ÉCOLE NORMALE SUPÉRIEURE
UNIVERSITY OF PARIS-SACLAY
HIGH PERFORMANCE COMPUTING

End of Studies Internship

MASTER OF RESEARCH

Modelisation and Simulation with High Performance Computing

Subject :

Periodic boundary conditions for dual-phase flow interfaces in TRUST/TrioCFD

Presented by :
M. YEUMO BARKWENDE
William

Dr. KHIZAR Anida - **Supervisor**
Dr. BRUNETON Adrien - **Co-supervisor**
M. HAFID Fikri - **Reviewer**

University Year 2022-2023

Contents

1 Summary

Flow in vertical reactor pipes can have multiple regimes. Two-phase fluids regime is a paramount subject at the CEA to ensure that the steps taken to deal with nuclear installations are done within a safe environment. The Front-tracking method is a derivative of the poly-MAC (poly marker).

A Lagrangian method is very precise[5], but it suffers from being algorithmically intensive as the domain need to be re-meshed way more frequently. An Eulerian method will not be as numerically intensive as a Lagrangian method. However, the precision at the interface will be worse. Thus, a method that wish to use both methods' strength using a fixed eulerian mesh for the grid and a mobile Lagrangian mesh for the structures was implemented.

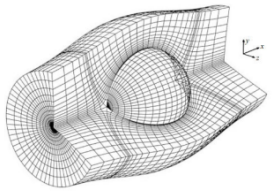

Figure 1 – Lagrangian adaptive mesh for mobile mesh (source: Legendre & Magnaudet 1998) [5]

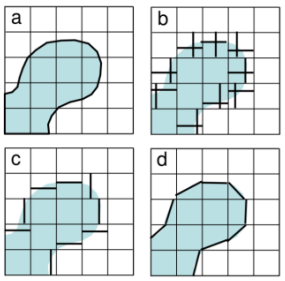

Figure 2 – Eulerian mesh for VOF methods and interfaces reconstruction (source: Tryggvason *et al.* 2011) [3]

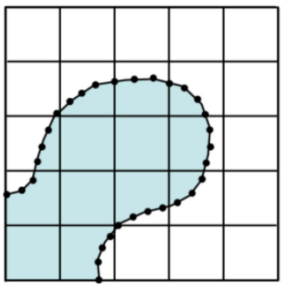

Figure 3 – Eulerian fixed mesh and Lagrangian mobile mesh (source: Tryggvason *et al.* 2011) [3]

Among a wide range of meshing methods for solving the two-phase Navier-Stokes equation, the Front-tracking method is used for TrioIJK and TrioCFD, for its accuracy and the reasonable number of markers it uses. In the current version of TrioIJK, periodic boundary conditions are implemented through an extension of the computational domain, which induces superfluous duplication of some data and an increase in the complexity of the code. The goal of this internship is to implement a new method that can remove this extension on TrioIJK and implement the periodic boundary condition for TrioCFD.

2 Acknowledgements

I wish to thank my colleagues in building 451, from the STMF and SGLS. Working among them was thrilling. It was always a pleasure to come to work. I also would like to particularly thank the colleagues in my office, Maxence and Manitas. And most of all, I would like to thank the full time employee in my laboratory : Adrien, Luc, Elie, Pierre, Maria-Giovanna and most of all, Anida. I owe this opportunity to work at the CEA to her and Adrien, who helped throughout my internship and allowed me to work independently.

3 Introduction

The periodic solutions for the Navier–Stokes equations have been extensively studied in many articles.

The open-source software TRUST, currently in development at CEA/DES (Paris-Saclay), allows us to resolve Navier-Stokes incompressible, weakly compressible, and compressible multi phase flows equations. It's used on bubble flow applications. Therefore, the bubbles are represented by a mobile surface mesh (the front) inside a fixed 3D grid (eulerian mesh). Thus, in order to simulate turbulent flows on a small representative domain, periodic boundary conditions are necessary on some edges of the grid.

Currently, the software only allows periodic boundary conditions for dual phase flow on a specific module (called BALTIK, Build an Application Linked to Trio_U Kernel) called TrioIJK. This module only deals with regular Cartesian meshes. The discretization of the mesh is structured which helps for algorithms since accessing the data of neighbouring nodes is a matter of increment in an array. In TRUST/TrioCFD, that is not the case, another data structure is needed to know your neighbour's index. This module uses an extended grid in which the front is duplicated on the other side of the domain's periodic boundaries. This method is complex and heavy in data storage, which limits its usability to small bubbles, and requires a lot of MPI communications. Therefore, a new algorithm which would not need this extended domain was implemented.

3.1 Geographic context

The CEA has nine sites throughout France. It is developing numerous partnerships with other research organisations, local authorities and universities.

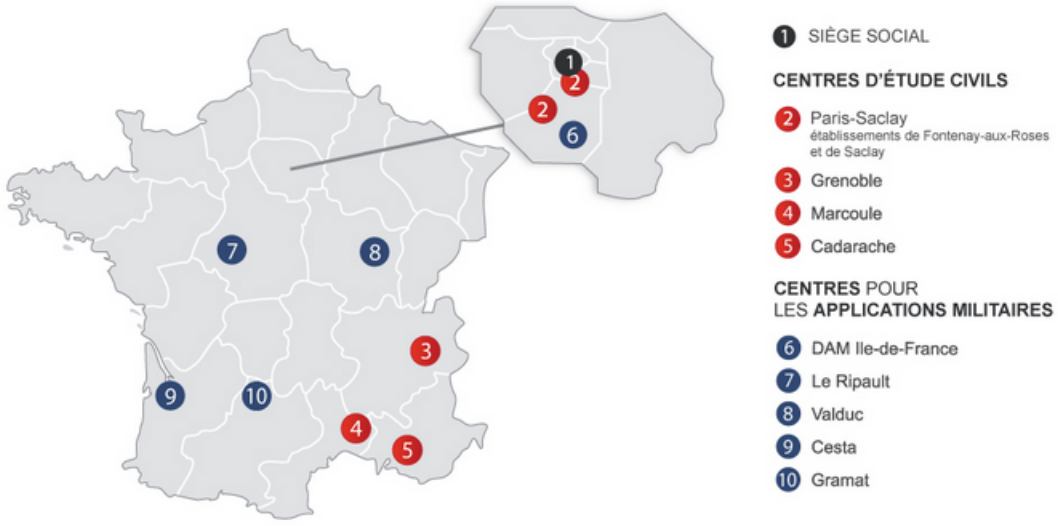

Figure 4 – Map of the different sites published in 2023 [1]

3.2 Organisation chart

The different research centers spread around France offer a wide array of activities. They are carefully managed around an organisation chart with four axis of development :

- Direction of Energies (DES).
- Direction of Military Applications (DAM).
- Direction of Technologies Research (DRT).
- Direction of Fundamentals Research (DRF).

