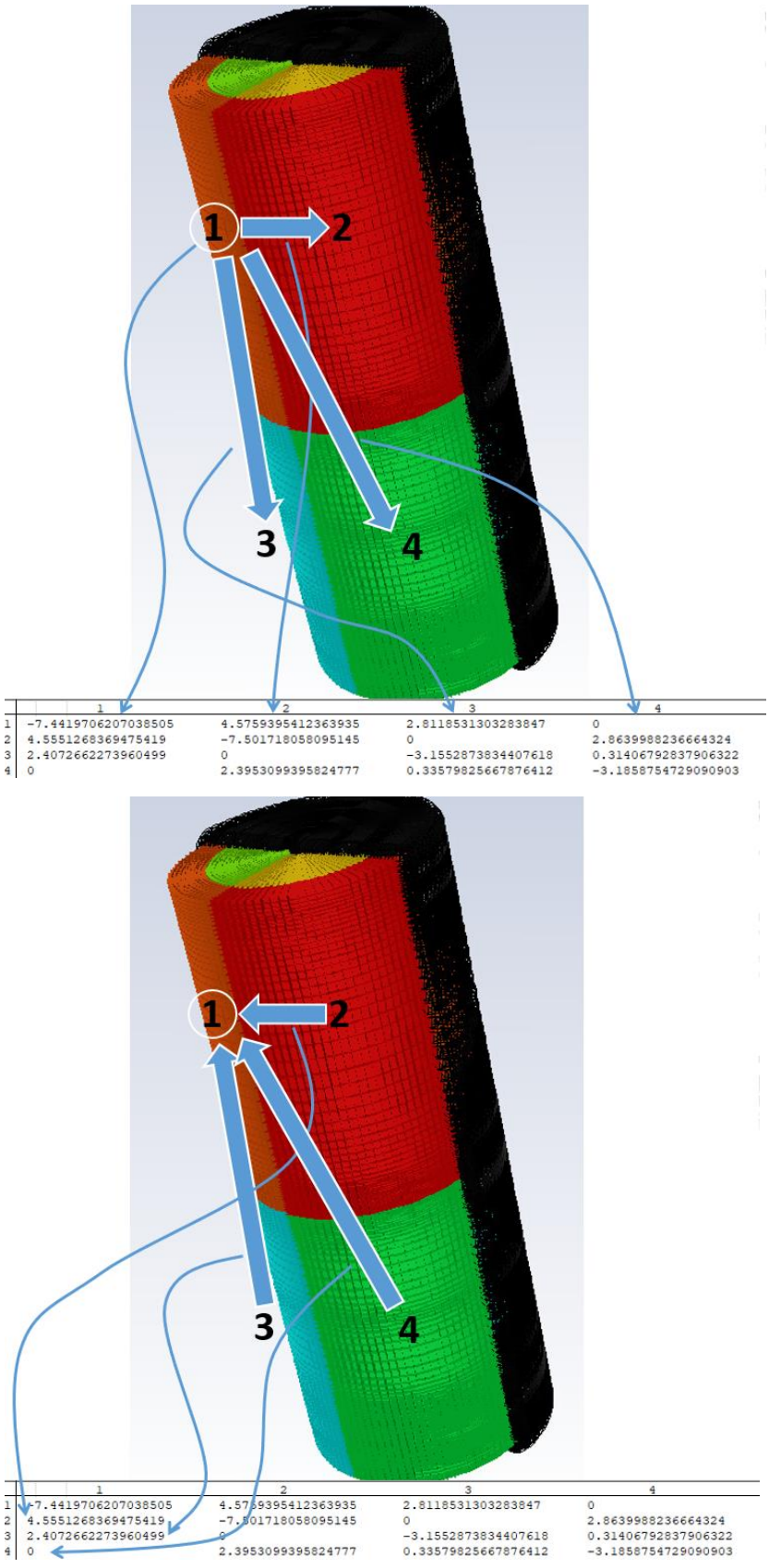

threadtype, threadChoice
PhaseFluxIO();
phaseflux, comp0, comp1
Set threadtype to continue with either inlet or outlet cells
Threadtype 10 or 20 → inlet cell face
Threadtype 5→ outlet cell face
Set threadChoice to continue with either singlephase or multiphase (liquid or gas phase)
If singlephase is enabled
If multiphase is enabled
Allow for averaging
output
threadChoice
PhaseFluxIntC();
TurbulenceIntC();
Flux or turbulence, comp0, comp1
If face is in between different compartments
Distinguish between faces at the boundary of the reactor and inward pointing faces
Set threadChoice to continue with either singlephase or multiphase (liquid or gas phase)
If singlephase is enabled
If multiphase is enabled
Allow for averaging
output
NC
Folder_location
Filename
Variable
Compartment_writing();
Volume_writing();
File written to HDD
Features:
• Variability in vector/matrix size
• Variability in folder and filename choice

Figure 6. Modular functions for the implementation of flux and turbulence calculations and saving thereof. *For the calculation of fluxes and turbulent kinetic energy between compartments, modular functions were written. The functions allow the user to choose between singlephase or multiphase calculations, whereby in multiphase differentiating between the liquid and gas phases is also a possibility. Furthermore, in the case of a multiphase system, it is possible to enable the use of averaged data in order to take into account time-dependent differences in liquid and gas flows. The averaged data is produced with an external function, which has to be run in the simulation before the functions in this figure can be used. All functions described below are implemented within a cell face loop, which loops over all cell faces in the bioreactor and runs the functions at each face. (**A**) The top function is used to calculate the fluxes at the inlet and outlet cells, which are specifically defined in the Ansys simulation. The function has two inputs, namely the 'threadtype' and 'threadChoice'. The 'threadtype' input, which can have the value 5, 10 or 20, allows for the differentiation between inlet cells, if values 10 or 20 are used, and outlet cells, if the value 5 is used. Subsequently, the 'threadChoice' can be set as 1, 2 or 3, which respectively steer the function towards implementing singlephase, liquid-phase or gas-phase-specific calculations. In case the multiphase is desired, the use of averaged data is possible, as described above. Every route through the function leads to the 'output', which is a route-specific calculation of the flux. Additionally, the function enables pointers to 'comp0' and 'comp1', which are the compartments between which the flux is calculated. (**B**) The bottom function schematic can be used for both the fluxes as well as the turbulent kinetic energy between compartments. With these functions only the 'threadChoice' has to be put in by the user, in order to choose the desired phase. Then, the functions determine whether the cell face, which the cell face loop has reached, is located on the border between compartment, since only intercompartmental fluxes and energy transfers need to be considered. The next if-statement leads to distinguishing between cell faces between cells and the exterior of the bioreactor and cell faces which are pointed inwards of the bioreactor. The subsequent steps are the same as the with the above-mentioned function, and lead to an 'output', which is again a route-specific calculation of the flux or the energy transfer. (**C**) The CFD produces a lot of data, from data pertaining to the grid cells to cell face-derived data. Furthermore, most data is available in the form of a vector, but matrices are also possible, i.e. the fluxes between compartments are available in the form of 2D array. In order to increase user-friendliness and accommodate to the differences in data structures, two functions were made, whereby "Compartment_writing();" writes 2D arrays to an external file, while "Volume_writing();" writes vectors to external files.*



	1	2	3	4
1	-7.4419706207038505	4.5759395412363935	2.8118531303283847	0
2	4.5551268369475419	-7.501718058095145	0	2.8639988236664324
3	2.4072662273960499	0	-3.1552873834407618	0.31406792837906322
4	0	2.3953099395824777	0.33579825667876412	-3.1858754729090903

	1	2	3	4
1	-7.4419706207038505	4.5759395412363935	2.8118531303283847	0
2	4.5551268369475419	-7.501718058095145	0	2.8639988236664324
3	2.4072662273960499	0	-3.1552873834407618	0.31406792837906322
4	0	2.3953099395824777	0.33579825667876412	-3.1858754729090903

Figure 7. Schematic visualization of the exchange matrix setup. *The exchange matrix has compartment numbers as rows and columns, whereby the diagonal gives the total flux exiting a compartment. The rows are the exiting fluxes divided into their respective destinations, whereas the columns represent the incoming fluxes split into their respective compartments of origin.*

Compartmentalization

The geometric compartments of the base script already reduce the computational power needed to perform bioreactor-wide calculations, but the model can be made more advanced by allowing for dynamic formation of compartments, based upon the hydrodynamic parameters extracted from the CFD simulation. To this end, this thesis will show two different attempts to construct dynamic compartments: a sorting-based method and an allocation-based method.

The sorting method

The first method is based on previous work done on the subject by Tajsoleiman *et al.* and involves the extraction of CFD data on hydrodynamic parameters, i.e. the liquid velocity (Tajsoleiman, T. *et al*., 2019). The data is subsequently sorted and split into even-sized matrices, each of which then corresponds to a compartment. The hydrodynamic parameters are extracted from the CFD using the Data_extraction(); function (see Appendix 1.5.2.1). Subsequently, two different sorting methods were used and compared, namely bubble sort (using function BubbleSort();, see Appendix 1.5.1.3) and heap sort (using function HeapSort();, see Appendix 1.5.1.2). The differences between the two sorting algorithms lie in the stability of the result and the best and worst cases, which are respectively the performance of a sorting algorithm when the input data is already sorted and when the input data requires the most resources to tackle. It is known that bubble sorting has a best-case big O notation of O(n)(Appiah, O. & Martey, E.M. 2015), while heap sorting performs better with O($n\ log\ n$) (Marcellino, M., *et al.* 2021). The same result is seen when comparing the performance of the two algorithms in a worst-case scenario: bubble sorting has a big O notation of O(n^2), while heap sorting has O($n\ log\ n$) (table 2).

Table 2. Comparison between the inherent features of sorting algorithms. *The heap sorting and bubble sorting algorithms both have advantages and disadvantages. The main advantage of the heap sorting is that it performs equally well in both best-case and worst-case scenarios, while being faster than bubble sorting in both cases. The main disadvantage is that heap sorting is an unstable algorithm, whereas bubble sorting is not.*

	Best-case	**Worst-case**	**Stability**
Heap sort	O($n\ log\ n$)	O($n\ log\ n$)	Unstable
Bubble sort	O(n)	O(n^2)	stable

The main drawback of the heap sort is that it is not considered a stable algorithm: equal elements are not preserved in their relative order. Therefore, a trade-off between stability and speed is observed, with heap sort and bubble sort each at opposing sides.

The practical usage of both algorithms was tested by way of generating an Nx8 matrix filled with random values, with N iteratively incremented from 1 to 200,000. Since the data that would be extracted from the CFD contains 235,104 rows and 8 columns, a matrix of 200,000 x 8 is in the same order of magnitude and is therefore representative.

After sorting the data, the data is divided into equally-sized bits, based on the size definition provided by the homogeneity factor. Each bit thus represents a compartment, which contains grid cells that have approximately the same value for the chosen hydrodynamic parameter. The function Split(); has been written to this end (see Appendix 1.5.1.4).

The allocation method

An overview

The second method for forming dynamic compartments is by iteratively allocating an identical compartmentalization number to two neighboring geometric compartments when their hydrodynamic parameters are comparable in magnitude. To this end a homogeneity factor is used as comparison medium and a vector was initialized as a zeros vector, which is used to track which geometric compartments are already merged. The following steps are then successively followed: as starting point the geometric compartments are needed, from each of which the average 3D velocity field is derived (figure 8). Then a cell face loop is executed which firstly determines whether the border between two geometric compartments is reached. If that is the case, then the largest of the three average velocities is used to compare two neighboring geometric compartments: when the velocity difference and direction of the neighbors falls within the range set by the homogeneity factor, then the neighbors should be merged. Subsequently, the grid cells of which the two compartments are comprised are allocated an identification number, which is stored in memory and used for visualization of the merged compartments. If, on the contrary, the velocity difference and direction of the neighbors falls outside of the bounds set by the homogeneity factor, the grid cells of both compartments are each given a different identification number, since no merging should take place. Lastly, the previously mentioned merge-tracking vector is edited, in order to prevent the reconsideration of already merged geometric compartments. See Appendix 1.5.2 for the detailed coding explanation.

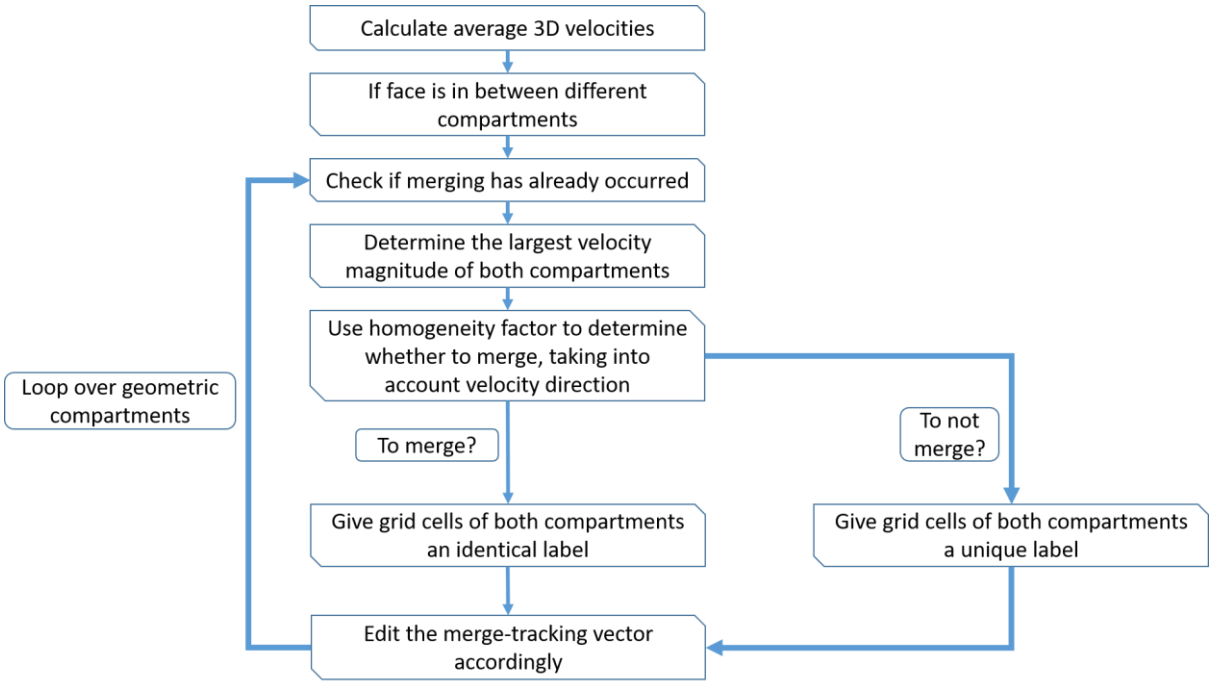


Figure 8. Overview of the dynamic compartmenting method. *Having calculated the average 3D velocities of the geometric compartments, a cell face loop is initiated in which the grid cell faces are tested on being on the border between two geometric compartments. If that is the case, the merge-tracking vector, which is an array of all zeros, is checked if merging has already occurred or not. Then the largest of the three average velocities is chosen for further calculations, which entail using a dynamically-determined homogeneity factor to decide whether two neighbouring geometric compartments should merge. Consequently, both geometric compartments are allocated the same label if they are merged, while each gets a unique label in case they are not merged. Finally, the merge-tracking vector is edited, in order to prevent the compiler from considering already merged geometric compartments again.*

Design choices

Previous research and literature show that the preference is to use the liquid velocity in the bioreactor as basis for the compartmentalization method. Instead of using a velocity parameter at a reference plane only (Tajsoleiman, T. *et al.*, 2019), the average velocity across the entire geometric compartment is used. Hereby, the geometric compartment is utilized as an independent entity (independent from the grid cells of which the compartment comprises). However, very local velocity data is hereby lost. The above is certainly a fact when a fewer number of geometric compartments are initially used.

Furthermore, in order to account for vortices in the bioreactor, the direction of the liquid flow is taken into account. Hereby, the liquid flow direction is either positive or negative and is one of the main considerations when comparing the velocity parameter with the homogeneity tolerance factor. Moreover, when the average velocity field of the geometric compartments is determined, the largest of the three velocities, x, y or z-direction velocities, are considered for further use. The reasoning behind the use of only one of the velocities is simplicity: the most significant velocity direction in the stirred-tank reactor because of the impellers is in the axial direction. Therefore, the assumption can be made that the x and z directions (y direction is considered axial in Ansys) have less prominent contributions to the overall velocity field.

Results and Discussion

Functionalization

Geometric compartment data

The usage of nested functions inside of a UDF instead of solely the UDF, adds modularity, improves the code readability and increases ease of use. In order to verify the usability of the functionalized script, both the base script and the functionalized scripts were executed in the Ansys simulation and the resulting volume-vectors and exchange matrices were subsequently imported into the post-processing program to determine the mixing time of a tracer. Firstly, the functionalized script itself outputs user-chosen data files (using the Compartment_writing(); and Volume_writing(); functions), for example about the geometric compartment volumes (table 3). In this situation, a simplified model is used for comparison purposes, by editing the ("InputCompartmentScript.txt") such that a total of four geometric compartments are formed. The compartment volumes are identical between the two scripts, even at the 15th decimal (not shown). Even though the bioreactor is divided into equally sized compartments, compartments 2 and 4 are smaller in volume, which is due to the presence of the impellers in said compartments.

Table 3. Compartment volumes comparing the functionalized script with the base case. *In order to be able to visually compare the functionalized script with the base case a reactor distribution comprising of 1x1x4 (tangential x radial x axial) was implemented and used. (**A**) The left table shows the compartment volumes (m^3) of the four compartments resulting from the base case situation. Two of the four compartments have a smaller volume, which is due to the presence of impellers in those two compartments. (**B**) The right table shows the volumes of the same four compartments, but resulting from the functionalized script instead. Because the initial formation of the geometric compartments did not illicit the need for a function to make the process more modular, no function was written and therefore the compartment volumes are identical between the two scripts.*

	Compartment volume (m^3)
Compartment 1	6.91811
Compartment 2	6.51382
Compartment 3	6.94658
Compartment 4	6.54153

Compartment volume (m^3)
6.91811
6.51382
6.94658
6.54153

In addition to the compartment volumes, the convective flux and energy flows between neighboring compartments are also exported to readable data files (table 4). Hereby, the convective fluxes are denoted in SI units of m^3/s and the turbulent kinetic energy transfer in m^2/s^2 units. Again, a simplified model is simulated for comparison purposes, whereby it is visible that the results between both the scripts are identical. The similarity in result is true until the 14th decimal, because numerical differences do occur from the 15th decimal onwards (not shown). A reason for the differences could be that the C compilers compile strictly from top to bottom. Even though the functions are prototyped at the top of the script, the compiler in unaware of the actual contents of the functions until their definitions are reached, which happens at the bottom of the script. Therefore, it is hypothesized that the numerical errors are due to the compiler revisiting snippets of code, instead of moving through the code only once. Another reason could be that the functions use pointers to the memory address of the data, instead of the data itself. In essence, the functions make indirect use of data, while the base script makes direct use of the variables. The indirectness resulting from pointer use could lead to small numerical errors in the end result. The numerical errors do not, however, impact the mixing time

calculations as can be seen in the following paragraph. The analogous compartment volumes and intercompartmental mass and energy flows between the base script and the functionalized script is tantamount to the functionalized script working as intended.

***Table 4. Intercompartmental mass flows and turbulence flows between compartments of both the base case as well as the functionalized script**. (**A**) The top two exchange matrices represent the mass flows between the compartments in m^3/s, whereas (**B**) the bottom two exchange matrices visualize the turbulent kinetic energy transfer (m^2/s^2) between compartments. On the diagonal you find the total flux (m^3/s) exiting the compartment, while the row elements show the total flux to that destination compartment (i.e. the first row second column is the flux from compartment 1 (the first row) to compartment 2 (hence the second column). Moreover, the columns represent fluxes into the compartment, i.e. matrix element [1,0] is the flux entering compartment 1 from the direction of compartment 2. Within each set of exchange matrices, the **top (A1 & B1)** matrix is the result of the base case script, whereas the **bottom (A2 & B2)** matrix is from the functionalized script. In comparing the two, small numerical differences are visible, most of which occur at or around the 15^{th} decimal number (not shown). Although the data type, arguments and parameters of the functions are initialized in the form of a prototype at the header of the script, the actual content of the functions is unknown to the compiler until the function definition is reached. Furthermore, the functions are called iteratively within a face loop, which adds to the complexity of compiling the script. Therefore, compared to reading the script from top to bottom without using functions, the extra steps involving the use of functions could be inductive of numerical errors in the calculations.*

A1

flux (m^3/s)	Compartment 1	Compartment 2	Compartment 3	Compartment 4
Compartment 1	-1.62763	1.62763	0	0
Compartment 2	1.627633	-2.65312	1.02549	0
Compartment 3	0	1.02549	-1.79161	0.76612
Compartment 4	0	0	0.76612	-0.76612

A2

flux (m^3/s)	Compartment 1	Compartment 2	Compartment 3	Compartment 4
Compartment 1	-1.62763	1.62763	0	0
Compartment 2	1.62763	-2.65312	1.02549	0
Compartment 3	0	1.02549	-1.79161	0.76612
Compartment 4	0	0	0.76612	-0.76612

B1

Turbulent kinetic energy (m^2/s^2)	Compartment 1	Compartment 2	Compartment 3	Compartment 4
Compartment 1	-2.08024	2.08024	0	0
Compartment 2	2.08024	-6.63546	4.55521	0
Compartment 3	0	4.55521	-5.76027	1.20505
Compartment 4	0	0	1.20505	-1.20505

B2

Turbulent kinetic energy (m^2/s^2)	Compartment 1	Compartment 2	Compartment 3	Compartment 4
Compartment 1	-2.08024	2.08024	0	0
Compartment 2	2.08024	-6.63546	4.55521	0
Compartment 3	0	4.55521	-5.76027	1.20505
Compartment 4	0	0	1.20505	-1.20505

Mixing time

The mixing time post-processing calculations are performed in Visual Studio Code using the Julia-based script ("compartment_eulerian_mixer.jl", see Appendix 1.6). The post-processing comprises reading the exchange matrices, of both the mass as well as energy transfers, and a tracer introduced into the system a user-specified location. Subsequently, the coefficient of mixing, the covariance, is calculated and plotted (figure 9). Hereby, the compartmental division of the bioreactor is given in the following format: A[*number of axial divisions*]R[*number of radial divisions*]T[*number of tangential divisions*]. It can be seen that the mixing coefficients of the same system are identical between the base script and the functionalized script, with the A18R2T1 model resulting in a mixing time of 54.8 s and the A26R6T1 model having a mixing time of 60.2 s. Both mixing times are an underestimation of the experimentally calculated mixing time of 72 s (Delafosse, *et al.* 2014). Because CFD-based compartment models have a major tradeoff between accuracy and computation power requirement, the A18R2T1 model has an inherently lower accuracy compared with the A26R6T1 model, which has more compartments. The computational power requirement does not play a role in either model, however, since the total number of calculations is a drastic reduction compared with the CFD model without compartments.The goal of the functionalization was to improve on the usability of the base script, which was achieved as proven by the mixing time validation. If the overall picture is concerned, the two logical aspects of the base script were functionalized, namely the convective flux and energy transfer calculations and the data writing (figure 3). Another minor part of the base script that could be made more modular would be the manual switches, especially when transferred into a graphical-user interface (GUI). In addition to the switches, data from the ("InputCompartmentScript.txt") file could also benefit from being moved a GUI: a sped-up input of different geometric compartment systems would be the result of such endeavor.

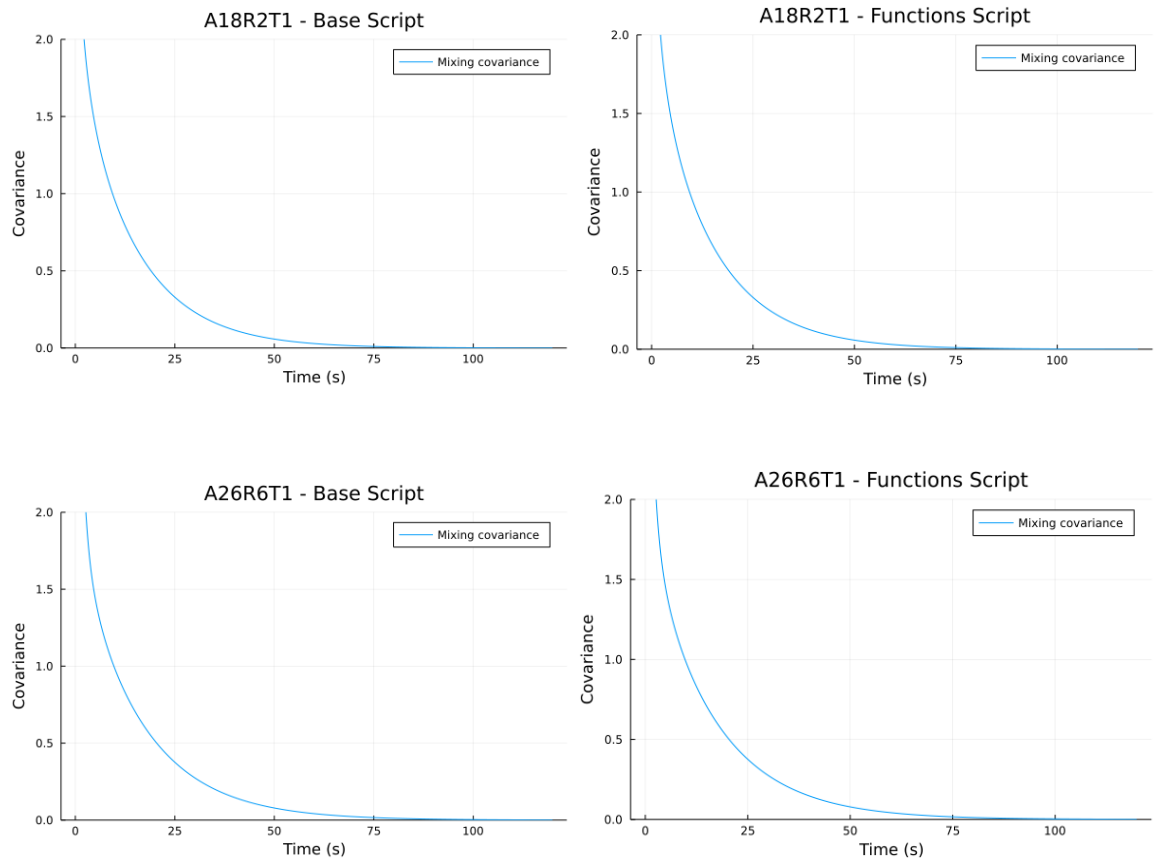


Figure 9. Mixing time covariance comparisons between two base case systems and two functionalized script functions. *Calculations of the mixing time are performed outside of the Ansys simulation environment, whereby the covariance of the mixing time is used as an indication of optimal mixing. Hereby a covariance of 0.0283 is indicative 95 % concentration mixing, and is considered sufficiently mixed (Hartmann, H. et al*. 2006*). The two systems used as comparisons are the A18R2T1 and A26R6T1 systems, which respectively consist of 18 axial, 2 radial and 1 tangential compartment, totalling 36 compartments, and 26 axial, 6 radial and 1 tangential compartment, giving a total of 156 compartments. It can be observed that the functionalized script produces the same results as the base case script, since the covariance has the same dynamics in both plots. Furthermore, the A18R2T1 scripts both have the same mixing time of 54.8 s and the A26R6T1 scripts also both share the same mixing time, but this time of 60.2 s.*

Compartmentalization

Sorting method

The sorting strategy of compartmentalization initially consisted of comparing two different sorting algorithms, the bubble sort and the heap sort. Because of the better best-case and worst-case known performances of the heap sort, it was expected that the heap sort would certainly show better results (figure 10). Indeed, the heap sort performed better in terms of processing time required to sort a 2D array with eight columns and up to 200,000 rows. However, the differences are minimal, and the bubble sort even catches up at the higher end of the plot. The lower range of the plot shows noise in the data, which is due to the limited number of times which the algorithms have been run. Apparently, it does not matter which algorithm is chosen with the data size that is used: both sorting algorithms remain the principal bottleneck of the compartmentalization strategy, since compartment model processing should be in the order of seconds (Tajsoleiman, T. *et al.* 2019).

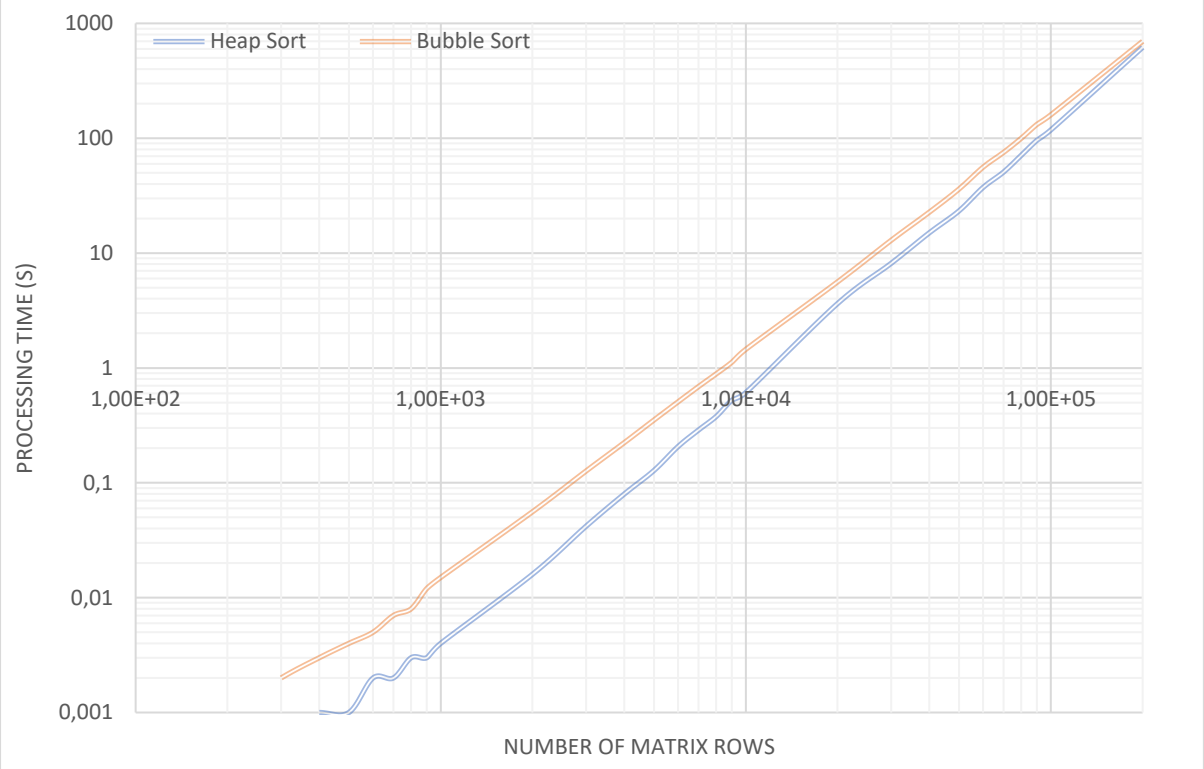


Figure 10. Processing time comparison between the bubble sort and the heap sorting methods. *Since data sorting is the main bottle neck in the computation requirement, two sorting methods were compared for processing time. In all cases the number of columns of the matrices is constant at 8, whereas the number of rows is gradually increased, from 1 to 200,000. Because the total number of grid cells in the bioreactor mesh is 235,104, a row count of 200,000 is in the same order of magnitude. It can be observed that the heap sorting method is always faster than the bubble sorting method, even though the differences are not major. The noise at the lower range of matrix rows is due to the runs being performed a limited number of times.*

In terms of sorting methods, both bubble sort and heap sort perform relatively the same, while both remaining the main bottleneck in processing time. In practical sense, the bubble sort is considerably easier to implement, but the simplicity thereof is also what makes this sorting method infamous. In future projects the bubble sort should be avoided, as advised by prof. Astrachan (Astrachan, O. 2003).

The heap sort, on the other hand, is useful in that it performs repeatedly the same, even though the inherent sorting instability is unavoidable. Even though one has to be mindful of the complexity of

coding and executing other sorting methods, more modern heap sort methods are available nowadays which could be attempted in future projects (Weiss, A. & Diekert, V., 2015).

During this research the sorting method of compartmentalization did not result in actual dynamic compartments, because of the inability to accomplish the classification step of the process (Tajsoleiman, T. *et al.* 2018). The classification step involved separating the sorted data sets into equally sized chunks, each of which would contain the data of a respective compartment. In practical terms, this meant working with 3D arrays, with the third dimension being the compartment number. The function Split(); was designed to do that, see appendix 1.5.1.4. C language is, however, not tailored towards extensive usage of multidimensional arrays, since all data is stored in a linear fashion, without differentiating between parts. Furthermore, because of time constraints, not enough effort could have been invested into finding a different solution.

Allocation method

The allocation method involved looping over all geometric compartments and comparing their respective average velocity magnitude and direction with a tolerance range (figure 8). The method is therefore based on the use of the velocity field in three dimensions (figure 11). The velocity fields show that the axial direction has the most prominent velocity differences in terms of both magnitude and direction, which corresponds with what is expected in a stirred-tank reactor. Furthermore, multiple vortices are observed in the regions around the impellers and towards the outer walls.