The Direction of Energies, where my internship is taking place, works on civil nuclear power, low carbon energies, matters and universe, and many other fields. This direction is active in multiples places in France. However, at Saclay its implementation is more specifically called ISAS (Institute of Applied Sciences and low carbon energy Simulation),

In that institute, the department I work for is the DM2S (Department of Systems and Structures Modelisation). It has for goals developing tools to conceptualise, simulate and evaluate nuclear systems. To do so, it depends on software developed internally or in partnership with other departments or external entities. There are multiple units or services inside the department which also contain subsidiaries called laboratories :

- Studies of Thermal and Mechanics Service (SEMT).
- Studies of Reactors and Applied Mathematics Service (SERMA).
- Thermo-hydraulics and Fluid Mechanics Service (STMF).
- Software and System Engineering Service (SGLS)

The service I belong to is the SGLS, which was created this year on the 1st of January. It used to be a laboratory of the STMF, but as time went on and as their missions became more diversified, its scale had to be reconsidered and it became a service. In this service you can find three laboratories :

Figure 5 – SGLS's laboratories

The SGLS is working on primarily three software developed internally : SALOME, URANIE, and TRUST.

- The LESIM works on SALOME, which is an open-source software designed for pre and post processing for finite element problem such as mesh generation.
- The LIAD works on URANIE, which is used to uncertainty propagation problematic, with hyper parameters for instance, that can have severe consequence like changing the phase of a fluid by being couple of degrees higher.
- The LCAN works on TRUST, which has tools for thermal hydraulic problems. It has multiples extensions that can work in tandem or individually called BALTIKs. TrioCFD and TrioIJK are such extensions.

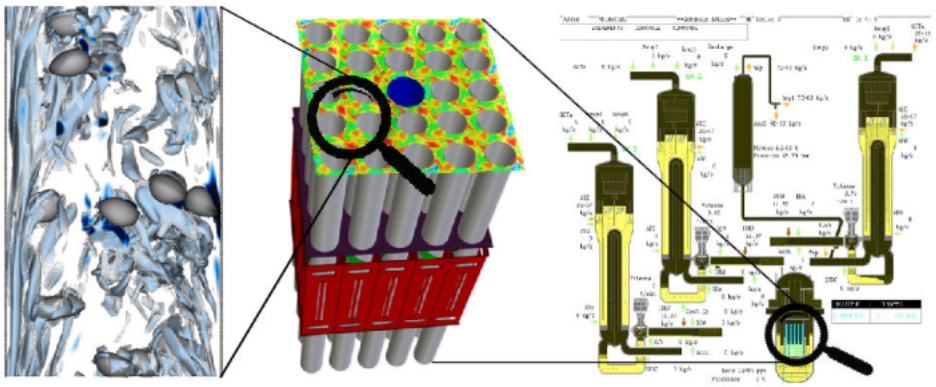

Figure 6 – Showcasing the cooling process with a simulation. From left to right : DNS approach (TrioCFD), RANS approach (Neptune_CFD), system approach (CATHARE).

While these software are important, each laboratory works on more than just those. They offer their expertise to other services internally or externally. In the LCAN, there are sub teams among the different employees. Each team works on differents projects related to high performance computing, TRUST or its extension and even software related to SERMA's workflow. Anida's team works on TRUST specifically and less on TrioCFD and TrioIJK. The latter is a Baltik which belongs to the STMF and more specifically the LSMF, Fluid Mechanic Simulation Laboratory. Anida and Adrien supervise this internship as the expertise in software development is there.

3.3 Scientific goals

As mentioned before, the main goal of the software TRUST and its extensions TrioCFD/TrioIJK, is to simulate small portion of different problematics in nuclear power plants. The main component in a nuclear power plant is the cooling system for regulating the core's temperature.

It's inconceivable to think about simulating the whole structure of the cooling system. The memory load would be massive. Thus, we simulate on a portion. In these power plants, the fluid used to cool is single phase and qualified as coolant. The nature of the fluid can vary from one power plant to another (water, sodium, etc.). This fluid will carry the heat generated by the core to the vapour generator.

However, this fluid cannot be considered as single phase in most cases, but as dual phase flow, for the following reasons :

- The nuclear reaction can lead to rare gazes such as xenon and neon.
- Oxidation of gears in the primary circuit leading to dihydrogen being generated.
- In the worst cases, and under specific conditions of both pressure and temperature, it is possible that the fluid changes phase, from liquid to gas.

Nonetheless, the dual phase flow is usually less performing than a single phase flow for heat exchange. The air pockets will have a worse heat transfer coefficient than the liquid. These air pockets' movement is dictated by Navier-Stokes equations :

$$\frac{\delta \rho u}{\delta t} + \nabla \cdot (\rho u \otimes u) = -\nabla P + \rho g + \nabla \cdot [\mu(\nabla u + \nabla^t u)] + n\sigma \sum_{i=1}^{N} k_i \delta^i \tag{1}$$

$$\nabla \cdot u = 0 \tag{2}$$

Variables	Meaning
ρ	Volumic mass
u	Velocity
P	Pressure
g	gravity
μ	Viscosity
σ	Surface Tension
K	Double of the average curve
n	Interface's normal towards the liquid
i	Index for a given bubble
δ	Dirac distribution
N	Number of bubbles in total

Adding periodic boundary conditions on TRUST's extensions, in particular TrioCFD, would allow extended simulation time without increasing the model's size. Thus, it would reduce the memory load on each simulation and allow some optimisations.

4 State of the art

In order to numerically create a nuclear reactor, the Commissioner for Atomic Energy and alternative energies (CEA), and more precisely the thermo-hydraulic and fluid mechanics service (STMF) as well as the software and system engineering service (SGLS), are heavily investing in developing a software that allow them to increase the safety of nuclear installations. As it stands, it's not possible to numerically solve Navier-Stokes equation at the scale of nuclear installations with our current hardware limitations. Thus, TrioCFD/TrioIJK's goal is to simulate phenomenons at a smaller scale in order to create predictive models for larger scale phenomenons.

During this internship, the goal was to review the new method for periodic boundary conditions for the front, optimise it, and check with all the current tests cases its stability and validity. Then, generalise this method for tetrahedral meshes for TrioCFD.

As of right now, TrioIJK has periodic boundary conditions implemented for dual phase fluids. This periodicity works on multiple layers : moving the front (markers on the interfaces of a structure, the concept will be explain later) and solving the equations. My focus will be on the former for both TrioCFD and TrioIJK but for different reasons. TrioCFD doesn't have periodic boundary condition for the front. And TrioIJK uses an "extended" domain which we want to remove. However, we do not want to impact any of the current equations implemented in both software.

4.1 Modelisation strategies

4.1.1 VOF

The VOF [4] method is defined by the volumic mass field which is called Indicatrice and noted $\mathcal{X}$. From a fixed eulerian mesh of the domain, we look at each cell of the mesh and compute the volume fraction of a fluid ($0 \leq \mathcal{X} \leq 1$). When $\mathcal{X} = 1$, the fluid we focus on (water or air) predominates this a specific cell of our mesh. And if in between of 0 and 1, the material properties of the two fluids are weight-averaged or our main fluid is missing if it's equal to 0.

Figure 7 – Indicatrice value in a dual phase flow with an eulerian representation [7]

Figure 8 – Indicatrice value in a dual phase flow with a mixed representation [7]

After each iteration it is necessary to update $\mathcal{X}$ by using the following transport equation :

$$\frac{\delta \mathcal{X}}{\delta t} + u \cdot \nabla \mathcal{X} = 0 \tag{3}$$

This equation impose an identical velocity to the interfaces in between the different fluids. This allows us to only need the fixed eulerian mesh to solve this problem which reduce both the space and time complexity of the implementation. However, it needs to be solved for each iteration. Furthermore, by construction of the problem, it's not possible to accurately resolve this issue with an eulerian mesh unless we increase its fineness. And, for the

post processing, reconstructing the interfaces between the different phases is necessary, which is complex as the information is lost in the previous iteration. Only the values in $\mathcal{X}$ are kept, and it can lead to loss of information as seen in Figure 7.