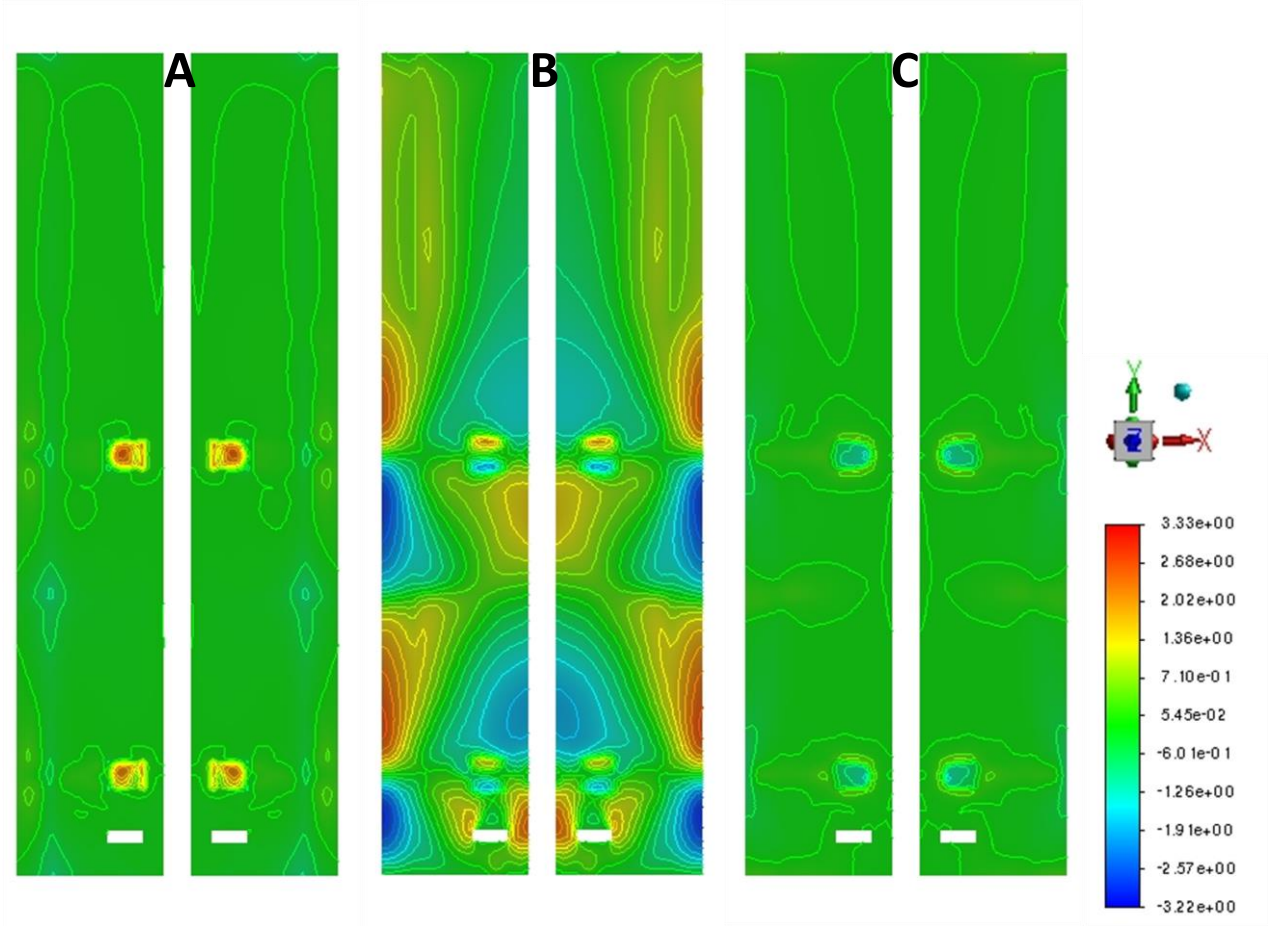


Figure 11. Central plane surface of the bioreactor showing the velocity fields in three dimensions. *The velocity fields shown here are respectively of the X-direction (**A**), Y-direction (**B**) and Z-direction (**C**) and the result of a very fine mesh overlay, comprising 235,104 grid cells, in Ansys. Furthermore, the compass which shows the directions as interpreted by the Ansys solver is given on the right-hand side, in addition to the legend, which displays the liquid velocity in units of m/s. Underneath each figure the colour explanations are given, in which a negative value indicates the opposite direction compared to what the compass indicates. It can be seen that the most prominent regions of X and Z-direction velocities are at the impeller locations, whereas in the axial direction (the Y-direction) multiple vortices are observed. The axial direction shows the most activity in terms of velocity magnitudal and directional changes, which is in line with what is generally observed in a stirred-tank reactor. See Appendix 3.1 for enlarged version.*

The geometric compartments are constructed first in ample numbers, in order to have enough compartments to merge (figure 12a). Since the RGB coloring range is limited in its usage for visualizing a great number of compartments (in this system 5000 geometric compartments) and being able to distinguish between them, the colors can not be used as a guide. Furthermore, the axial direction is chosen to have the finest distribution, since the largest variety of liquid flow velocities are observed in the Y-direction (figure 11b).

After implementing the compartmentalization script (see Appendix 1.5.2), the geometric compartments are merged according to the compartmentalization script scheme, resulting in a more dynamic distribution of compartments (see figure 12b). Firstly, it is apparent that the regions around the impellers are more densily packed with compartments, compared to the upper regions of the bioreactor. The bottom corners are also densily packed, since multiple smaller Y-direction vortices are present in those regions. Furthermore, the green compartment shows the upper part of an axial-direction vortex. Moreover, the bigger dynamic compartments are indications of shallow differences in Y-direction velocities in those regions, which corresponds to what is observed in the velocity fields. In three dimensions the compartments implicate that different liquid flow velocities and directions (i.e. vortices) exists tangentially (figure 13). It can be seen that compartmenting happens in the radial direction as well, which means that the impellers not only affect the flow axially, but also a radial flow gradient is present. The flow gradients do have an effect on the concentration gradient of the substrates, which would hamper the efficiency of the syngas fermentation.

After merging geometric compartment which should be merged according to the homogeneity factor, the initial total of 5000 geometric compartments of the A1250R4T1 system are reduced to 655 dynamic compartments. The reduction by a factor of 7.6 consequently means that instead of calculating mass and energy transfers in a 5000 x 5000 matrix (which are 25e6 calculations for the mass transfer and another 25e6 for the energy transfer), now only a 655 x 655 matrix is considered: 4.29e5 calculations for each transfer. This indicates that indeed a great improvement has been achieved in terms of reduction of the work load on the CPU/GPU system. Increasing the initial number of geometric compartments results in an increase in the subsequent number of dynamic compartments, albeit not in a directly proportional fashion (table 5): the more geometric compartments initially used, the bigger the impact by the dynamic compartmenting.

Table 5. The number of dynamic compartments resulting from the corresponding number of initial geometric compartments. *No directly proportional increase is observed between the two compartment models, which indicates that the allocation method is capable of drastic reduction of number of subsequent calculations needed.*

Geometric compartments	Dynamic compartments	Reduction factor
5,000	655	7.63
10,000	830	12.05
50,000	2566	19.49

It can, additionally, be seen that indeed the most detailed regions are around the impellers and the regions along the outer walls where vortices are observed in the Y-direction velocity field (figure 14). The compartments are formed symmetrically with the impeller shaft being the mirroring plane. Moreover, the dynamic determination of the homogeneity factor has a built-in possibility for the user to alter how detailed the compartments are: the tighter the homogeneity factor range, the more compartments are formed. Hereby, an increased number of compartments results in a closer representation of the geometric compartment result and therefore reduces the noise between the

mixing time results of both. There is a caveat, however, in a tradeoff between accuracy of results and speed of calculations: the more compartments, the more accurate the results, but this is to the detriment of the computation speed.

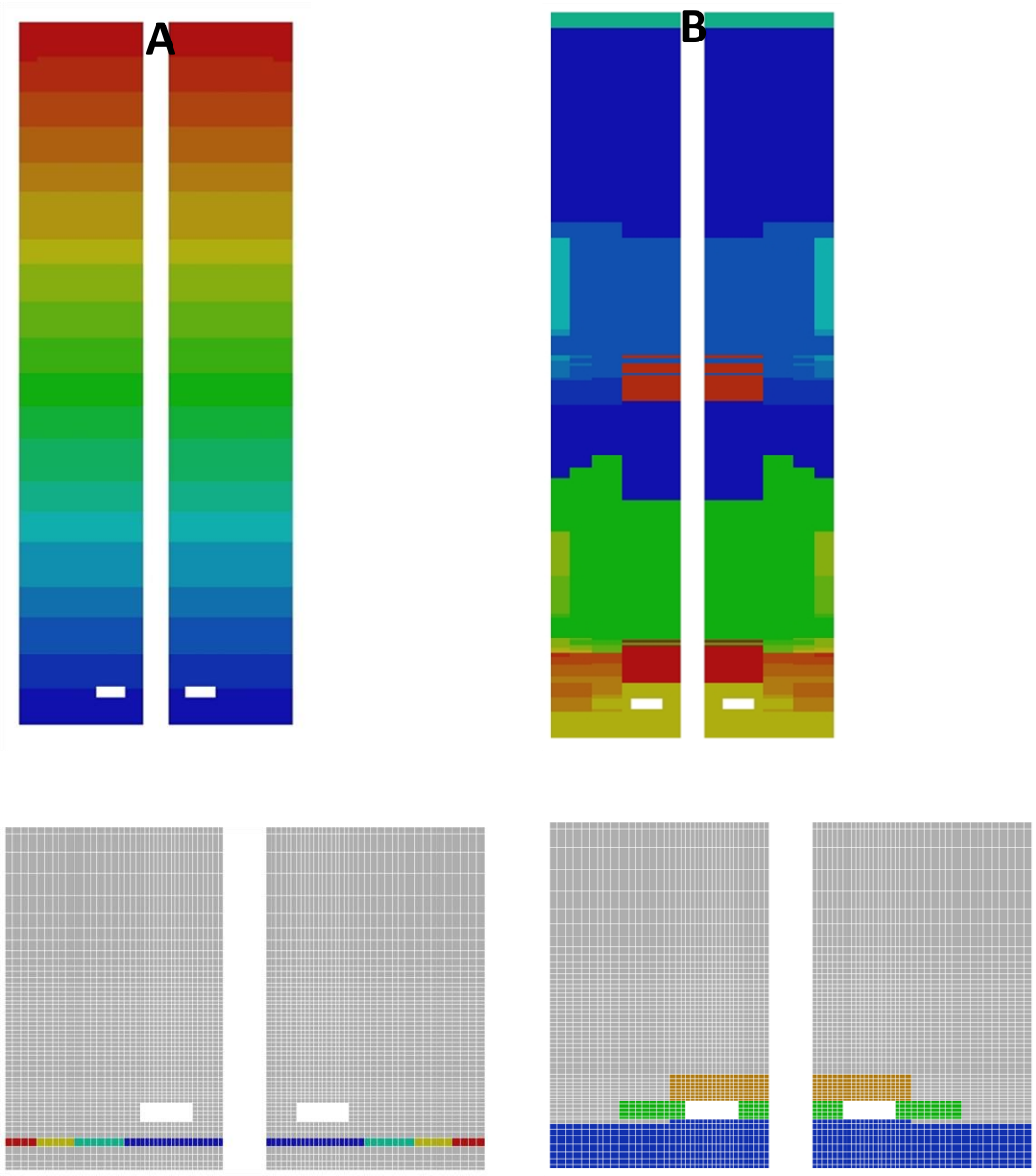


Figure 7. Central plane surface of the bioreactor showing the division into geometric compartments (A) and dynamic compartments (B). *(**A**) The geometric distribution is of the A1250R4T1 system, totalling 5000 geometric compartments. The bottom left pane shows the bottom part zoom in, with the mesh grid overlayed and four geometric compartments as illustration. (**B**) After implementing the compartmentalization script, the geometric compartments are merged according to the compartmentalization script scheme, resulting in a more dynamic distribution of compartments. The bottom right pane shows the zoomed in bottom part of the bioreactor, showing that indeed the four geometric compartments of the top figure are merged, along with the surrounding compartments. See Appendix 3.2 for enlarged version of the mesh grid.*

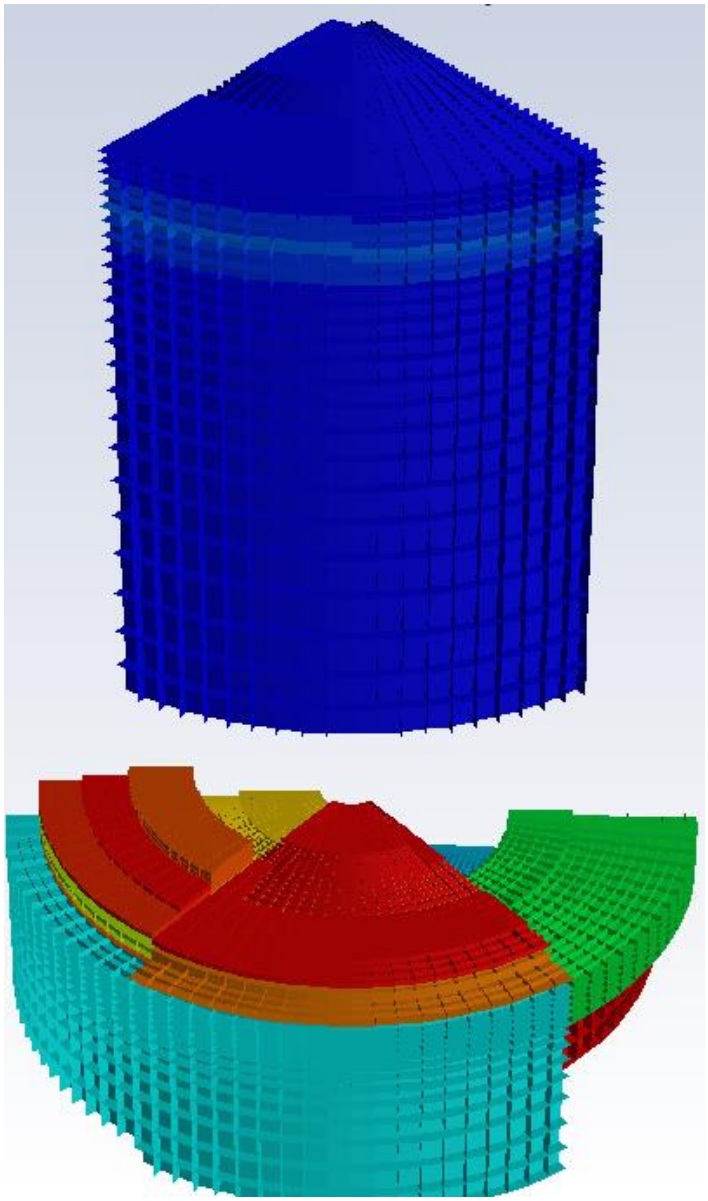


Figure 13. 3D rendering of the middle part of the dynamically comparted bioreactor. *The allocation method succeeds in merging geometric compartments in all three dimensions as can be seen in this figure. Every colour corresponds to an individual compartment, while the geometric compartments are added as an overlay and are visible as small squares. The middle part of the bioreactor is shown, together with the one compartment from the top part of the bioreactor. See Appendix 3.3 for enlarged version.*

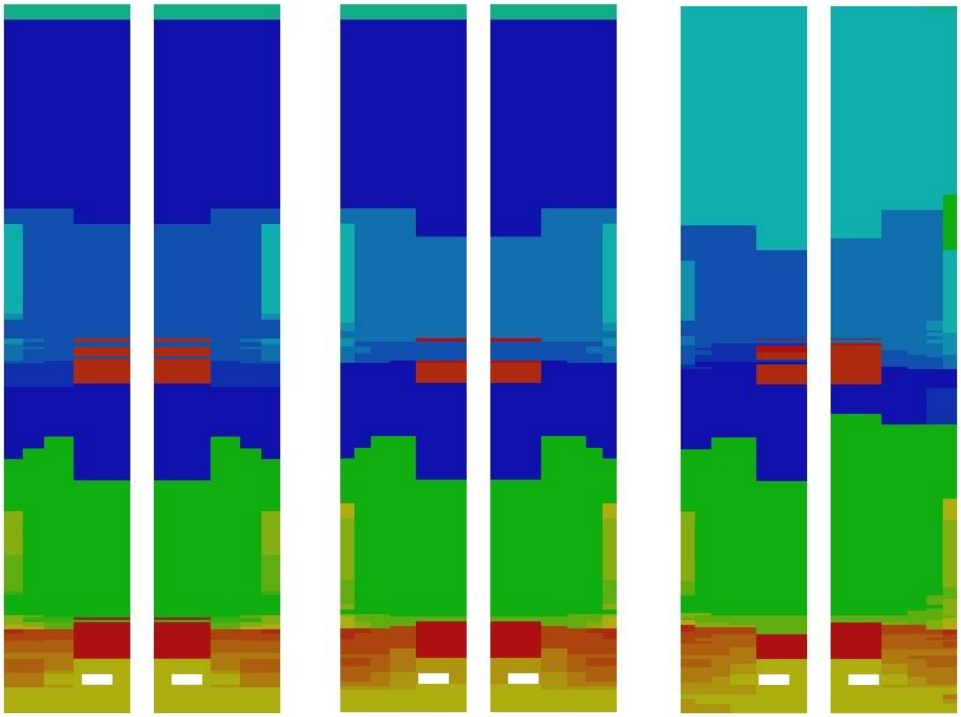


Figure 14. Dynamic compartmentalization with increasing numbers of initial geometric compartments*. From left to right the systems are of the A1250R4T1, A2000R5T1 and A2000R5T5 types, respectively, and therefore correspondingly contain 5000, 10,000 and 50,000 geometric compartments. The accuracy of the dynamic compartments increases slightly, but is not significantly visible, because the compartmentalization strategy succeeds in consistently reducing the number of compartments to 655, 830 and 2566 respectively. The vast number of geometric compartments in the A2000R5T5 system appears to result in a distorted compartmentalization, since already merged compartments cannot be unmerged, even if they would theoretically better fit in with a different cluster. See Appendix 3.4 for enlarged version.*

The allocation method successfully produced a dynamically-rendered compartmentalization model by merging the majority of geometric compartments. The ability of the model to reduce the number of geometric compartments by orders of magnitude makes the allocation method a handy tool for major improvements to calculation times and computational power requirements. However, when a vast number of initial geometric compartments is used, the time it takes for the Ansys compiler to execute the program is increased dramatically: from a split second for a moderate number of geometric compartments to upwards of ten minutes for 50,000 geometric compartments.