To solve this issue, other method can be used such as the mixed approach that has a fixed Eulerian mesh to describe the domain but a Lagrangian mesh to describe the object of study like in Figure 8.

4.1.2 Front-Tracking

Simulating incompressible fluids is complex. In 1965, Francis H. Harlow, J. Eddie Welch, and the T-3 team at the Los Alamos National Laboratory developed the Marker-And-Cell (MAC) method[6]. These markers are used to locate the fluid's free surface.

At each time step the fluid is modeled as a velocity field. Then, the velocity between these markers is calculated via interpolation. The new position of each marker is deduced with the interpolated velocity. This allows us to bypass the transport equation necessary in the VOF method.

However, refining the mesh will require a great number of markers. These markers are present in the whole fluid and not just the surface. There is also a risk of an imbalance in the markers distribution which requires a redistribution, increasing the execution time.

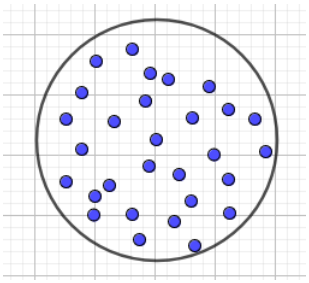

Figure 9 – Mark and Cell method

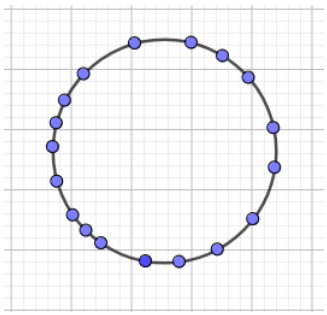

Figure 10 – Front Tracking method

The front tracking method was first introduced in 2001 by Gretar Tryggvason to simulate multiphase flows. Since then, the method was polished and the algorithms around it evolved. This method needs less marker compared to the MAC method. The process is similar to the MAC method but only uses the interfaces separating the two fluids.

4.2 Implementation on TrioIJK

The code base TrioIJK was made to solve dual phase flow mechanic problems. The main difference between TrioCFD and TrioIJK is that for the latter the domain is on a Cartesian grid instead of an Eulerian tetrahedral or hexahedral mesh. However, the interfaces are still following a Lagrangian mesh. This means that most of the tools developed in TRUST[2] or even TrioCFD[2] are available to use in TrioIJK.

The Cartesian grid is defined the .data file that controls the experiment. The user sets the length of domain in each direction and the number of nodes in each of these direction. For instance, the x axis can be 1cm in length but coarsely meshed with 10 nodes, making the distance between each nodes 1mm.

In TrioIJK, the main method is a function called `run`. The way TRUST works with its extensions is that the developer will overwrite a key function from TRUST called `Interpreter`. This function will read the .data file that has all the hyper parameters the user wants to use for the simulation. And the end of this method, `run` is called.

4.2.1 Old Implementation

We will only focus on the the version for the explicit Euler time scheme and always in a dual phase flow problem. The following algorithm is a simplified version of what is implemented but highlights the main function that will be impacted in the new implementation.

Algorithm 1 run

```
Initialise values of the different variables
for i in number_of_timesteps do
    Compute the number of real and ghost bubbles
    Compute the volume of each bubbles
    Displace interfaces
    Update the arrays of nodes, facets, elements by going through the mesh
    Update the velocity {Which is done by solving equations}
    Compute the acceleration source
    Update Indicatrice
    Update Thermal Properties
    Compute Indicatrice for the next time step
end for
```

First, the initialisation of our problem. There is a variable in TrioIJK that controls an object called the extended domain : `ijk_splitting_ft_extension`. Its value depends on what the user needs to ensure the periodicity boundary conditions limits, as it influences the size of the extended domain.

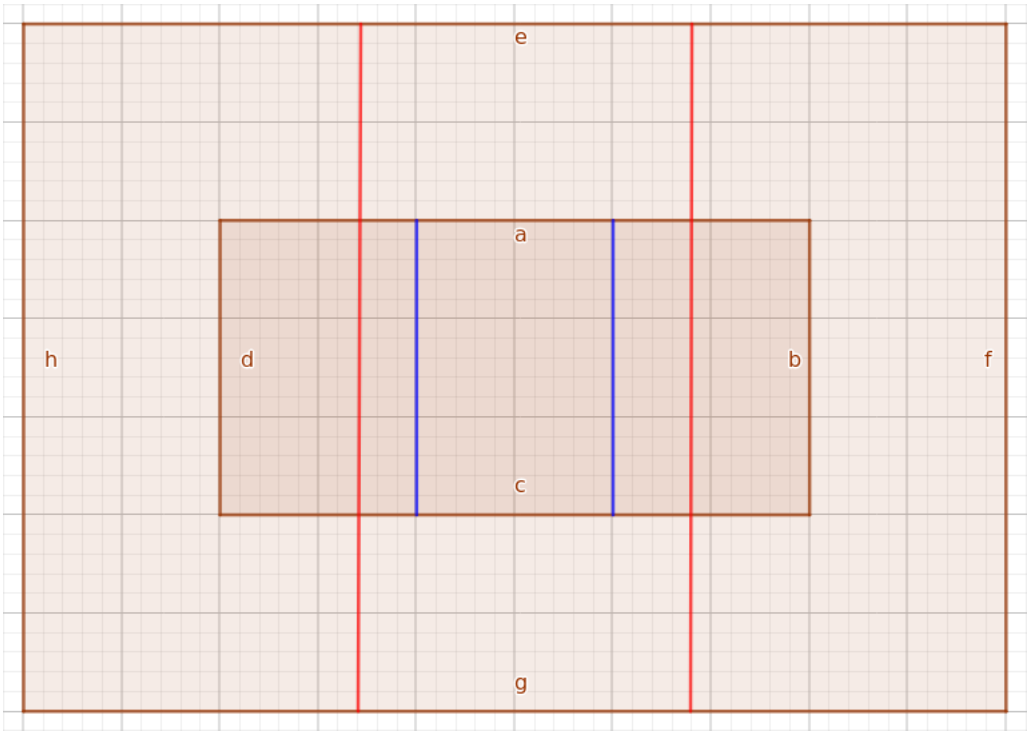

Figure 11 – Domain splitting in TrioIJK

With the figure above, on the x and y axis periodic boundary conditions have been implemented (segments a, b, c and d are periodic) and 3 processors are being used for the simulation. The inner box represents the "real" domain, the one which size has been defined in the .data file and where our "real" bubble will appear. This box

has been divided in 3 sections of the same size (blue lines), each one belonging to a specific processor respectively. The outer box on the other end, represents the extended domain. This domain is also split among each processor (red lines). The border in between a red and blue line is problematic for us.

Since both domains don't have the same size, but are still split on the same number of processors, some of them have a portion of the physical (inner) domain that is not on the extended domain of the same processor. This information belongs to the extended domain of the neighbouring processor, thus we need to update the information of the extended domain through MPI communications with the redistribute class.