Since the mixing script was exclusively designed to determine the mixing time of a geometrically divided model (see Appendix 1.6), implementation of mixing time calculations in the case of the dynamically compartmentalized model was unfortunately not achieved. A first future addition to the compartmenting model of this work is, therefore, altering the mixing script to make it work on the dynamic compartments.

Moreover, volumetrically small compartments are not filtered out, while they do unnecessarily increase the number of calculations. A future project could attempt to tackle the above-mentioned shortcoming by, for example, implementing a loop in which the compartment volumes are determined and, if they fit within a secondary tolerance range for the volume, the respective compartment must be merged using an expanded primary tolerance range (the homogeneity factor for the average compartment velocities).

Lastly, this research focused on single-phase systems, but always kept the possibility of multi-phase extension available. Therefore, expanding on the work of this project to include the gas-phase would be a fantastic next step in the evolution of this project.

Conclusion

Syngas fermentation modelling is a challenging process, which demands a lot of computational power to perform. Drastically reducing the workload on the CPU, without losing accuracy of results, is the principal goal of all compartment models. During this research it was intended to improve on previous work on dynamic compartmentalization by firstly make a geometric compartmenting script modular and easy to use and secondly, designing a creative way of forming dynamic compartments.

The first goal of functionalizing a geometric compartmenting script was validated by comparing the intercompartmental exchanges and the mixing time. From both the validations it can be concluded that the functionalized script is a successful emulation of the geometric compartmenting script and can thus be used in future projects.

The second goal of this work, the designing of a dynamic compartmenting model, was attempted in two ways, namely a data extraction, sorting and splitting strategy imitated from previous work on the topic by Tajsoleiman *et al* (Tajsoleiman, T., *et al.* 2019). And a second, novel strategy of in-CFD compartmenting. The strategy taken from literature was successful to the extent that the data was sorted using a heap sort algorithm, which is a more suitable sorting algorithm compared with the bubble sort (Astrachan, O., 2003). The subsequent splitting of the data was not successful, however. Therefore, further steps, from small compartment elimination to discontinuous compartment classification, were not attempted.
The novel strategy, grounded on geometric compartments, was able to conclusively construct dynamic compartments based on the liquid flow velocities in the bioreactor. The compartments show that most of the significant liquid flows in terms of magnitude and big changes in direction (i.e. vortices), occur around the impellers, in the bottom of the reactor and along the walls. It was expected that the bioreactor boundaries would show increased flow activity, since the momentum of the flow is redirected suddenly by the stationary walls. The impellers were also expected to have a profound effect on the liquid flow. Unexpectedly, though, is the big region in the top of the reactor in which the liquid flow does not show a gradient. It can hypothesized that mass transfer in the top region is not optimal and therefore reduces the overall syngas fermentation.

The sorting method for dynamic compartmenting had as strong point that it did not necessitate the usage of the entire available CFD data, but the caveat is exactly that: a loss in CFD data, which could otherwise be used for further calculations. Another weak point for the sorting method is the reliance on data sorting, which is counterproductive in its usage, since the computational time for data sets of hundreds of thousands of rows in the order of magnitude of tens of minutes. Moreover, data sorting method requires more subsequent steps in order to produce valid compartments, namely the elimination of small compartments and a solution for discontinuous compartments.

The allocation method, on the contrary, does not have the same disadvantages. Data is not sorted, but compartments are instead constructed in-CFD. Furthermore, discontinuous compartments are not an issue, since inherently the allocation method only merges neighboring compartments. Small compartments could, however be present, so a solution should be thought of in future projects. A central strong point of the allocation method is that the homogeneity factor is determined in automatic fashion. Therefore, allowing future users to define the number of compartments they want to dynamically form.

Looking at the bigger picture, this research is a logical next step in the evolution of compartment models. The forming of compartments in-CFD is a stepping stone towards dynamic compartments without data loss and without disruption of work flow. Furthermore, the use of the very fast C language for compartmenting is also a boon for future users. Moreover, the addition of a homogeneity factor which is determined automatically has not been done before.

Future research should focus on the continuation of the allocation method by eliminating small volume compartments and fine-tuning the automatic implementation of the homogeneity factor. Furthermore, mixing time calculations should be performed in order to validate the dynamic compartment model. Moreover, the gas-phase has not been implemented in the present model. Therefore, in order to truly model syngas fermentation, the gas-phase should be added to the dynamic compartment model in future work. Additionally, the modularity of the functionalized script and its switches and user-inputted data should be transferred to a GUI for ultimate ease of use.

References

(2020). *Hydropower generators produce clean electricity, but hydropower does affect the environment*. U.S. Energy Information Administration. https://www.eia.gov/energyexplained/hydropower/hydropower-and-the-environment.php.

Appiah, O. & Martey, E.M. (2015). Magnetic bubble sort algorithm. *Int. Jour. Comp. App.*. 122(**21**).

Astrachan, O. (2003). Bubble Sort: An Archaeological Algorithmic Analysis. *Duke University*, *Computer Science Department.*

Benalcázar, A.E., Noorman, H., Maciel Filho, R. & Posada, J.A. (2020). Modeling ethanol production through gas fermentation: a biothermodynamics and mass transfer-based hybrid model for microbial growth in a large-scale bubble column bioreactor. *Biotechnol. Biofuels.* 13:1–19. https://doi.org/10.1186/s13068-020-01695-y.

Benevenuti, C., Botelho, A., Ribeiro, R., Branco, M., Pereira, A., Vieira, A.C., Ferreira, T. & Amaral, P. (2020). Experimental Design to Improve Cell Growth and Ethanol Production in Syngas Fermentation by *Clostridium carboxidivorans*. *Catalysts*. 10(1): 59. https://doi.org/10.3390/catal10010059

Bredwell, M.D., Srivastava, P. & Worden, R.M. (2008). Reactor design issues for synthesis-gas fermentations. *Biotechnology Progress*. 15(**5**). 834-844.

Committee on Environmental Impacts of Wind Energy Projects, National Research Council, Board on Environmental Studies and Toxicology, & Division on Earth and Life Studies. (2007).

Delafosse, A., Calvo, S., Collignon, M., Delvigne, F., Crine, M. & Toye, D. (2015). Euler–Lagrange approach to model heterogeneities in stirred tank bioreactors – Comparison to experimental flow characterization and particle tracking. *Chemical Engineering Science*. 134. 457-466.

Delafosse, A., Collignon, M, Calvo, S., Delvigne, F., Crine, M., Thonart, P. & Toye, D. (2014). CFD-based compartment model for description of mixing in bioreactors. *Chem. Eng. Sci.* 106:76-85.

DiPippo, R. (2016). *Geothermal power plants: principles, applications, case studies, and environmental impact* [E-book]. Amsterdam: Butterworth-Heinemann.

Engineering ToolBox, (2003). Fuels - Higher and Lower Calorific Values. [online] Available at: https://www.engineeringtoolbox.com/fuels-higher-calorific-values-d_169.

Environmental Impacts of Wind-Energy Projects [E-book]. National Academies Press.

Haringa, C. (2022). An analysis of organism lifelines in an industrial bioreactor using Lattice-Boltzmann CFD. *Eng. Life Sci.* 1-16.

Haringa, C., Tang, W. & Noorman, H.J. (2022). Stochastic parcel tracking in an Euler–Lagrange compartment model for fast simulation of fermentation processes. *Biotech. & Bioeng.*

Hartmann, H., Derksen, J.J. & van den Akker, H.E.A. (2006). Mixing times in a turbulent stirred tank by means of LES. *Fluid Mech. And Transp. Ph.* 52(**11**):3696-3706.

Heindel, T.J. & Kadic, E. (2014). An introduction to bioreactor hydrodynamics and gas-liquid mass transfer. *John Wiley & Sons.*

IEA (2021), Global Energy Review 2021, IEA, Paris https://www.iea.org/reports/global-energy-review-2021.

Istiqomah, N.A., Kresnowati, M.T.A.P. & Setiadi, T. (2021). Syngas fermentation for production of ethanol. *IOP Conf. Series: Materials Science and Engineering*. 1143.

Jourdan, N., Neveux, T., Potier, O., Kanniche, M., Wicks, J., Nopens, I., Rehman, U. & Le Moullec, Y. (2019). Compartmental modelling in chemical engineering: a critical review. *Chem. Eng. Sci*. 210:115-196.

Klemens, B. (2014). 21st century C. O'Reilly. 9781491903896.

Kuo, G. (2019). When fossil fuels run out, what then?. https://mahb.stanford.edu/library-item/fossil-fuels-run/.

Le Moullec, Y., Gentric, C., Potier, O. & Leclerc J.P. (2010). Comparison of systemic, compartmental and CFD modelling approaches: Application to the simulation of a biological reactor of wastewater treatment. *Chemical Engineering Science*. ISSN: 00092509. DOI: 10.1016/j.ces.2009.06.035.

Li, X., Griffin, D., Li, X. & Henson, M.A. (2019). Incorporating hydrodynamics into spatiotemporal metabolic models of bubble column gas fermentation. *Biotechnol. Bioeng*. 116:28–40. https://doi.org/10.1002/bit.26848.

Lloyd-Jones, R. & Lewis, M. (1998). British Industrial Capitalism since the Industrial Revolution. *Routledge*. 1st edition. https://doi.org/10.4324/9781315072487

Marcellino, M., Pratama, D.W., Suntiarko, S.S. & Margi, K. (2021). Comparative of advanced sorting algorithms (quick sort, heap sort, merge sort, intro sort, radix sort) based on time and memory usage. *ICCSAI*. 154-160.

Mims, C. (2012). Moore's law over, supercomputing "in triage," says expert. [online] Available at: https://www.technologyreview.com/2012/05/09/186142/moores-law-over-supercomputing-in-triage-says-expert/.

Mittal, S. (2014). A survey of techniques for improving energy efficiency in embedded computing systems. *International Journal of Computer Aided Engineering and Technology*. 6(**4**). 440–459.

Monir, M.U., Abd Aziz, A., Yousuf, A. & Alam, Md. Z. (2020). Hydrogen-rich syngas fermentation for bioethanol production using *Sacharomyces cerevisiea. Int. Journal of Hydrogen Energy.* 45:36. 18241-18249.

Munasinghe, P.C. & Khanal, S.K. (2010). Biomass-derived syngas fermentation into biofuels: Opportunities and challenges. *Bioresource Technology*. 10:13. 5013-5022.

Nanda, S., Azargohar, R., Dalai, A.K. & Kozinski, J.A. (2014). An assessment on the sustainability of lignocellulosic biomass for biorefining. *Renewable and Sustainable Energy Reviews*. 50:925-941.

Oosterhuis, N.M.G. (1984). Scale-up of bioreactors: a scale-down approach. *Technische Hogeschool Delft*.

Ritchie, H., Roser, M. & Rosado, P. (2020). CO_2 and Greenhouse Gas Emissions. Published online at OurWorldInData.org. Retrieved from: 'https://ourworldindata.org/co2-and-other-greenhouse-gas-emissions'.

Ruggiero, G., Lanzillo, F., Raganati, F., Russo, M.E., Salatino, P. & Marzocchella, A. (2022). Bioreactor modelling for syngas fermentation: kinetic characterization. *Food and Bioproducts Processing*. https://doi.org/10.1016/j.fbp.2022.04.002.

Siebler, F., Lapin, A., Hermann, M. & Takors, R. (2019). The impact of CO gradients on *C. Ijungdahlii* in a 125 m3 bubble column: Mass transfer, circulation time and lifeline analysis. *Chem. Eng. Sci*. 207:410-423.

Tajsoleiman, T., Spann, R., Bach, C., Gernaey, K.V., Huusom, J.K. & Kruhne, U. (2019). A CFD based automatic method for compartment model development. *Comp. and Chem. Eng.* 123:236-245.

Weiss, A. & Diekert, V. (2015). QuickHeapsort: Modifications and Improved Analysis. *Theory of Computing Systems*. 59:209-230.

Xiang, Y., Luo, H., Liu, G. & Zhang, R. (2022). Improvement of organic acid production with sulfate addition during syngas fermentation using mixed cultures. *Water Cycle*. Vol. 3. 26-34.

Appendices

Appendix 1 – Scripting & Programming

1.1 Script execution tutorial

With a multitude of C scripts to work with, it can be overwhelming and unclear which scripts to run and when. For that reason, a small tutorial is included for ease-of-use.

In order to run the base script, a header file is needed as a prerequisite. Moreover, an essential file is the compartment inputs file, which contains data about the dimensions of the bioreactor in question and holds user-input information on the number of geometric compartments to use in the axial, radial and tangential directions (figure 2). Furthermore, when the utilization of time-average values is desired, the averaging script has to be executed before running the base script:

1. Make sure the ("InputCompartmentScript.txt") is present in the main working directory
2. When in Ansys Fluent, load the case study ("BaseCase_mix_edited.cas")
3. User-Defined > Functions > Compiled...
4. Under Source Files choose ("Averaging_Script.c") and under Header Files choose ("Header_File.h")
5. Build > Load
6. User-Defined > Execute on Demand... > choose the just build and loaded file

Now that the averaging data is saved to memory and available for use, it is possible to execute the functionalized script:

1. Repeat steps 1, 2 and 3 from the above-mentioned steps
2. Under Source Files choose ("Functionalized_Script.c") and under Header Files choose ("Header_File.h")
3. Repeat steps 5 and 6 from the above-mentioned steps

Next, in order to run the compartmentalization script, perform the following steps:

1. Repeat steps 1, 2 and 3 from the previously-mentioned steps
2. Under Source Files choose ("Compartmentalization_Script.c") and under Header Files choose ("Header_File.h")
3. Repeat steps 5 and 6 from the previously-mentioned steps

Finally, post-processing is done in Visual Studio Code, implementing a script in Julia:

1. When in Visual Studio Code, load the post-processing file ("compartment_eulerian_mixer.jl")
2. Edit the folder data in the second block to correspond to the user's situation
3. Execute the active file in REPL

1.2 The averaging script

Calculation of average values for the x, y and z direction liquid velocity, the x, y and z direction gas velocity, the turbulent kinetic energy, the dissipation of the turbulent kinetic energy and the gas fraction in the bioreactor is necessary if average values are desired for the determination of the convective flux and turbulent kinetic energy from the CFD model. The Ansys macro's used are the following:

C_UDMI	Storing of data for later use
C_U	Extraction of x-direction velocity
C_V	Extraction of y-direction velocity
C_W	Extraction of z-direction velocity
C_K	Turbulent kinetic energy
C_D	Turbulent energy dissipation rate
C_VOF	The volume of fraction, i.e. gas fraction

These macro's collect data at the grid cell level; therefore, a grid cell loop is used to loop over all grid cells.

```
begin_c_loop (cell,tr)
{
    /* velocity averages & stdev, liquid */
    BU = C_UDMI(cell,tr,0);
    C_UDMI(cell,tr,0) = (C_UDMI(cell,tr,0)*(timer-1) + C_U(cell,SUBT_Liq))/(timer) ;
    M2 = C_UDMI(cell,tr,6)+((C_U(cell,SUBT_Liq) - BU)*(C_U(cell,SUBT_Liq)-C_UDMI(cell,tr,0)));
    C_UDMI(cell,tr,6) = M2/timer;

    BU = C_UDMI(cell,tr,1);
    C_UDMI(cell,tr,1) = (C_UDMI(cell,tr,1)*(timer-1) + C_V(cell,SUBT_Liq))/(timer) ;
    M2 = C_UDMI(cell,tr,7)+((C_V(cell,SUBT_Liq) - BU)*(C_V(cell,SUBT_Liq)-C_UDMI(cell,tr,1)));
    C_UDMI(cell,tr,7) = M2/timer;

    BU = C_UDMI(cell,tr,2);
    C_UDMI(cell,tr,2) = (C_UDMI(cell,tr,2)*(timer-1) + C_W(cell,SUBT_Liq))/(timer) ;
    M2 = C_UDMI(cell,tr,8)+((C_W(cell,SUBT_Liq) - BU)*(C_W(cell,SUBT_Liq)-C_UDMI(cell,tr,2)));
    C_UDMI(cell,tr,8) = M2/timer;

    /* velocity averages & stdev, gas */
    BU = C_UDMI(cell,tr,3);
    C_UDMI(cell,tr,3) = (C_UDMI(cell,tr,3)*(timer-1) + C_U(cell,SUBT_Gas))/(timer) ;
    M2 = C_UDMI(cell,tr,9)+((C_U(cell,SUBT_Gas) - BU)*(C_U(cell,SUBT_Gas)-C_UDMI(cell,tr,3)));
    C_UDMI(cell,tr,9) = M2/timer;

    BU = C_UDMI(cell,tr,4);
    C_UDMI(cell,tr,4) = (C_UDMI(cell,tr,4)*(timer-1) + C_V(cell,SUBT_Gas))/(timer) ;
    M2 = C_UDMI(cell,tr,10)+((C_V(cell,SUBT_Gas) - BU)*(C_V(cell,SUBT_Gas)-C_UDMI(cell,tr,4)));
    C_UDMI(cell,tr,10) = M2/timer;

    BU = C_UDMI(cell,tr,5);
    C_UDMI(cell,tr,5) = (C_UDMI(cell,tr,5)*(timer-1) + C_W(cell,SUBT_Gas))/(timer) ;
    M2 = C_UDMI(cell,tr,11)+((C_W(cell,SUBT_Gas) - BU)*(C_W(cell,SUBT_Gas)-C_UDMI(cell,tr,5)));
    C_UDMI(cell,tr,11) = M2/timer;

    /* turbulence averages & stdev, gas */
    BU = C_UDMI(cell,tr,12);
    C_UDMI(cell,tr,12) = (C_UDMI(cell,tr,12)*(timer-1) + C_K(cell,SUBT_Liq))/(timer) ;
    M2 = C_UDMI(cell,tr,15)+((C_K(cell,SUBT_Liq) - BU)*(C_K(cell,SUBT_Liq)-C_UDMI(cell,tr,12)));
    C_UDMI(cell,tr,15) = M2/timer;

    BU = C_UDMI(cell,tr,13);
    C_UDMI(cell,tr,13) = (C_UDMI(cell,tr,13)*(timer-1) + C_D(cell,SUBT_Liq))/(timer) ;
    M2 = C_UDMI(cell,tr,16)+((C_D(cell,SUBT_Liq) - BU)*(C_D(cell,SUBT_Liq)-C_UDMI(cell,tr,13)));
    C_UDMI(cell,tr,16) = M2/timer;

    BU = C_UDMI(cell,tr,14);
    C_UDMI(cell,tr,14) = (C_UDMI(cell,tr,14)*(timer-1) + C_VOF(cell,SUBT_Gas))/(timer) ;
    M2 = C_UDMI(cell,tr,17)+((C_VOF(cell,SUBT_Gas) - BU)*(C_VOF(cell,SUBT_Gas)-C_UDMI(cell,tr,14)));
    C_UDMI(cell,tr,17) = M2/timer;
}
end_c_loop (cell,tr)
```

1.3 The base script

The base script forms the basis for the construction of geometric compartments, the calculations for the convective fluxes and the turbulent kinetic energy transfers and the exportation of the calculated data to external files.