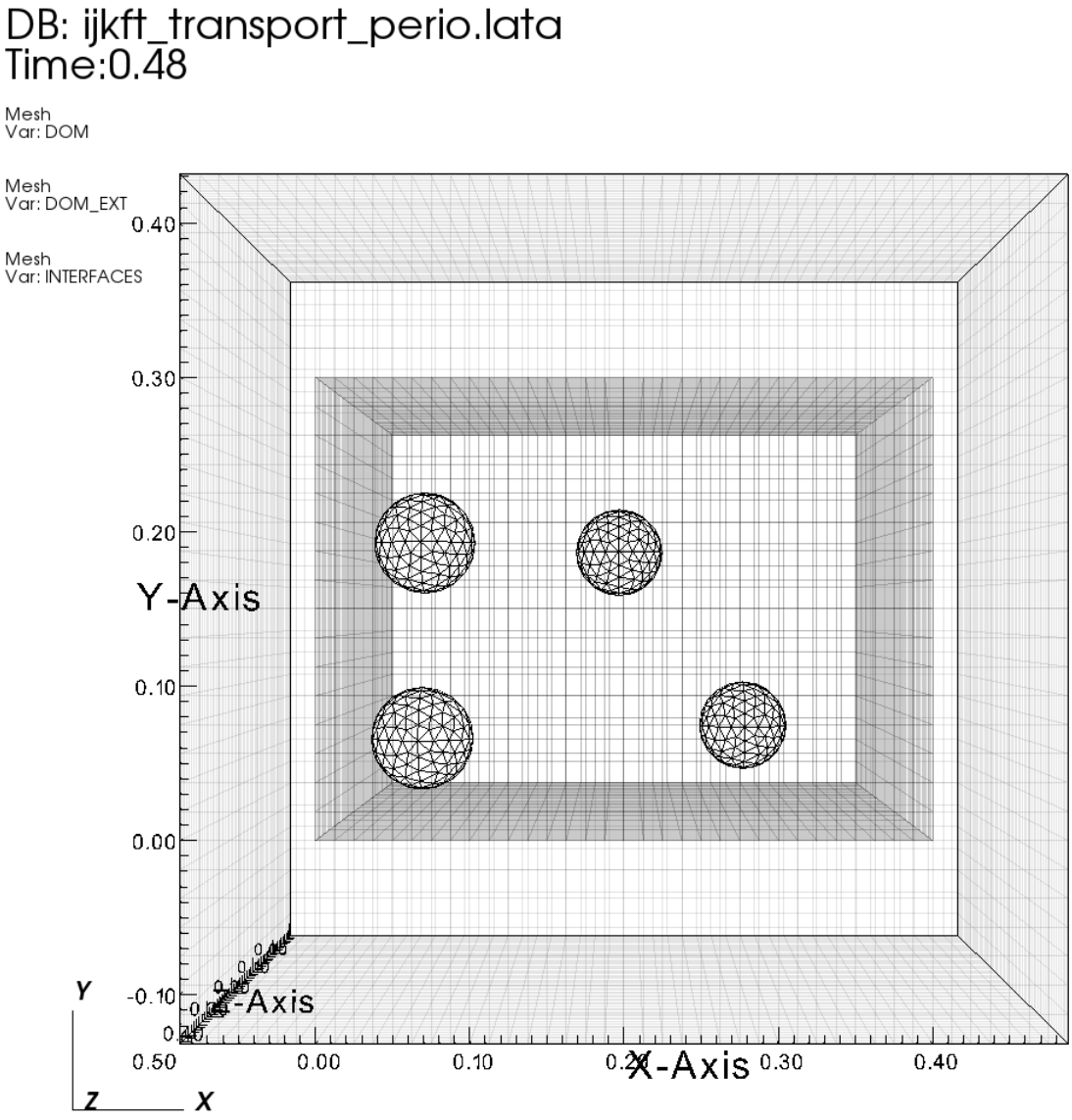

Figure 12 – 3D Representation of the real and extended domain with periodicity on the X and Y axis and 4 real bubbles.

This example uses one of the default test cases available to both the users and developers. We have 4 air bubbles in a fluid, moving diagonally upward. The inner and outer box can be differentiated.

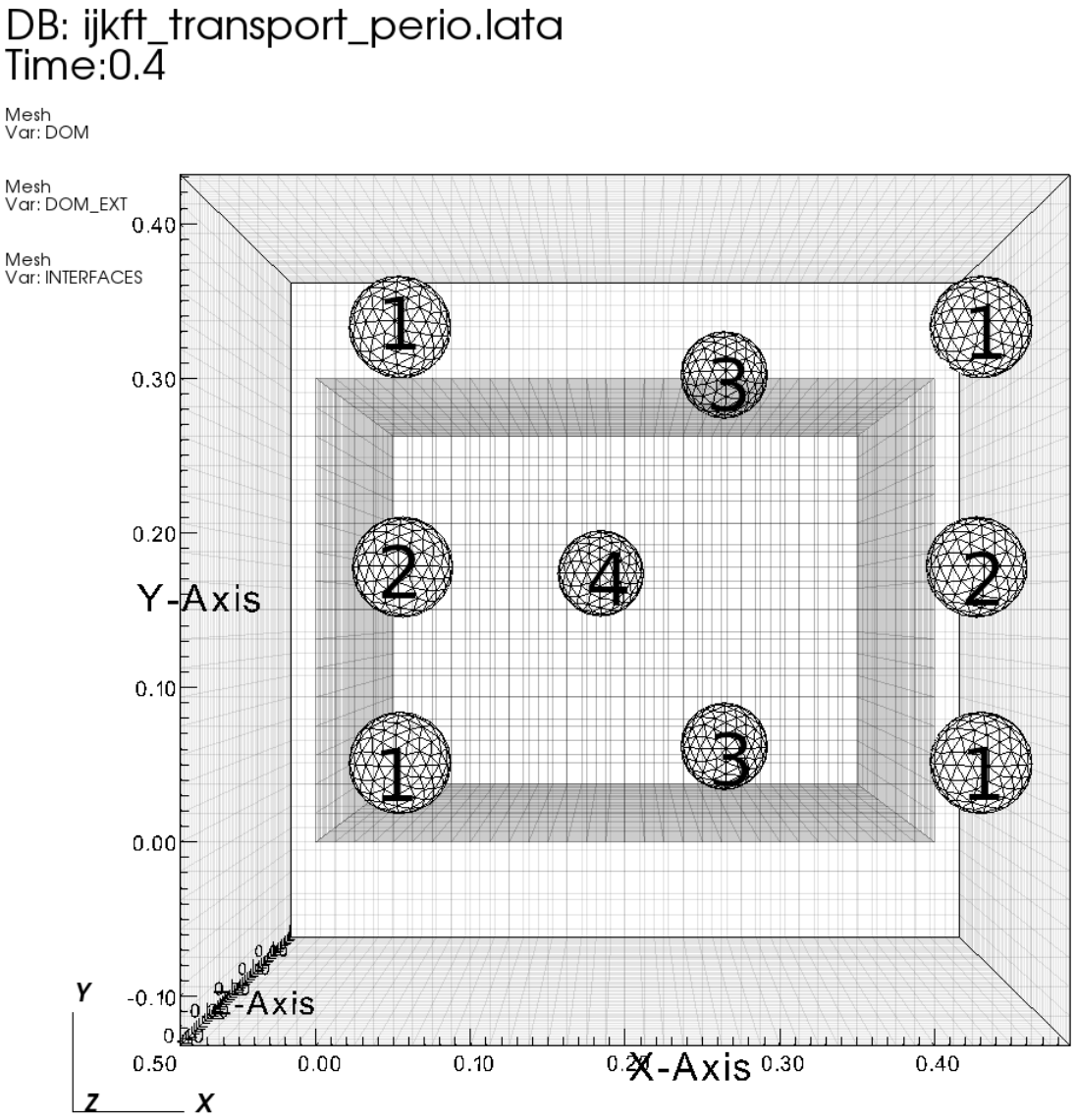

Figure 13 – 3D Representation of the real and extended domain with periodicity on the X and Y axis and 4 real bubbles and some ghost bubbles with an index to link them.

This image was taken a few frames earlier, which can be deduced from the time being lower on this screenshot. This displays well how visually the extended domain works. To help with the comprehension of what is happening I indexed each "real" bubbles with a specific number and all "ghost" bubbles with the index of their corresponding "real" bubble.

The concept is simple, the program redistributes the velocity in the inner domain on the extended domain by copying the velocity close to the opposite border and inside the inner domain. This leads to the velocity being known by the entire extended domain. Move the mesh and all its components with the speed obtained previously. When a bubble is close to a periodic boundary, it will duplicate itself on the other side of that boundary. In this image, the real bubble 2 on the left used to be a ghost bubble, while the ghost bubble 2 used to be a real bubble. A bubble is considered real until it completely leaves the inner domain, which would mean that the ghost bubble is entirely inside. However, a more complex case would be for the real bubble 1. The previous real bubble one in the one on the top right corner. While it was leaving the inner domain, it was crossing multiple boundary conditions at the same time, which led to multiple duplicate ghost bubbles : one at the top left corner, bottom right corner and

bottom left corner. Therefore, two out of three ghost bubbles will always stay as such since they only take either one of the boundary condition into account while the other one uses both.

Such an implementation is complex and slow. Velocity is not the only variable being redistributed to the entire extended domain, the restoring force, repulsive force, barycenters and much more. Therefore, this implementation is costly in execution in time and space and more MPI communication. Thus, a new method was thought of back in 2021 to improve the periodicity.

4.2.2 New Implementation

There are a lot of information that needs to be discussed and managed in the new implementation. In order for the new algorithm to work, it needs to be possible to compute the volume and barycenter of each bubble at all time. Furthermore, each Lagrangian facet needs to be positioned correctly in the eulerian element she physically intersects, to make the Indicatrice possible to compute.

While the main algorithm is mostly the same new methods have been introduced while some have been rendered obsolete by design. Everything related to ghost bubbles, duplicating bubbles and destroying those duplicates is now useless. Redistributing the information from the inner domain to the extended domain is also no longer necessary (which at the time of writing this report, has not been yet removed from the main program).

To have a better understanding of what we need and the program will evolve, it is important to keep in mind that each bubble uses a Lagrangian mesh with a front tracking method. The vertices (nodes or markers) are stored in a data structure called `sommets_`, and the facets which describe the connectivity of the bubble are stored in a data structure called `facettes_` However, new methods and structures need to be made :

- First, revisit the method for moving the vertices through the periodic boundary since we need to consider the extended domain as entirely gone. This is done by automatically moving the vertex on the other side of the inner domain instead of the periodic boundary like it does with extended domain.

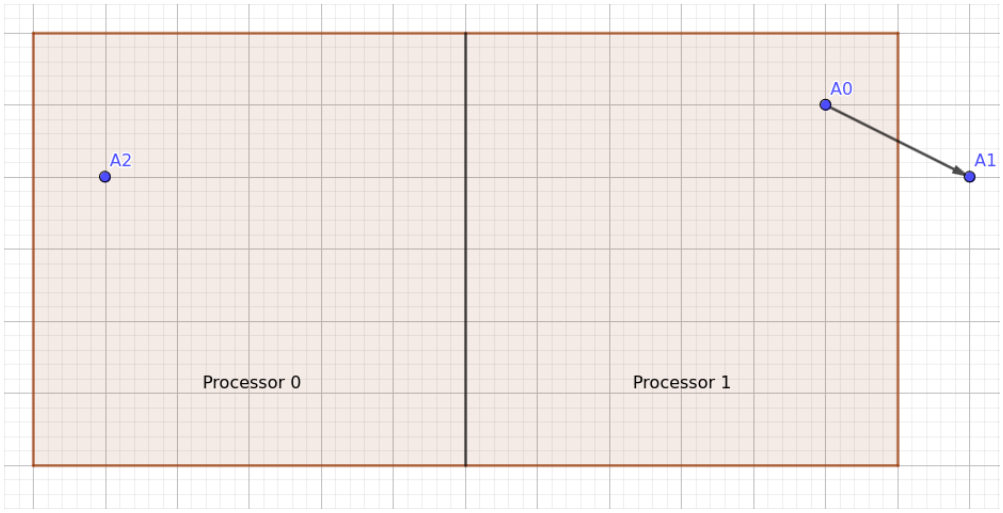

Figure 14 – Displacement of a vertex.

  A1 is the position of vertex A0 after a movement with the extended domain and A2 is its new position in the new implementation.

- Then, we need to maintain the connectivity of the bubbles. This is paramount for barycenter computation, volume computation and much more. New data structure will be needed for this.

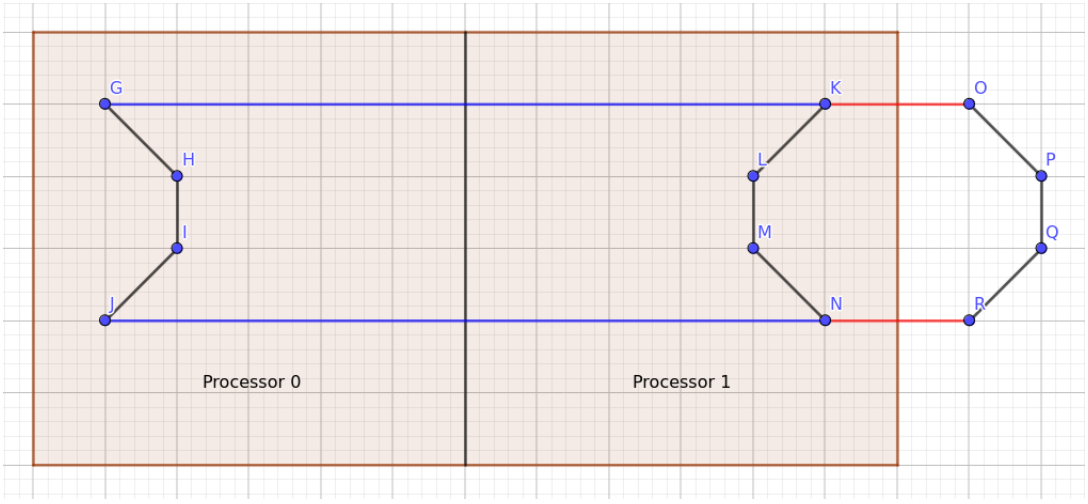

Figure 15 – Connectivity of a bubble with the new displacement method.

In the previous implementation, the connectivity would be ensured by the red lines, but in our case, the bubble is linked by the blue lines with our new movement method. This type of configuration is problematic when computing the barycenter, the volume of each bubble or even the normal to the facets. Thus, a new method and data structure was introduced : `Maillage_FT_IJK::compute_sommets_etendus()` and `sommets_etendus_`.