Formation of the geometric compartments starts with reading an input file which contains the user-desired information about the number of axial, radial and tangential compartments:

```
fscanf(fp,"%*s %d", &NCT);
fscanf(fp,"%*s %d", &NCR);
fscanf(fp,"%*s %d", &NCZ);
fscanf(fp,"%*s %lf", &height);
fscanf(fp,"%*s %lf", &diameter);
```

NCT	Number of tangential compartments
NCR	Number of radial compartments
NCZ	Number of axial compartments

Then the grid points of the geometric compartments are determined, in order to know their respective boundaries in between which the grid cells are put:

```
t_grid[0] = t_min;
for (i = 1; i<NCT+1; i++)
{
    t_grid[i] = t_grid[i-1] + t_range/NCT;
}

r_grid[0] = r_min;
for (j = 1; j<NCR+1; j++)
{
    r_grid[j] = sqrt((double) j/NCR)*r_max;
}

z_grid[0] = z_min;
for (k = 1; k<NCZ+1; k++)
{
    z_grid[k] = z_grid[k-1] + z_range/NCZ;
}
```

Then the geographical locations of the cells are determined inside of a loop, in order to allocate the cells to their corresponding geometric compartment.

```
begin_c_loop(c, t)
{

    C_CENTROID(cp,c,t);
    x_pos = cp[0];
    y_pos = cp[2];
    z_pos = cp[1];
```

C_CENTROID	Extraction of the coordinates of the grid cell

Then the grid cells are fitted between the geometric compartment boundaries. Henceforth the geometric compartments are filled with their corresponding grid cells and are thus formed.

```
for (i = 0; i < NCT; i++)
{
    t_pos = theta_pos(x_pos,y_pos,angularref);
    C_UDMI(c,t,1) = t_pos;
    if (t_pos >= (double)t_grid[i] && t_pos <= (double)t_grid[i+1])
    {
        fi = i;
        flagi = 1;
    }
}

for (j = 0; j < NCR; j++)
{
    r_pos = rad_pos(x_pos,y_pos);
    if (r_pos >= (double)r_grid[j] && (double)r_pos <= (double)r_grid[j+1])
    {
        fj = j;
        flagj = 1;
    }
}

for (k = 0; k < NCZ; k++)
{
    if ((double)z_pos >= (double)z_grid[k] && (double)z_pos <= (double)z_grid[k+1])
    {
        fk = k;
        flagk = 1;
    }
}
```

The base script then continues to calculate the convective fluxes and the turbulent kinetic energies and also contains code to export the data to external files. These are all explained in appendix 1.4.

1.4 The functionalized script

The functionalized script uses the base script as basis and converts the convective flux and turbulent kinetic energy calculations and the data exporting parts to nested-functions for increased modularity and ease of use.

At first, the designed functions have to be prototyped for the C compiler to understand the functions and their arguments when the functions are called:

```
void PhaseFluxIO(int threadtype, int threadChoice, double* phase_flux, int* comp0, int* comp1);
void PhaseFluxIntC(int threadChoice, double* phase_flux, int* comp0, int* comp1);
void TurbulenceIntC(int threadChoice, double* turb_ex, int* comp0, int* comp1);
void Compartment_Writing(int NC, char* Folder_location, char* Filename, double** MatrixVariable);
void Volume_Writing(int NC, char* Folder_location, char* Filename, double* Variable);
```

1.4.1 PhaseFluxIO();

The function PhaseFluxIO(); calculates the convective flux for grid cells which are located at an inlet or an outlet of the bioreactor. The function first translates the user-input for the 'threadChoice' argument into the relevant thread:

0	Mixture thread (single-phase)	f_thread
1	Liquid thread (multi-phase)	f_subt_liq
2	Gas thread (multi-phase)	f_subt_gas

```
if (threadChoice == 0){
threadC = f_thread;}
else if (threadChoice == 1){
threadC = f_subt_liq;}
else if (threadChoice == 2){
threadC = f_subt_gas;}
```

Then, both sides of the grid cell face are scanned for the presence of a geometric compartment. Because inlet and outlet grid cells are located at the boundary of the bioreactor, no neighboring compartment exists. Therefore, an imaginary environment compartment, called N_IOC, is added.

```
/*Select compartment on left side of face
and return that value for global use*/
*comp0 = C_UDMI(c0,t0,0);
/*Select compartment on right side of face
and return that value for global use*/
*comp1 = N_IOC-1;
```

Subsequently, the relevant thread is used, optionally together with the time-averaged data if set, in order to calculate the convective flux between two neighboring compartments.

```
if (threadChoice == 1 || threadChoice == 2)
{
    /*If true will calculate the convective flux using
    average values from the user defined memory;
    Must run averaging script for this to work*/
    if(SWITCH_AV)
    {
        /*First the velocity vector is filled with the relevant face
        information from the UDM --> 1 is u-direction,
        2 is v-direction, 3 is w-direction*/
        NV_D(vel,=,F_UDMI(c0,t0,1), F_UDMI(c0,t0,2),F_UDMI(c0,t0,3));
        /*Then the average liquid hold up is determined,
        which is 1 - the average gas hold up,
        information on which is stored in UDM 9*/
        AVHO = (1-F_UDMI(c0,t0,9));
        /*Finally, the phase flux is calculated by multiplying
        the average liquid hold up with the dot product of the
        velocity vector and the face area*/
        *phase_flux = AVHO*NV_DOT(vel,farea);
    }
    /*Else if average values should not be used*/
    else
    {
        /*Take the density of the relevant subthread,
        which is set by threadChoice*/
        rho_l = F_R(f,threadC);
        /*Then calculate the phase-flux for the relevant
        subthread using the local density*/
        *phase_flux = F_FLUX(f,threadC)/rho_l;
    }
}
/*If singlephase, use instantaneous values since
steady-state is assumed*/
else if (threadChoice == 0)
{
    /*Take the density of the relevant cell*/
    rho_l = C_R(c0,t0);
    /*Then calculate the phase-flux for the relevant
    subthread using the local mesh cell density*/
    *phase_flux = F_FLUX(f,threadC)/rho_l;
}
```

1.4.2 PhaseFluxIntC();

The function PhaseFluxIntC(); is used when the grid cell is an internal cell. Hereby it is important that only the convective flux between two grid cells, which are located on opposite sides of the geometric compartment border, are taken into consideration:

```
/*Only take into account adjacent compartments,
intra-compartment exchanges not needed*/
if (comp0 != comp1)
{
```

The script also distinguishes between internal grid cells at the boundaries of the bioreactor and grid cells not at the boundaries. For the grid cells at the boundaries, in the case of instantaneous value usage, the density is assumed homogeneous between the phases and therefore the density from the mixture phase is taken:

```
if (BOUNDARY_FACE_THREAD_P(f_thread))   /*If boundary face*/
{
    if (threadChoice == 1 || threadChoice == 2)
    {
        if (SWITCH_AV)
        {
            /*Fill in the velocity vector using u-, v- & w-direction liquid velocitiy
            values of the cells on the left side of the face*/
            NV_D(vel,=, C_UDMI(c0,t0,1), C_UDMI(c0,t0,2), C_UDMI(c0,t0,3));
            /*Add the u-, v- & w-direction liquid velocitiy values of the cells
            on the right side of the face*/
            NV_D(vel,+=, C_UDMI(c1,t1,1), C_UDMI(c1,t1,2), C_UDMI(c1,t1,3));
            /*Calculate the average liquid hold up of the two cells*/
            AVHO = ( ( 1-C_UDMI(c0,t0,9) ) + ( 1-C_UDMI(c1,t1,9) ) ) / 2.;
            /*Calculate the phase flux by multiplying the average liquid hold up
            with the dot product of the velocity and face area vectors
            and make it global*/
            *phase_flux = AVHO*NV_DOT(vel,farea)/2.0;
        }
         /*If averaged values are not desired, use instantaneous values instead*/
        else
        {
            /*Calculate the liquid density*/
            rho_l = F_R(f,threadC);
            /*Calculate the phase flux and make it global*/
            *phase_flux = F_FLUX(f,threadC)/rho_l;
        }
    }
    /*If singlephase, use threadChoice == 0*/
    else if (threadChoice == 0)
    {
        *phase_flux = F_FLUX(f,threadC)/rho_l;
    }
}
```

Else if the internal grid cells are not located at the boundaries of the bioreactor, the density can not be assumed homogeneous and must thus be calculated separately for both the liquid and gas phase:

```
else /* if not a boundary face thread use alternative formulation */
{
    if (threadChoice == 1 || threadChoice == 2)
    {
        /* if averaging */
        if(SWITCH_AV)
        {
            /*Fill in the velocity vector using u-, v- & w-direction liquid velocitiy
            values of the cells on the left side of the face*/
            NV_D(vel,=, C_UDMI(c0,t0,1), C_UDMI(c0,t0,2), C_UDMI(c0,t0,3));
            /*Add the u-, v- & w-direction liquid velocitiy values of the cells
            on the right side of the face*/
            NV_D(vel,+=, C_UDMI(c1,t1,1), C_UDMI(c1,t1,2), C_UDMI(c1,t1,3));
            /*Calculate the average liquid hold up of the two cells*/
            AVHO = ( ( 1-C_UDMI(c0,t0,9) ) + ( 1-C_UDMI(c1,t1,9) ) ) / 2.;
            /*Calculate the phase flux by multiplying the average liquid hold up
            with the dot product of the velocity and face area vectors
            and make it global*/
            *phase_flux = AVHO*NV_DOT(vel,farea)/2.0;
        }
        /* if instant... */
        else
        {
            /*If liquid phase*/
            if (threadChoice == 1)
            {
                /*Point the solver to the liquid thread*/
                subt_liq_1 = THREAD_SUB_THREAD(t0,0);
                /*get fluid density*/
                rho_l = C_R(c0,subt_liq_1);
                /*and get flux*/
                *phase_flux = F_FLUX(f,threadC)/rho_l;
            }
            /*If gas phase*/
            else if (threadChoice == 2)
            {
                /*Point the solver to the gas thread*/
                subt_gas_1 = THREAD_SUB_THREAD(t0,1);
                /*get gas density*/
                rho_l = C_R(c0,subt_gas_1);
                /*and get flux*/
                *phase_flux = F_FLUX(f,threadC)/rho_l;
            }
        }
    }
    /*If singlephase*/
    else if (threadChoice == 0)
    {
        /*Calculate flux*/
        *phase_flux = F_FLUX(f,threadC)/rho_l;
    }
}
```

1.4.3 TurbulenceIntC();

The calculations for the turbulent kinetic energy transfers between neighboring compartments are performed using the function TurbulenceIntC();. This function, like function PhaseFluxIntC(); (Appendix 1.4.2), solely considers grid cells located on the border between two compartments: intra-compartment fluxes are not needed:

```
/*Only take into account adjacent compartments,
intra-compartment exchanges not needed*/
if (comp0 != comp1)
{
```

Again, like function PhaseFluxIntC(); (Appendix 1.4.2), a distinction is made between grid cells at the boundary of the bioreactor, of which the subsequent calculations are shown below, and grid cells in the interior:

```
/*If boundary face*/
if (BOUNDARY_FACE_THREAD_P(f_thread))
{
    /*If multiphase*/
    if (threadChoice == 1 || threadChoice == 2)
    {
        /* if averaging */
        if (SWITCH_AV)
        {
            /*Calculate the average liquid hold up of the two cells*/
            AVHO = ( ( 1-C_UDMI(c0,t0,9) ) + ( 1-C_UDMI(c1,t1,9) ) ) / 2.;
            /*Calculate the phase area*/
            phase_area = AVHO * NV_MAG(farea);
            /*Calculate the turbulent kinetic energy*/
            *turb_ex  = (sqrt(C_UDMI(c0,t0,7)*2./3.) + sqrt(C_UDMI(c1,t1,7)*2./3.)) * phase_area/2.;
        }
        /*If averaged values are not desired, use instantaneous values instead*/
        else
        {
            *turb_ex  = (sqrt(F_K(f,f_thread)*2./3.)) * area * F_VOF(f,threadC);
        }
    }
    /*If singlephase*/
    else if (threadChoice == 0)
    {
        *turb_ex  = (sqrt(F_K(f,f_thread)*2./3.)) * area;
    }
}
```

For interior grid cells the calculations performed look as follows:

```
else /* if not a boundary face thread use alternative formulation */
{
    /*If multiphase*/
    if (threadChoice == 1 || threadChoice == 2) /*If multiphase, provide possibility to use averaged values*/
    {
        /* if averaging */
        if(SWITCH_AV)
        {
            AVHO = ((1-C_UDMI(c0,t0,9)) + (1-C_UDMI(c1,t1,9)))/2.;
            phase_area = AVHO * NV_MAG(farea);
            *turb_ex  = (sqrt(C_UDMI(c0,t0,7)*2./3.) + sqrt(C_UDMI(c1,t1,7)*2./3.)) * phase_area/2.;
        }
        /* if instant... */
        else
        {
            *turb_ex  = (sqrt(C_K(c0,t0)*2./3.) + sqrt(C_K(c1,t1)*2./3.)) * area/2. * F_VOF(f,threadC);
        }
    }
    /*If singlephase*/
    else if (threadChoice == 0)
    {
        *turb_ex  = (sqrt(C_K(c0,t0)*2./3.)+sqrt(C_K(c1,t1)*2./3.))*area/2.;
    }
}
```

1.4.4 Compartment_writing();

In order to export the obtained convective flux and turbulent kinetic energy data to readable files, the function Compartment_writing(); is made. This function takes as arguments the folder location and filename as well as the dimensions of the matrix. In order to prevent the replacement of memory, a variable, temp1, is utilized, which temporarily holds the data from memory location 'MatrixVariable[n][j]'. Then the data is printed with 17 significant decimals in the shorter of the C data types double or exponential:

```
/*Set pathname and open file*/
/*Take pathname and filename from function arguments*/
sprintf(PATHNAME_Comp, "%s%s", Folder_location, Filename);
/*Open the above-specified file in write mode*/
CompOutput = fopen(PATHNAME_Comp, "w");

/*Loop through rows of the matrix*/
for (n=0; n<NC; n++)
{
    /*Loop through columns of the matrix*/
    for(j=0; j<NC; j++)
    {
        /*Fill the temporary empty double, temp1,
        with the row and column data values*/
        temp1 = MatrixVariable[n][j];
        /*Print the data of temp1 in the opened file*/
        fprintf(CompOutput, "%.17g\t", temp1);
    }
    /*After all columns of a row are looped through
    go to next line before looping through the next row*/
    fprintf(CompOutput, "\n");
}
/*Close the file, thereby saving the data*/
fclose(CompOutput);
```

1.4.5 Volume_writing();

The same idea as the function in Appendix 1.4.4 applies to the writing of vector-sized compartment data, i.e. the compartment volume or number of grid cells in the each compartment. The only difference between this function and Compartment_writing();, is that this function only applies one loop:

```
/*Take pathname and filename from function arguments*/
sprintf(PATHNAME_Vol, "%s%s", Folder_location, Filename);
/*Open the above-specified file*/
VolOutput = fopen(PATHNAME_Vol, "w");

/*Only need to loop through rows for volume-related data*/
for (n=0; n<NC; n++)
{
    /*Fill the temporary empty double, temp1, with the row data values*/
    temp1 = Variable[n];
    /*Print the data of temp1 in the opened file*/
    fprintf(VolOutput, "%.17g\n", temp1);
}
/*Close the file, thereby saving the data*/
fclose(VolOutput);
```

1.5 Compartmentalization strategies

Two compartmentalization strategies were employed in this research, the sorting method and the allocation method. Both methods rely on the usage of the homogeneity factor, which determines the range against which the desired hydrodynamic parameter is compared.