Algorithm 2 compute_sommets_etendus

Ensure: sommets_etendus_
  On the current processor, compute the barycenter of each bubbles in its domain.
  Exchange this information with all processors.
  Initialise sommets_etendus_.
  for `i in number_of_bubbles_on_processor` **do**
    for `direction in` $\mathbb{R}^3$ **do**
      Search on other processors if the bubble has a barycenter distant from mine, such as my barycenter's coordinates superior to the other for this directions
      if `Such a processor exist` **then**
        Subtract on my processor the length of inner domain for the current direction to obtain the coordinates the extended vertex.
      end if
    end for
  end for

This way of reestablishing the connectivity will create vertices that have coordinates outside of the inner domain.

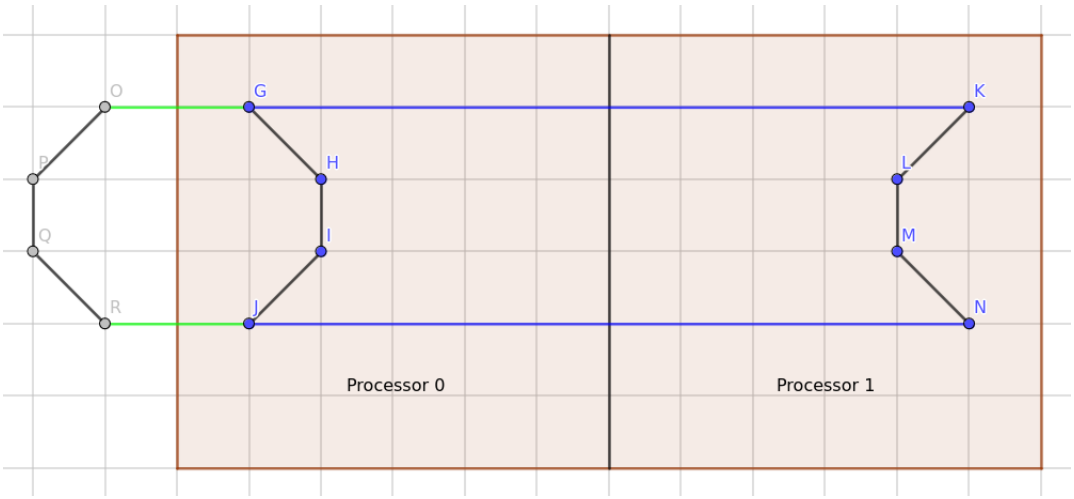

Figure 16 – Bubble with the extended vertices in grey.

The barycenter of the bubble has a greater value on the processor 1 along the x axis, thus the extended vertices will have their x coordinate subtracted by the length of the domain on that axis. This gives us the grey vertices on the left. The vertices that form the bubbles on the left are stored in `sommets_etendus_`.

- At this stage, the movement and the connectivity is being managed by the default structures `sommets_`, `facettes_`, and `sommets_etendus_`. Now, each Lagrangian facet need to intersect the eulerian mesh in a way to compute the Indicatrice with the method presented above. To do so, our current structures are not enough, as `sommets_` can create facets that will cross the entirety of the eulerian domain (blue lines in the figures above) and some facets formed by the extended vertices do not cross any eulerian element (such as [OP], [PQ] or [QR] in the figure above).

  So, a new data structure is needed to create the facets that can cross multiple eulerian element due to the periodic nature of the boundaries. This implies that some facets may need to be duplicated and up to eight times in 3D in some cases.

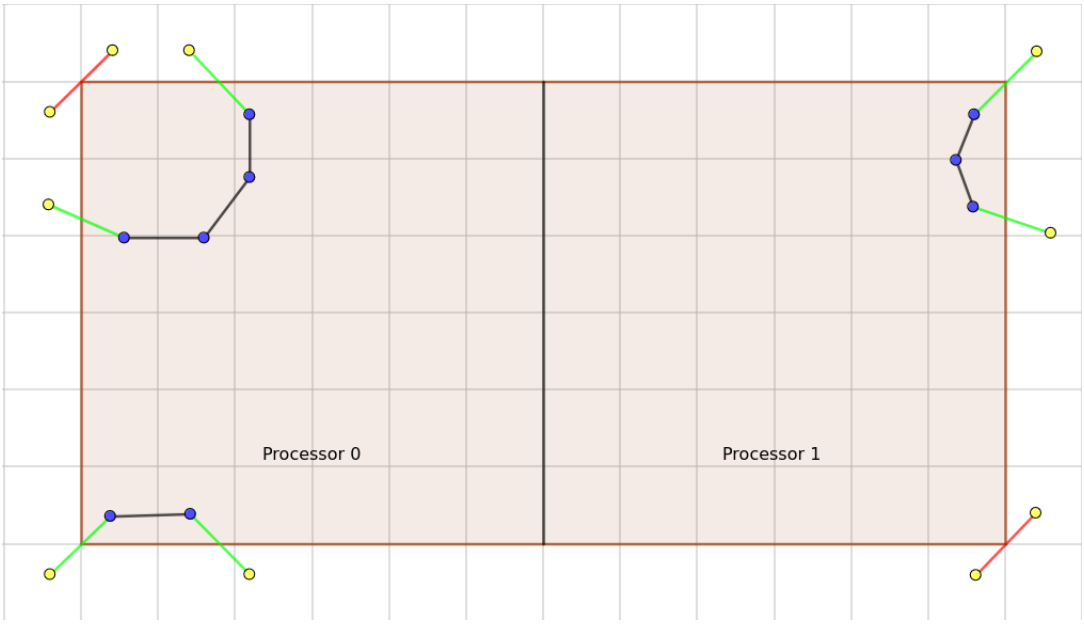

Figure 17 – Example in 2D of the facets we need and may want.

In the example above, the black segments represent the facets that have both of their vertices in the inner domain and would have not been problematic to deal with. The green segments have one of their vertices outside of the inner domain but are clearly crossing an eulerian element, so the facet and the vertex need to be included in our new data structures. And the red segments, have both of their vertices outside of the domain and in this case it's hard to say whether or not they cross an eulerian mesh without zooming but we'll consider they do. All the vertices and facets represented will be stored in `sommets_temp_` and `facettes_temp_`. Furthermore, the facets do not link vertices on the opposite side of the domain as it did with `sommets_`. To get this result, we first need to look for the facets that have to be sent to a specific processor and send them to it. Then, they are generated thanks to the coordinates available in `sommets_etendus_`. And to duplicate them, if necessary, we translate the facets wherever necessary.

Taking everything into account, moving the Lagrangian mesh has been thoroughly modified from what it originally was. Before, the program would just duplicate the bubble symmetrically and create ghost bubbles that would slowly become real bubbles as the real one left the inner domain. And now, we need to move each vertex carefully, create extended vertex to keep the connectivity and temporary ones to compute the Indicatrice for instance and really showcase the periodicity.

4.3 Implementation on TrioCFD

As of writing this report, only a few methods have been successfully implemented into TrioCFD. TrioIJK has an eulerian mesh but it has Cartesian coordinates. Meanwhile, TrioCFD uses a tetrahedral mesh for the inner domain. It's something I'm working on and plan to finish before the end of my internship in October.

5 Results

This whole project was first started in 2021 by a previous intern whose work was used to understand the whole process and idea on how the new implementation is supposed to work. However, the software and its extension changed extensively since then and even during my internship, thus I had to update a lot of methods that weren't there back in 2021 and do a lot of debugging before anything could run properly. By doing so, I encountered a lot of bugs along the way, some will be described in more details as they've been persistent on multiple occasions and some are more specific or just hard to understand.

Therefore, I tried to standardise the variables' name and methods' description as much as possible for the files I modified. A lot of work put into making the code clear and understandable so that someone with little to no understand of what a method does, can understand what's happening. This made it easier for myself when I had to modify multiple files at time and keep track of all variables called and their meaning. While it's still a work in progress since the files I worked on were sometimes have close to ten thousands lines, not everything was updated to an easy to understand state yet.

Furthermore, some methods were implemented twice as they either used `sommets_` or `sommets_temp_` for their calculation. It wasn't really explained inside the code why either one was needed or not, and some of those methods were also never called by the main program. So, the refactoring process allowed me to reduce a lot of bloat present in the code and optimise the data structures to be called only when needed.

However, there is detail that is important to note, the way TRUST and its extension was built for the major part of my internship only utilised MPI which is great for performance. But, multi threading was mostly overlooked and did not even compile properly. This was later addressed in more ways than one when they introduced the ability to use GPUs for some solvers. My testing was mostly focused on the MPI part since it was the most problematic due to some processors lacking the needed data to continue or operate properly. The computation part needed less work.

5.1 Performance

The extended domain in the old implementation is a real issue. In TrioIJK, you can specify the size of the inner domain through parameters for each direction for a 3D space. Furthermore, you can also set the number of elements, which is the level of discretization you want. And, another parameter the user can set is the size of the extension for the extended domain. These three values are important because a lot of the data consumed by the software will be on the mesh unless it's extremely coarsely mesh. It's not uncommon for domains to have tens of millions of nodes. If the initial domain has for size 10x10x10 and I want 1000 elements in each direction, that's a billion nodes. Any extension of the domain will add a considerable amount of nodes, and its size will depend on the size of the bubbles in the domain so that they can fully leave the inner domain as the ghost bubbles become a real bubble.

For the following tests, I used the provided test case that can be found in TrioCFD's repository for TrioIJK named `ijkft_transport_perio`, but some few parameters will be modified.

- The domain size will be the same, with 0.4 x 0.3 x 0.5 but the number of elements for each axis will be 64 uniformly.
- dt_post = 1 and nb_pas_dt_max = 60, to make the test case last longer and have a rather smooth and long animation.
- For the bubble, a radius of 0.035 and the we will be using the following mesh generated by gmsh :

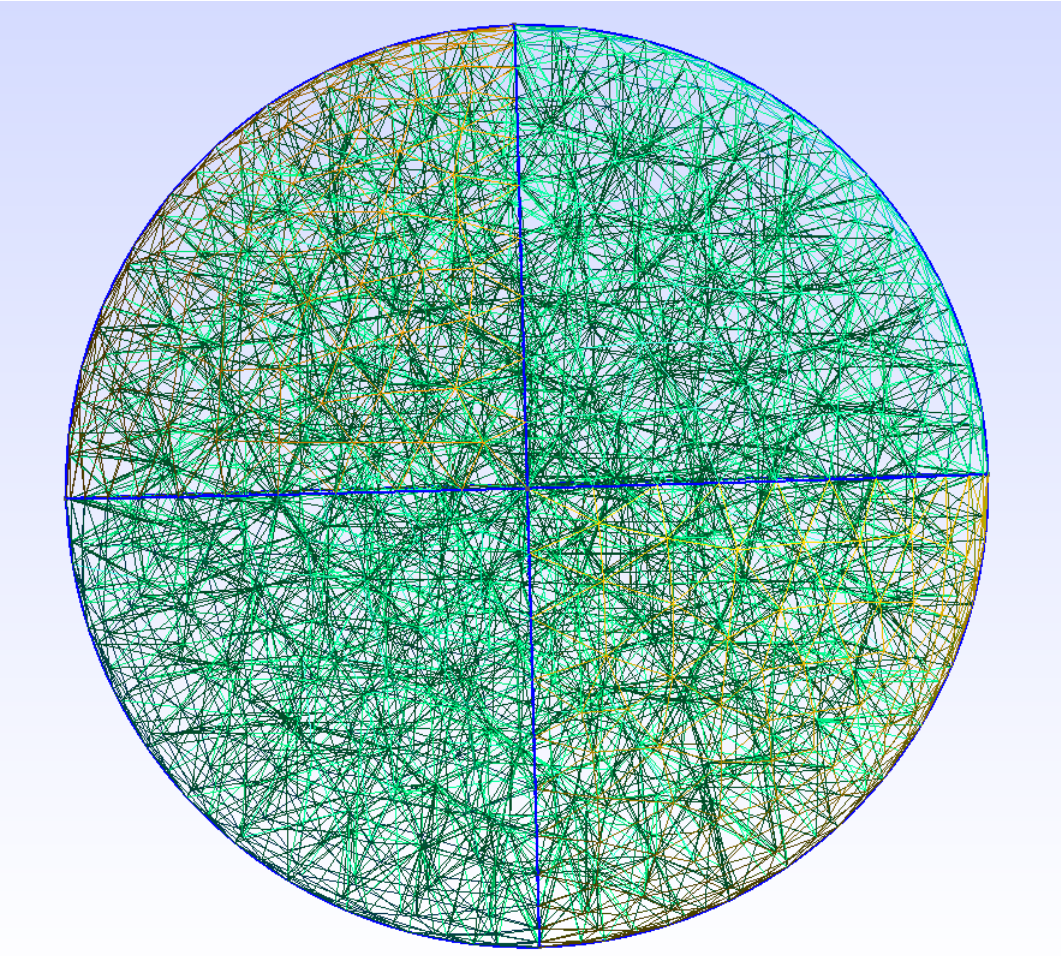

Figure 18 – Bubble mesh

For starter, let's see how impactful on the memory usage the old method is with increasing extended domain size and MPI process.

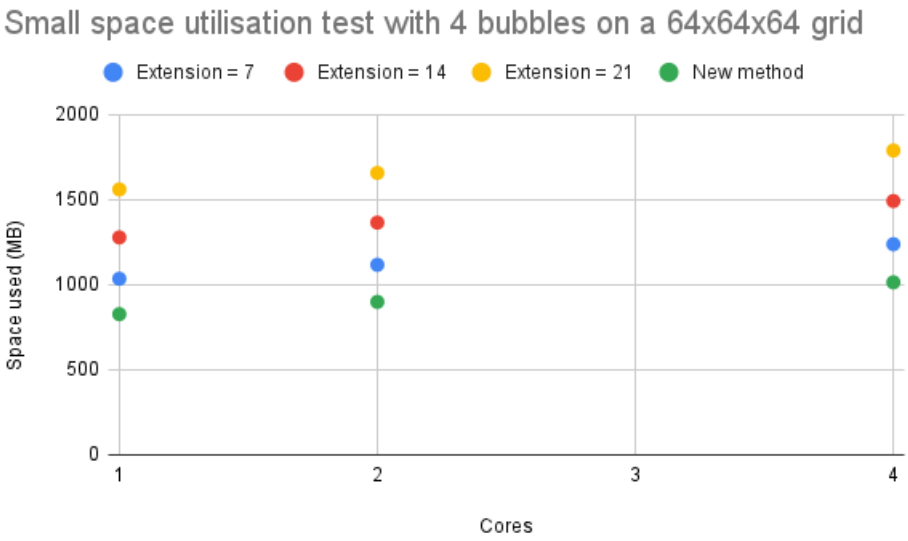

Figure 19 – Evolution of space utilisation

This graph displays the increasing space usage when using higher domain extension values and MPI process. Whereas the new implementation only increases in size with the number of MPI process. This can hold back users from simulating more complex models. However, in order to ensure the periodicity, the size of the extended domain need to be increased when the bubbles are relatively big compared to the domain.