1.5.1 Sorting method

The sorting method for dynamic compartmenting relies on extracting the desired hydrodynamic data from the CFD and sorting the data in ascending manner. After sorting of the grid cells, they are divided into groups of equal size based on the range set by the homogeneity factor: the dynamic compartments. The sorting method employs two functions, namely Data_extraction(); and HeapSort();. The former is used to extract from the CFD the desired hydrodynamic parameters and the latter is used to sort the data in ascending order according to the heap sorting algorithm.

1.5.1.1 Data_extraction();

Extraction of the CFD data is done by looping over all grid cells and using Ansys macro's from data extraction. Hereby, the data is stored in a matrix with 7 columns in the case of single-phase or liquid-phase or 8 columns in the case of the gas-phase, since the gas holdup is also an important parameter in syngas fermentation:

```
/*Set chosen thread: 1=liquid phase, 2=gas phase*/
if(threadChoice == 1)
{threadC = f_subt_liq;}
else if(threadChoice == 2)
{threadC = f_subt_gas;}
```

```
/*Extract data of liquid phase*/
if(threadChoice == 1)
{
    /*Macros for tri-directional velocities*/
    NV_D(vel, =, C_U(cell,thread), C_V(cell,thread), C_W(cell,thread));
    x_vel_L = vel[0];
    y_vel_L = vel[1];
    z_vel_L = vel[2];

    /*Macro for grid cell 3D coordinates*/
    C_CENTROID(coords, cell, thread);
    xcoord = coords[0];
    ycoord = coords[1];
    zcoord = coords[2];

    /*Use the NumCells counter to point to current row in matrix*/
    /*Every column is filled with relevent data*/
    mat[NumCells-1][0] = NumCells;
    mat[NumCells-1][1] = x_vel_L;
    mat[NumCells-1][2] = y_vel_L;
    mat[NumCells-1][3] = z_vel_L;
    mat[NumCells-1][4] = xcoord;
    mat[NumCells-1][5] = ycoord;
    mat[NumCells-1][6] = zcoord;
}
```

```
/*Extract data of gas phase*/
else if(threadChoice == 2)
{
    /*Macros for tri-directional velocities and the gas hold up*/
    NV_D(vel, =, C_U(cell,thread), C_V(cell,thread), C_W(cell,thread));
    x_vel_G = vel[0];
    y_vel_G = vel[1];
    z_vel_G = vel[2];
    gas_frac = C_VOF(cell, threadC); /*Gas hold up*/

    /*Macro for grid cell 3D coordinates*/
    C_CENTROID(coords, cell, thread);
    xcoord = coords[0];
    ycoord = coords[1];
    zcoord = coords[2];

    /*Use the NumCells counter to point to current row in matrix*/
    /*Every column is filled with relevent data*/
    mat[NumCells-1][0] = NumCells;
    mat[NumCells-1][1] = x_vel_G;
    mat[NumCells-1][2] = y_vel_G;
    mat[NumCells-1][3] = z_vel_G;
    mat[NumCells-1][4] = gas_frac;
    mat[NumCells-1][5] = xcoord;
    mat[NumCells-1][6] = ycoord;
    mat[NumCells-1][7] = zcoord;
}
```

1.5.1.2 HeapSort();

With the relevant hydrodynamic data available in matrix format, the matrix can be sorted based on one of the parameters. In this work the y-direction liquid velocity is chosen as baseline parameter for sorting, since the velocity field in the axial direction (y-direction in Ansys) was observed to have the most activity.

Heap sorting is based on converting all the data into a so-called max heap; a process which results in the largest value in the array ending up at the first element:

```
/*loop for converting complete binary tree to max heap*/
for(i = nrows/2; i >= 0; i--){
    /*As long as left child exists;
    which means that the end of the array is
    not reached yet*/
    for(j=2*i+1; j < nrows; j = 2*i+1)
    {
        bci = j;
        if(j+1 < nrows)
        {
            /*Select the biggest of the two childs...*/
            if(a[j][2] < a[j+1][2])
            {
                bci = j+1;
            }
        }
        /*...the biggest of the childs is swapped with the parent*/
        if(a[i][2] < a[bci][2])
        {
            /*Swap entire rows*/
            for(m = 0; m < 7; m++){
                temp=a[i][m];
                a[i][m]=a[bci][m];
                a[bci][m]=temp;
            }
        }
        else
            break;
        i=bci;
    }
}
```

Then this first element (the largest element in the array) is swapped with the last element of the array, because ascending order is desired:

```
for(i = nrows-1; i > 0; i--)
{
    /*Swap first element of maxheap with last element of list*/
    /*Swap entire rows*/
    for(m = 0; m < 7; m++){
        temp = a[0][m];
        a[0][m] = a[i][m];
        a[i][m] = temp;
    }
```

After the first max heap has been constructed and used once for placing the largest value at the last element of the array, the max heap is iteratively constructed without taking into account the recently swapped element (i.e. at first iteration: make max heap, then swap first element with last. At second iteration: make max heap without considering the last element, then swap new first element with second to last. At third iteration: make max heap without considering the last two elements, then swap new first element with third to last. Etc.):

```
/*loop for converting complete binary tree iteratively to max heap*/
for(k = i/2; k >= 0; k--){
    /*As long as left child exists;
    which means that the end of the array is
    not reached yet*/
    for(j=2*k+1; j < i; j = 2*k+1)
    {
        bci = j;
        if(j+1 < i){
            /*Select the biggest of the two childs...*/

            if(a[j][2] < a[j+1][2]){
                bci = j+1;
            }
        }
        /*...the biggest of the childs is swapped with the parent*/
        if(a[k][2] < a[bci][2]) /*Sort based on column 2*/
        {
            /*Swap entire rows*/
            for(m = 0; m < 7; m++){
                temp=a[k][m];
                a[k][m]=a[bci][m];
                a[bci][m]=temp;
            }
        }
        else
            break;
        k=bci;
    }
}
```

1.5.1.3 BubbleSort();

Bubble sorting is an algorithm which loops through the data, iteratively comparing an element with the next element. In case the order of those two compared elements is not in ascending order, the two elements are swapped from position. The algorithm continues swapping until the element is not swapped, and therefore has reached its correct location, respecting the ascending order which the algorithm wants to achieve. In that case the next element in the array is considered and the same steps are applied.

```
/*Sorting process using bubble sort technique*/
    /*Loop over all rows*/
    for(i=0; i<nrows-1; i++)
    {
        /*Loop over all columns*/
        for(j=0; j<nrows-1-i; j++)
        {
            /*Sort all columns based on column 1(x vel) */
            if(mat[j][1] > mat[j+1][1])
            {
                for (n=0; n<ncols; n++){
                    temp = mat[j][n];
                    mat[j][n] = mat[j+1][n];
                    mat[j+1][n] = temp;
                }
            }
        }
    }
```

1.5.1.4 Split();

With the data sorted, the sorted matrix should be split into chunks of equal size according to the size definition provided by the homogeneity factor. In this research the Split(); function did not work as intended, as the function splits the sorted matrix column-wise, instead of row-wise. The function theoretically should add a third dimension to the sorted matrix, which contains the numbering scheme for the dynamic compartments:

```
/*Function to split the 2D array into evenly sized sub-2D arrays */
void split(int rows, int cols, double **mat, int chunksize, int chunks[][rows][cols])
{
    int i;
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            /*Adds a third dimension (i/chunksize) which contains
            numbering scheme for split compartments*/
            chunks[i / chunksize][i][j] = mat[i][j];
        }
    }
}
```

1.5.2 Allocation method

To the contrary of the sorting method, the allocation method constructs dynamic compartments in-CFD. The allocation method works in tandem with the base script geometric compartment formation: during geometric compartmenting, for each geometric compartment the velocities are summed:

```
velx_comp[n] += C_U(c,t);   /*Add the x-direction velocities of the cell c,t to its compartment*/
vely_comp[n] += C_V(c,t);   /*Add the y-direction velocities of the cell c,t to its compartment*/
velz_comp[n] += C_W(c,t);   /*Add the z-direction velocities of the cell c,t to its compartment*/
```

Then the average velocities for every geometric compartment are calculated using the number of grid cells out of which the compartment exists:

```
/*Calculate average velocities of the geometric compartments*/
for(n = 0; n < NC; n++){
    avg_velx_comp[n] = velx_comp[n] / (numcells[n]);
    avg_vely_comp[n] = vely_comp[n] / (numcells[n]);
    avg_velz_comp[n] = velz_comp[n] / (numcells[n]);
}
```

Subsequently, the homogeneity factor is automatically calculated using a user estimate of the number of dynamic compartments desired. Hereby, the biggest and smallest of the average compartment velocities are determined and the homogeneity factor calculated. The same calculations are performed for the y and z directions of the velocity:

```
/*Get largest and smallest x-direction velocities*/
bigx = avg_velx_comp[NC-1];
smallx = avg_velx_comp[NC-1];
/*Loop through list and compare each list value with the last value in the list*/
for(i = 0; i < NC; i++){
    numx = avg_velx_comp[i];
    /*"If value numx is larger than the last value in the list,
    then it (numx) is for now the largest value of the entire list"*/
    if(bigx < numx){bigx = numx;}
    /*"After that, compare each list value with the new largest number"*/
    if(smallx > numx){smallx = numx;}
}
/*MACROCOMPS contains the number of macrocomps we want*/
x_vel_diff = (bigx-smallx) / MACROCOMPS;
```

Then, a vector for keeping track which geometric compartment have merged is initialized as a zero vector:

```
/*Initialize dynamic vector with zeros*/
dynvec = (int*) calloc(NC,sizeof(int));
for(i=0; i<NC; i++){dynvec[i] = 0;}
```

A cell face loop is then set in motion. Herein two if statements are first evaluated, namely 'is the cell face located on the border between two geometric compartments?' and 'has any or both of the geometric compartments already been merged before?':

```
/*If cells are on both sides of the border between
geometric compartments*/
if (comp0 != comp1)
{
    /*But first check if either geocomp needs merging at all
    (if either or both geocomps have value 0 at their
    elements in the dynamic vector, dynvec)*/
    if( dynvec[comp0] == 0 || dynvec[comp1] == 0 )
```

Added complexity is achieved by letting the algorithm decide which velocity direction to use for deciding whether to merge geometric compartments. This is achieved by determining the largest of the three average compartment velocities and using that as comparison medium. The homogeneity factor (vel_diff) is subsequently set accordingly:

```
/*Determine which average velocity is the largest in
the geometric compartment (x, y or z-direction)*/
if ( avg_velx_comp[comp0] >= avg_vely_comp[comp0] )
{
    /*x is largest if..*/
    if ( avg_velx_comp[comp0] >= avg_velz_comp[comp0] )
    {
        avg_vel_comp[comp0] = avg_velx_comp[comp0];
        avg_vel_comp[comp1] = avg_velx_comp[comp1];
        vel_diff = x_vel_diff;
    }
    /*z is largest if..*/
    else
    {
        avg_vel_comp[comp0] = avg_velz_comp[comp0];
        avg_vel_comp[comp1] = avg_velz_comp[comp1];
        vel_diff = z_vel_diff;
    }
}
else
{
    /*y is largest if..*/
    if ( avg_vely_comp[comp0] >= avg_velz_comp[comp0] )
    {
        avg_vel_comp[comp0] = avg_vely_comp[comp0];
        avg_vel_comp[comp1] = avg_vely_comp[comp1];
        vel_diff = y_vel_diff;
    }
    /*z is largest if..*/
    else
    {
        avg_vel_comp[comp0] = avg_velz_comp[comp0];
        avg_vel_comp[comp1] = avg_velz_comp[comp1];
        vel_diff = z_vel_diff;
    }
}
```

The homogeneity factor is afterwards used as the range in which both the average velocities of the neighboring compartments should fit if they were to be merged. Moreover, the velocity direction is also taken into consideration. Four possibilities exist:

1. Velocity direction is the same and magnitude is within homogeneity range
2. Velocity direction is not the same, but magnitude is within homogeneity range
3. Velocity direction is the same, but magnitude is not within homogeneity range
4. Velocity direction is not the same and magnitude is not within homogeneity range

First, situation 1:

```
/*Determine whether or not neighboring compartments should be
merged based on velocity difference*/
/*If product of velocities is positive value,
that means velocity directions are the same*/
if(avg_vel_comp[comp0] * avg_vel_comp[comp1] > 0)
{
    /*Subtract absolute average velocities and compare absolute hereof with vel_diff*/
    if(abs(abs(avg_vel_comp[comp0]) - abs(avg_vel_comp[comp1])) <= vel_diff)
    {/*In this part of the code we have pointed to two geocomps which both neighbor each other
       and have approximately the same average velocity magnitude and direction*/
```

The subsequent grid cell loop should naturally only consider grid cells in the compartments in question:

```
/*During looping only consider cells which are in comp0 and comp1*/
if (C_UDMI(c,t,0) == comp0 || C_UDMI(c,t,0) == comp1)
```

Then three situations are distinguished. Firstly, none of the compartments in question have previously been merged:

```
/*If both geometric compartments have as of yet not been involved in any merging
  (which is evident from the fact that their respective elements in the dynamic vector (dynvec) have value 0)
  give cells in both compartments a number stored in UDM 10*/
if(dynvec[comp0] == 0 && dynvec[comp1] == 0)
{
    /*Cell c,t gets assigned macro comp number, mcomp, which is stored in UDM10*/
    C_UDMI(c,t,11) = mcomp;
    vol_comp2[mcomp] += C_VOLUME(c,t);
}
```

Secondly, the left compartment (relative to the cell face) has not yet been merged:

```
/*If comp1 was already merged with another geocomp,
but comp0 hasn't already been merged
  which means that cells c1,t1 already have an
  mcomp value attached to them, which is stored in UDM10
  therefore assign this same number to cells c0,t0*/
if(dynvec[comp0] == 0 && dynvec[comp1] != 0)
{
    /*Assign the same mcomp value of cell c1,t1 to cell c,t*/
    C_UDMI(c,t,11) = C_UDMI(c1,t1,11);
    vol_comp2[mcomp] += C_VOLUME(c,t);
}
```

And thirdly, the right compartment (relative to the cell face) has not yet been merged:

```
/*If comp0 was already merged with another geocomp,
but comp1 hasn't already been merged*/
if(dynvec[comp0] != 0 && dynvec[comp1] == 0)
{
    /*Assign the same mcomp value of cell c0,t0 to cell c1,t1*/
    C_UDMI(c,t,11) = C_UDMI(c0,t0,11);
    vol_comp2[mcomp] += C_VOLUME(c,t);
}
```

Lastly, the compartments are appointed value 1 in their respective tracking vector, to prevent their futile re-merging in future looping:

```
mcomp += 1;
dynvec[comp0] += 1; /*Marks this geometric compartment as having been involved with merging*/
dynvec[comp1] += 1; /*Marks this geometric compartment as having been involved with merging*/
```

Then situation 3:

In the situation that the velocity direction is the same, but the magnitude does not fall within the range set by the homogeneity factor, the compartments are assigned two different compartment numbers, since they will not be merged:

```
/*In this case microcomp velocities are not
close enough, but velocity
direction is the same*/
else
{
    mcompL = mcomp;
    mcompR = mcompL+1;
```

Then situation 2:

When the velocity direction is different, but the magnitude falls within homogeneity factor range:

```
/*If product of velocities is negative value,
that means velocity directions are not the same*/
else if(avg_vel_comp[comp0] * avg_vel_comp[comp1] <= 0)
{
    /*Subtract average velocities and compare absolute hereof with vel_diff*/
    if(abs(avg_vel_comp[comp0] - avg_vel_comp[comp1]) <= vel_diff)
    {/*In this part of the code we have pointed to two geocomps which
    both neighbor each other and have approximately the same average
    velocity magnitude but different direction*/
```

Finally, situation 4:

When neither the direction, nor the magnitude appear to match between the compartments:

```
/*In this case microcomp velocities are not close enough,
and velocity direction is not the same*/
else
{
    mcompL = mcomp;
    mcompR = mcompL+1;
```

1.6 Mixing time calculation

Validation of the functionalized and compartmentalized strategies is done by calculating the mixing time of the model. The mixing time is calculated using a Julia-based script provided by Dr. Haringa, C., which makes use of a set of ordinary differential equations to solve the mass balances. An initial guess is given and the equation is provided in the function “compdiff!”. Subsequently, the built-in function ODEProblem() is called to define the mathematical problem, which is consequently solved using the built-in function solve(). One of the benefits of using Julia for scientific data calculations, is the inherent capability of Julia to increase performance. In this code the @fastmath module is called, which uses algebraically equivalent math functions, but without respecting rules like associativity. Thereby, the speed with which calculations are performed is increased dramatically.

```
# first set the initial guess. in this case, mixing in compartment model of N instance
cinit = zeros(Float64, CompartmentProp.Amount)    # alloc initial also for biomass
cinit[compartment_data.FeedComp] = 1;
dC = zeros(Float64, CompartmentProp.Amount)    # alloc differential

# set integration options
tspan = (time_settings.StartTime, time_settings.EndTime)
dt = 0;

rlv = 1 ./ liq_vol  #pre-divide. [1/m3]
total_liq_flow .= total_liq_flow .* rlv  # dimless liquid flow (1/s)

par = (total_liq_flow)

function compdiff!(dN,N,p,t)
    tlf = p
        @fastmath mul!(dN,tlf,N)
end

# Compute the solution
prob = ODEProblem(compdiff!,cinit,tspan,par)
@fastmath @time sol = solve(prob,BS3(), reltol=1e-3, maxiters = 10e6, dtmax = 0.1, saveat = 0.1)
```

Appendix 2 - Functionalization

The functions in detail

Every function has as input certain arguments to allow for very specific usage and has three output variables. The inputs into the functions pertain to differentiating between single-phase and multi-phase usage and is achieved using the 'threadChoice' variable. 'threadChoice' can be chosen to be 0, 1 or 2, with each value representing respectively the dispersed ('mixed')-phase, liquid-phase and gas-phase. Changing the 'threadChoice' variable steers the Ansys compiler towards using the relevant overlaying subthread: Ansys appoints for each phase-level thread (the subthread) a specific which can be set by the user. The base script uses value 0 to point to the liquid-phase and value 1 for the gas-phase, but only if Ansys has actually allocated memory for multi-phase use. If no memory has been allocated for multi-phase, then the main thread is used instead of the multi-phase subthreads.