With the new implementation, the space gain is at least 20%. And this gain only increases with the size of the extended domain. However, this difference decreases as the domain gets filled with bubbles. In the example above, the difference between the new implementation and the previous one with `ijk_splitting_ft_extension` = 7 is around 25%.

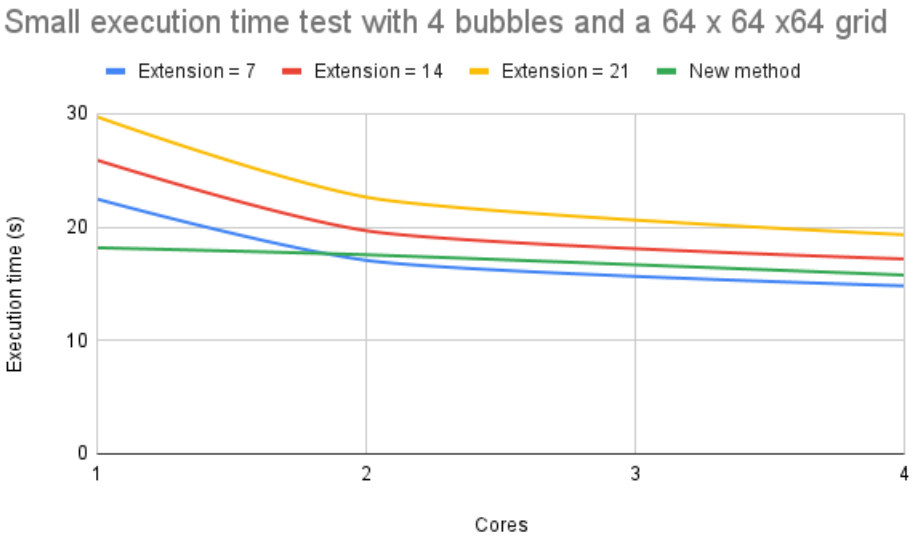

Figure 20 – Evolution of execution time

The results of those three graphs were obtained with the test case `ijkft_transport_perio` and the previ-

ously mentioned modification, on a laptop. However, these results do show that the previous method is somewhat efficient with up to 30% decrease in execution time if we double the resources allocated for the same problem. And, something that's not shown on these graphs but in the files generated by TRUST, the time spent in communication for sends and receives is at most 10%. 80% of those communications are for the movement of the interfaces and 5 to 10% for the redistribution, due to the extended domain.

On the other hand, the new implementation seems to have a harder time scaling with increasing number of of resources at first glance and is slightly less effecient than the old implementation on a higher MPI process count. Furthermore, the time spent in MPI communication is more significant due to the new data structures that were implemented and need to be shared around processors.

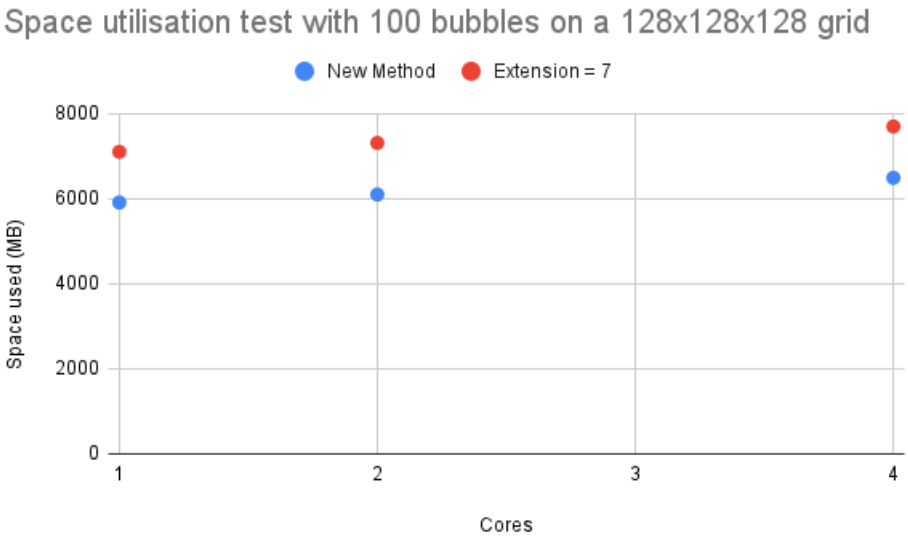

Figure 21 – Evolution of space utilisation

On a more complex test case, greater number of bubbles and finer mesh, the new method still has the advantage on memory load in sequential or parallel.

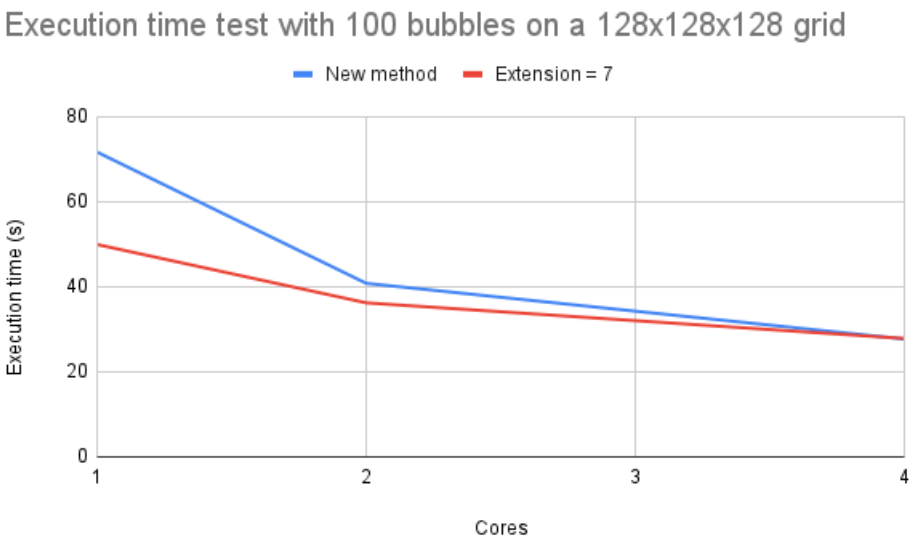

Figure 22 – Plot for ijk_splitting_ft_extension = 7

For a more complex test case, with 100 bubbles, the new implementation has worse single core performance. However, the execution time is close to being divided by 2 by doubling the resources allocated. Which means it has potential to become strong scaling if refined or launched on a more stable system like a cluster.

5.2 Bugs

Multiples compile options : the optimised version can work while the debug version will crash and call an error. After fixing those errors with gdb, these following issues not been met.