As mentioned before, the main in-program difference between inlet and outlet cells is the use of two other subthreads, identified with values 5, 10 or 20. Hereby, value 5 represents an outlet cell, while values 10 and 20 represent inlet cells. In order to be able to differentiate between inlet and outlet cells a variable called 'threadtype' was introduced, which can be set any of the above-mentioned three values. The 'threadtype' variable is the main difference between functions PhaseFluxIO(); and PhaseFluxIntC();.

As far as the output go, either the flux or the turbulent energy (depending on the function used) and the identification numbers of both neighboring geometric compartments are the principal outputs of the functions (figure 6). The identification numbers, comp0 and comp1, are used for distinguishing between which compartments the fluxes are calculated and are consequently necessary to fill in the flux in the correct element of the exchange matrix.

Upon calling the functions with the desired input arguments, the functions perform calculations using data from both the CFD as well as a previously run script, which calculates average values for the velocities, the turbulent kinetic energy and its dissipation rate as well as the gas fraction. The calculations performed are the following:

- **Inlet/outlet cells:**
  - **If averaging is enabled:**
    - $\psi = \bar{\varepsilon}_L * (\vec{v} \cdot \vec{\alpha}_f)$ (1.1)
  - **If instantaneous values are used:**
    - $\psi = \frac{\dot{m}}{\rho}$ (1.2)
- **Internal cells:**
  - **If averaging is enabled:**
    - $\psi = \frac{\sum \bar{\varepsilon}_L}{2} * \frac{(\sum \vec{v} \cdot \sum \vec{\alpha}_f)}{2}$ (2.1)
    - $\alpha_{phase} = \bar{\varepsilon}_L * |\vec{\alpha}_f|$ (2.2)

- $E_{k,turb} = (\sqrt{\overline{E_{k,turb,left\ cell}} * \frac{2}{3}} + \sqrt{\overline{E_{k,turb.right\ cell}} * \frac{2}{3}}) * \frac{\alpha_{phase}}{2}$ (2.3)

- **If instantaneous values are used:**
  - $\psi = \frac{\dot{m}}{\rho}$ (2.4)
  - $E_k = \sqrt{E_k * \frac{2}{3}} * |\vec{\alpha}_f| * \alpha_{fraction}$ (2.5)

When the use of time-dependent average values is desired, the flux, ψ, is calculated by multiplying the average liquid holdup by the dot product of the velocity and face area vectors (equation 1.1). Hereby the average liquid holdup originates from the averaging script, while the velocity vector and face area vector are extracted from the CFD using Ansys-specific macros. When considering internal cells, the average liquid holdup, the velocity vector and face area vector of both the neighbouring cells are summed and divided by two to get the average between the two cells (equation 2.1). Furthermore, the turbulent kinetic energy, $E_{k,turb}$, is calculated by first determining the face area by way of multiplication of the average liquid holdup and the face area vector magnitude (equation 2.2). Then the average turbulent kinetic energy of both cells is derived from the averaging script and used (equation 2.3).

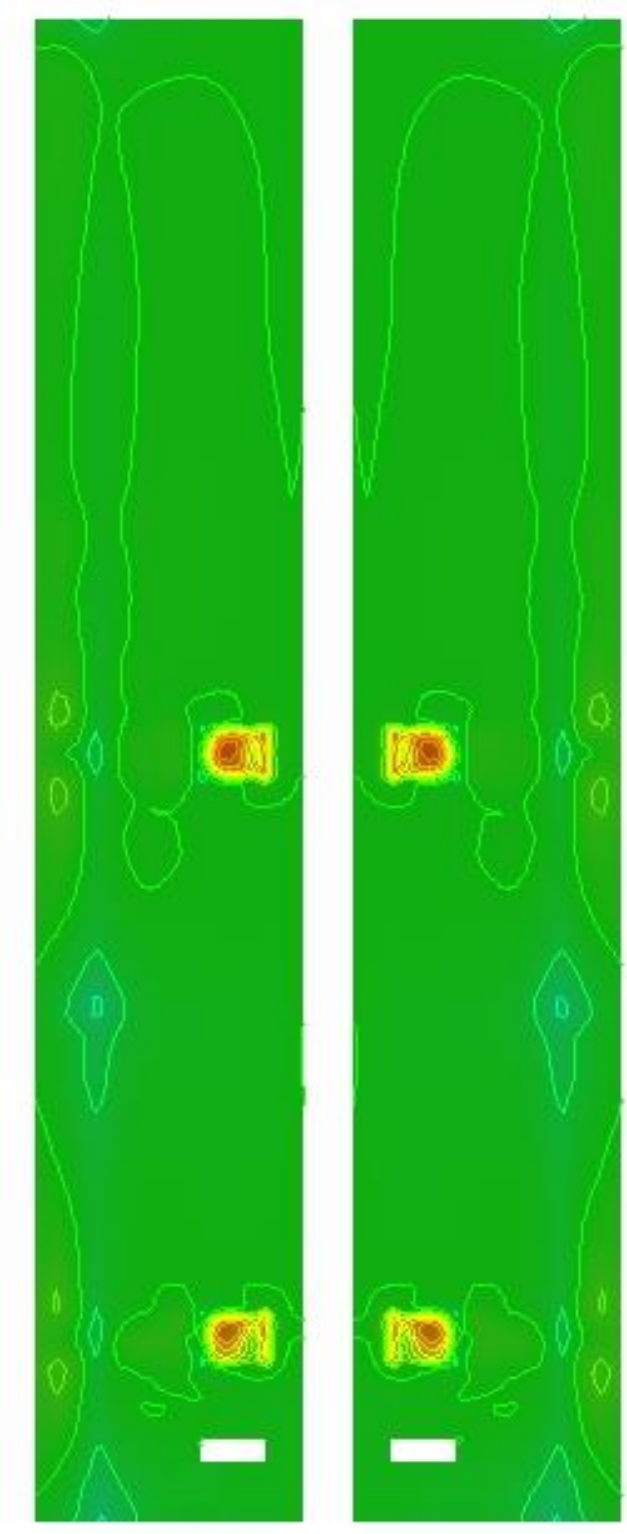
When the instantaneous values are used instead, Ansys-specific macros are used to derive the mass flow rate and local density from the CFD and used in the calculation of the flux (equations 1.2 & 2.4). Additionally, the turbulent kinetic energy is calculated by again using an Ansys-specific macro and the magnitude of the face area vector (equation 2.5). In the case of multiphase, the phase fraction of either liquid or gas-phase, $\alpha_{fraction}$, is used as an extra scalar.

As mentioned before, two functions were written to accommodate for the writing of data to external files, namely Compartment_writing(); and Volume_writing();. Both require as inputs the number of rows and columns, 'NC', in which the data is arranged, the folder location and filename to save to an external location and the data variable which has to be written. The main difference between the two writing functions is that Compartment_writing(); writes a matrix, whereas Volume_writing(); writes a vector. The benefits are that every user can define a unique location to save data to and the functions can be used irrespective of the data variable in question. The intercompartmental flux and turbulent kinetic energy data is written to external files using the Compartment_writing(); function, since the data is a matrix, whereas compartment volumes are exported using Volume_writing();.

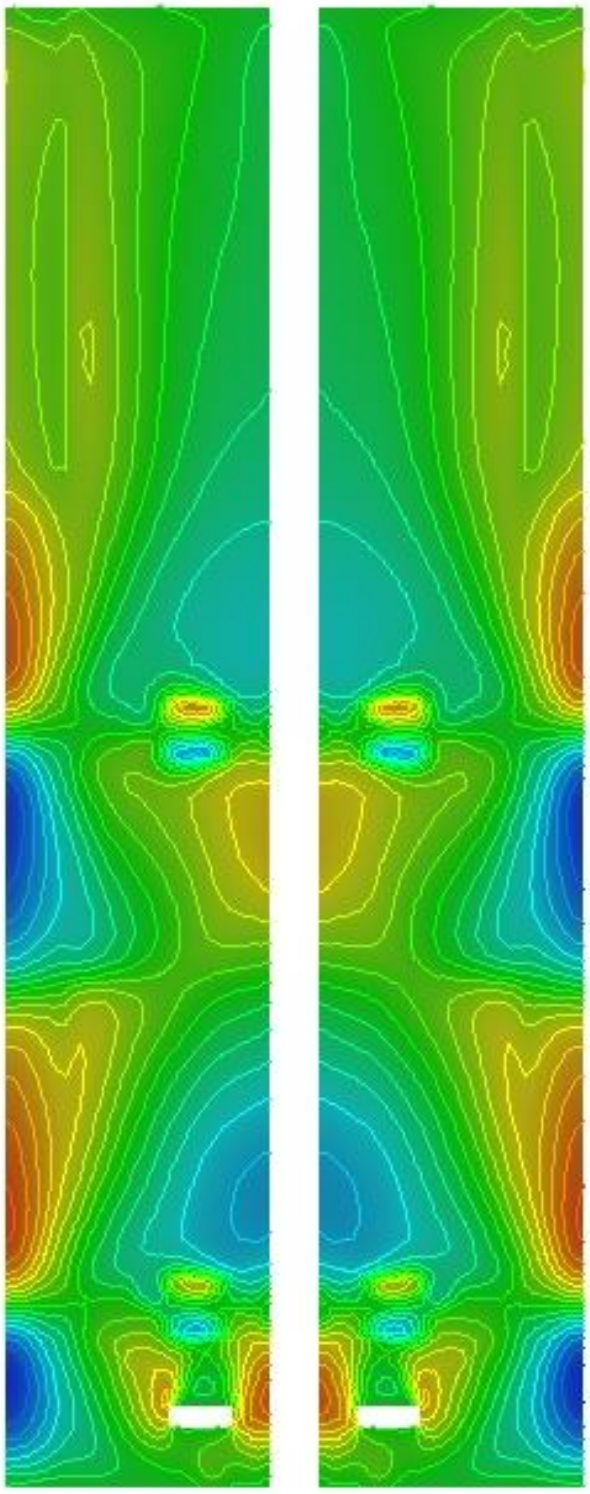
Appendix 3 – Enlarged figures

3.1 - Velocity field

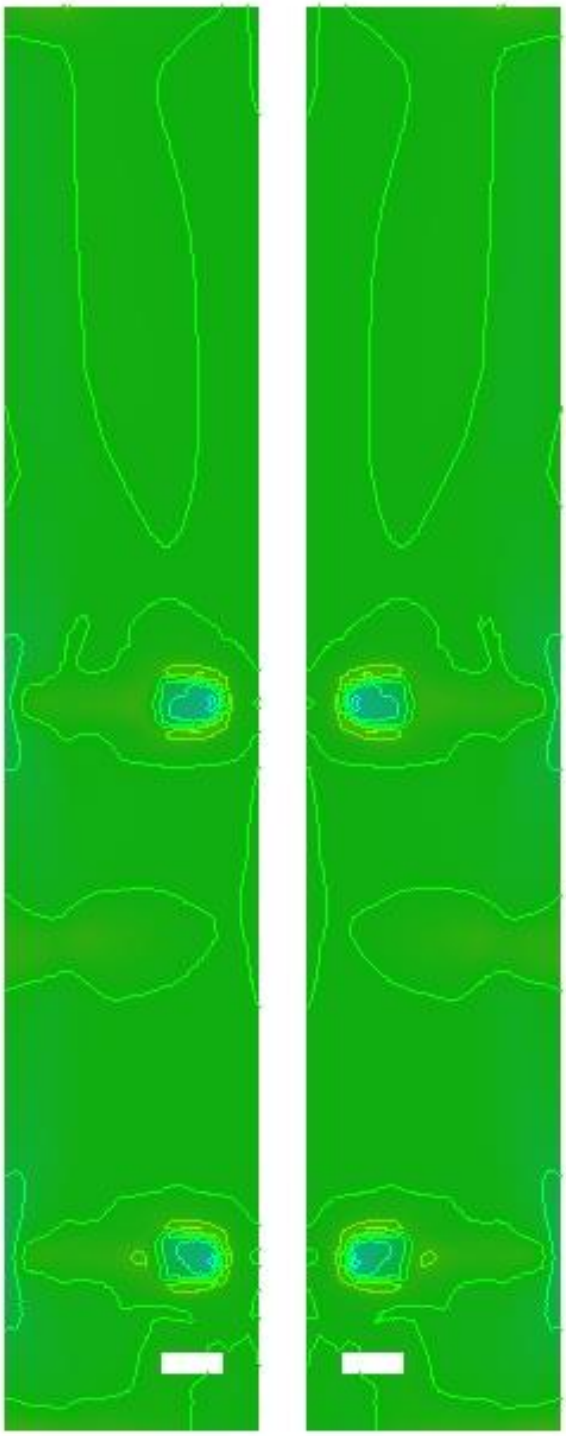
X-direction velocity field:



Y-direction velocity field:

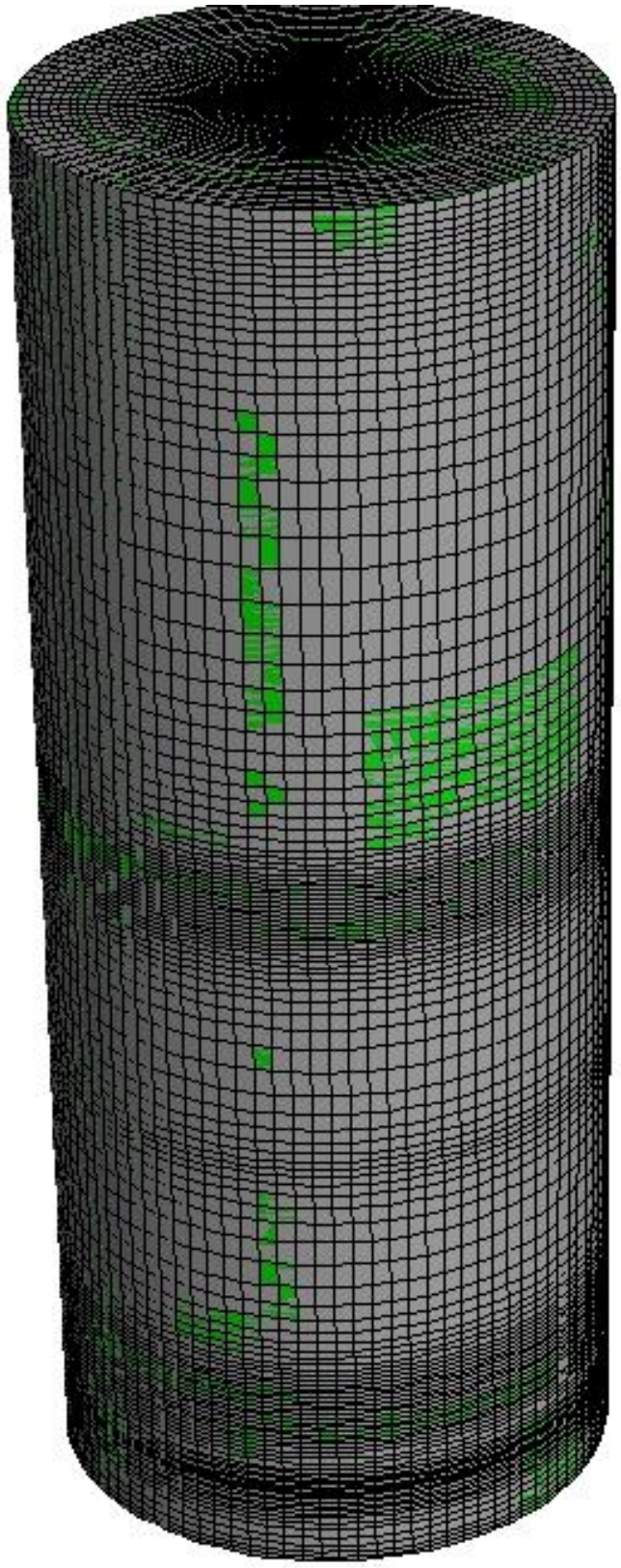


Z-direction velocity field:

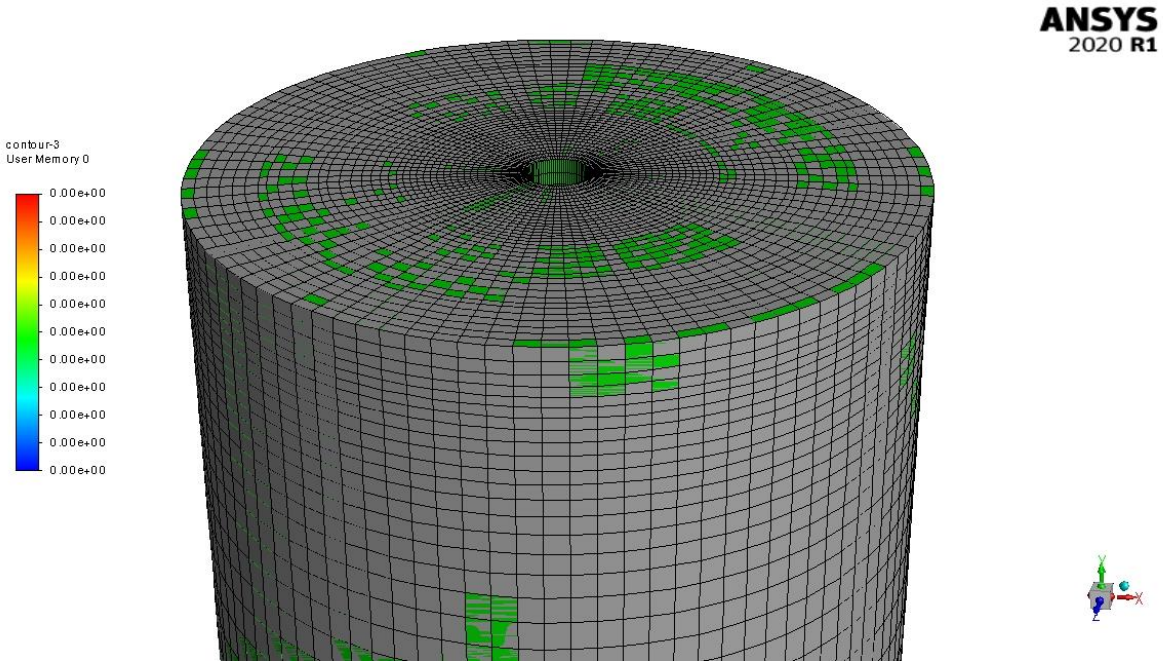


3.2 – Meshing

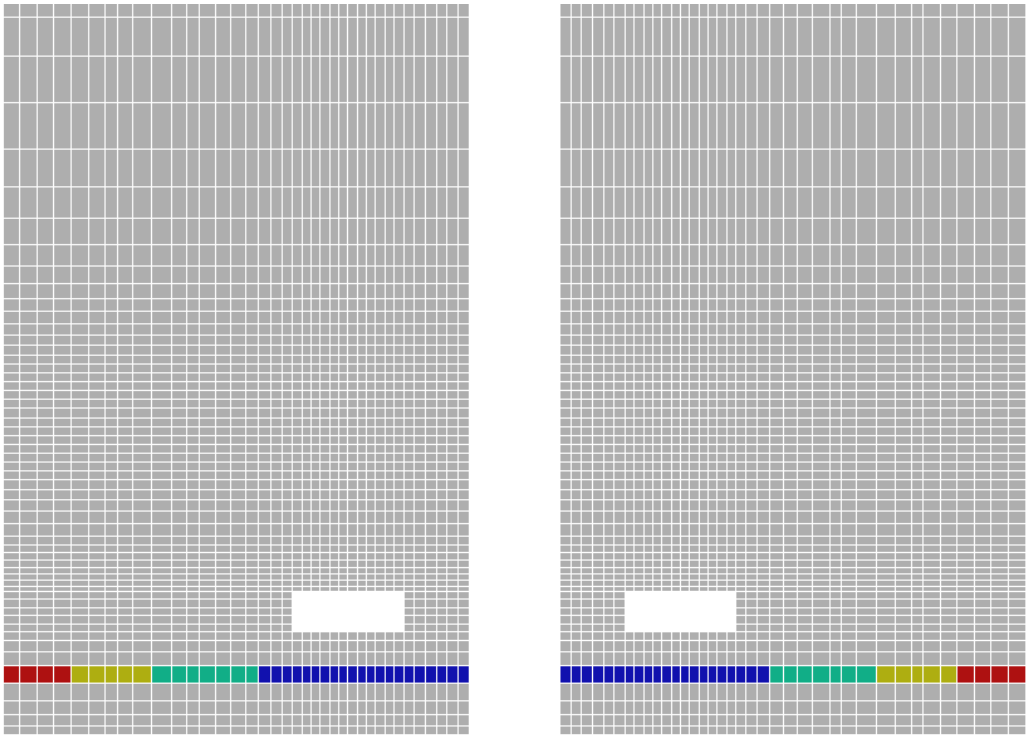
Mesh grid overlay of entire bioreactor:



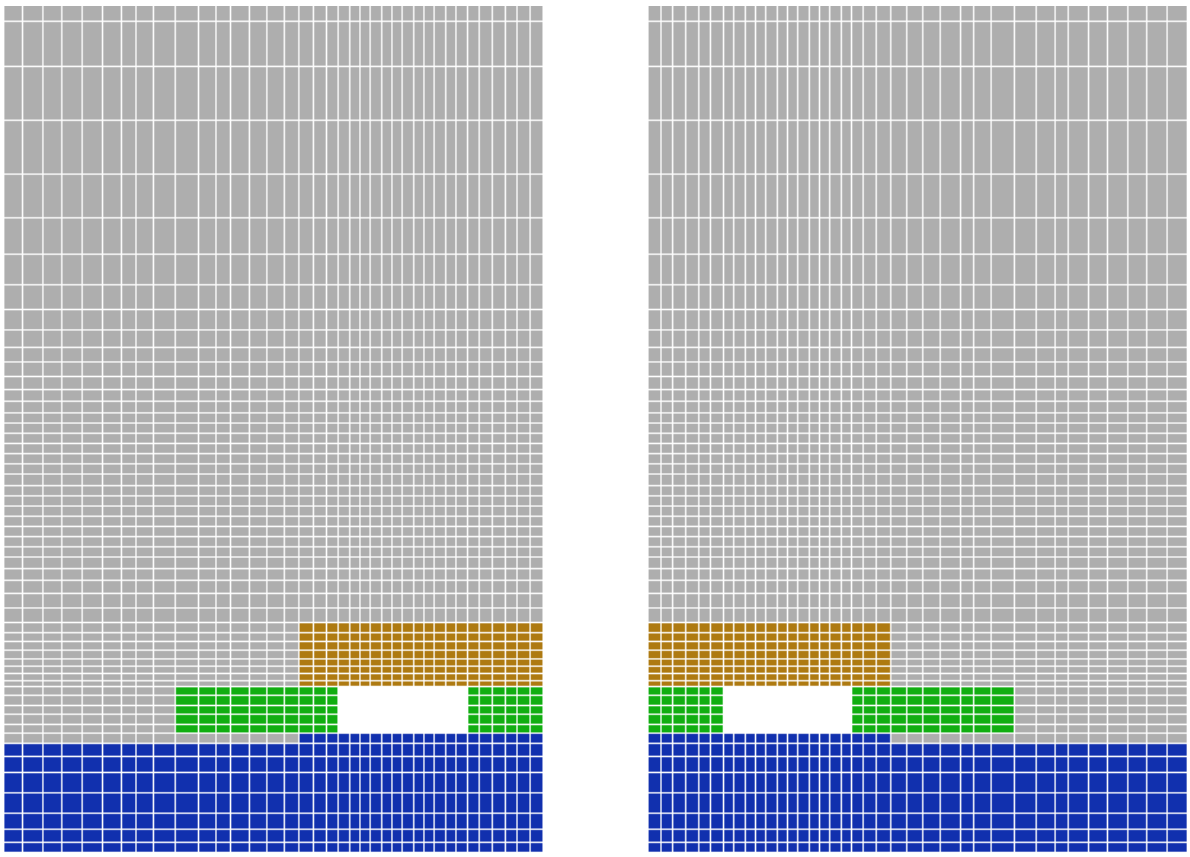
Zoomed in on top portion of mesh grid overlay:



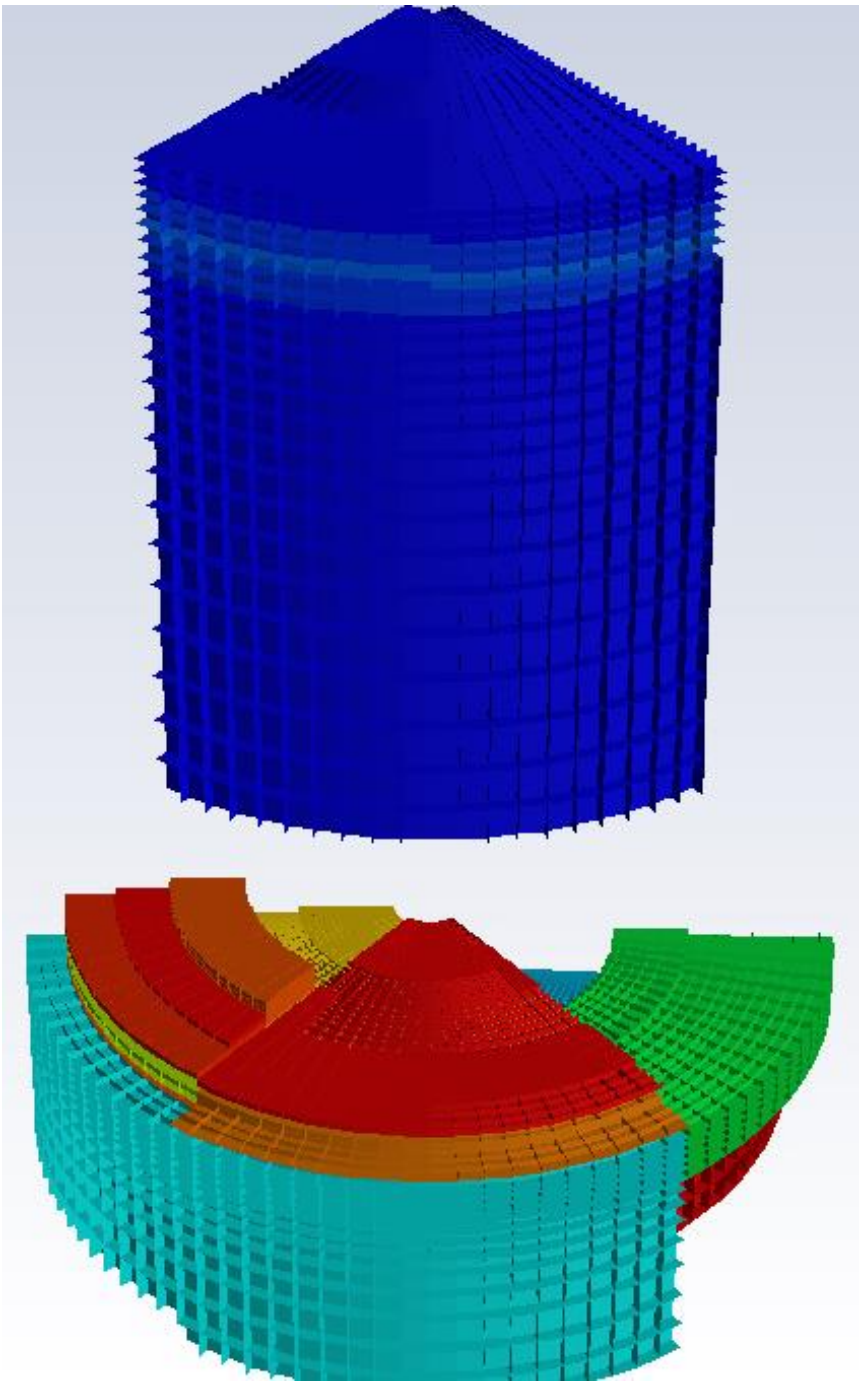
Zoomed in bottom portion of mesh grid overlay with four geometric compartments:



Zoomed in bottom portion of mesh grid overlay with three dynamic compartments:

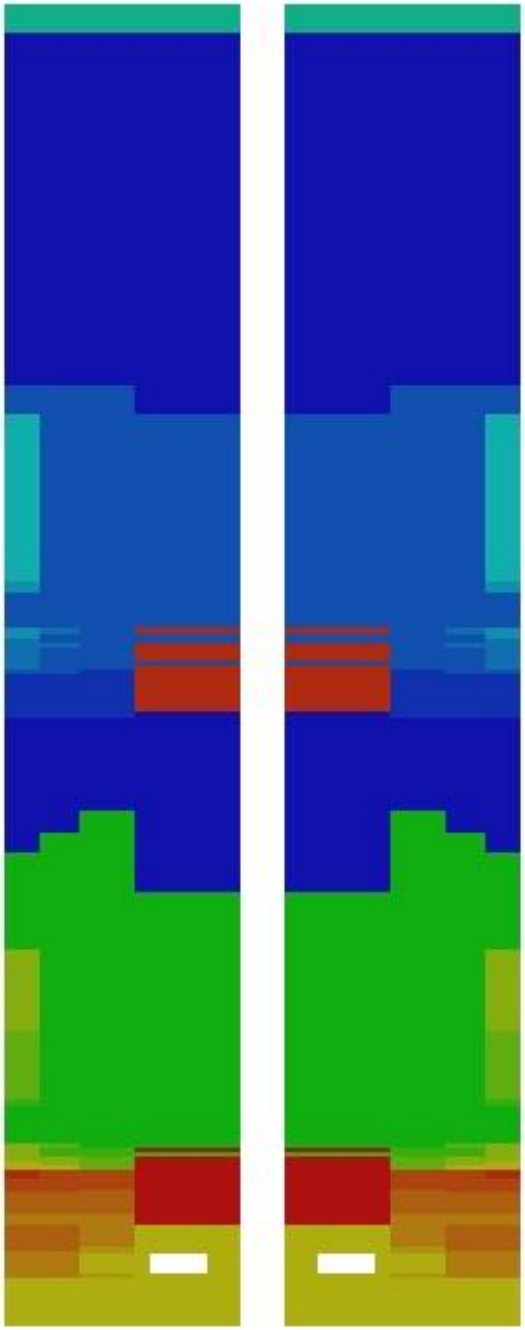


3.3 – 3D rendering of dynamic compartments

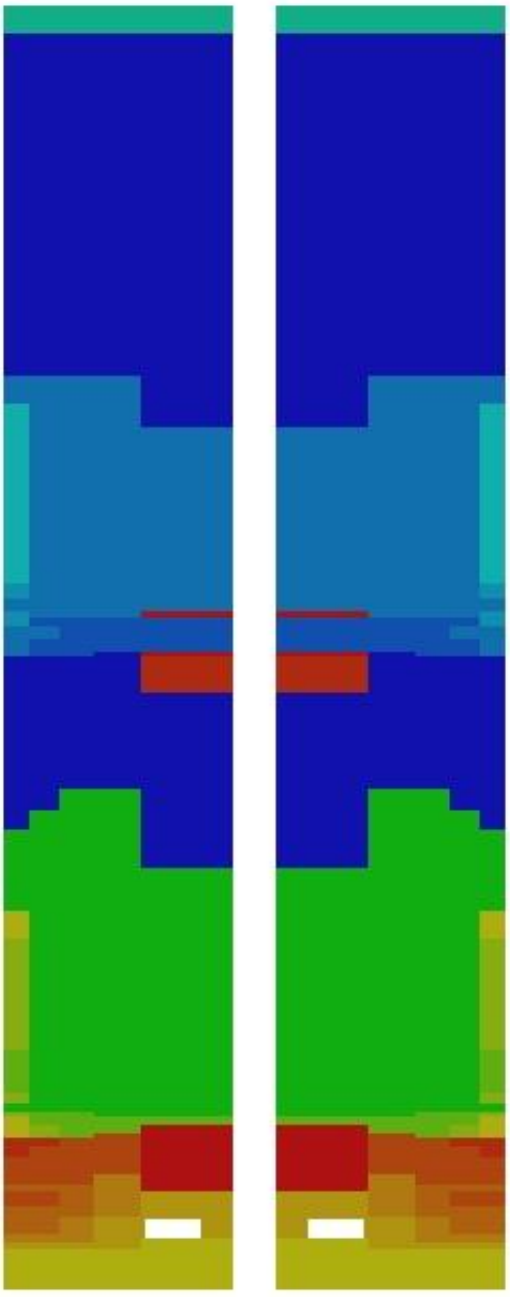


3.4 – Dynamic compartments

Dynamic compartments result from 5000 geometric compartments:



Dynamic compartments result from 10000 geometric compartments:



Dynamic compartments result from 50000 geometric compartments: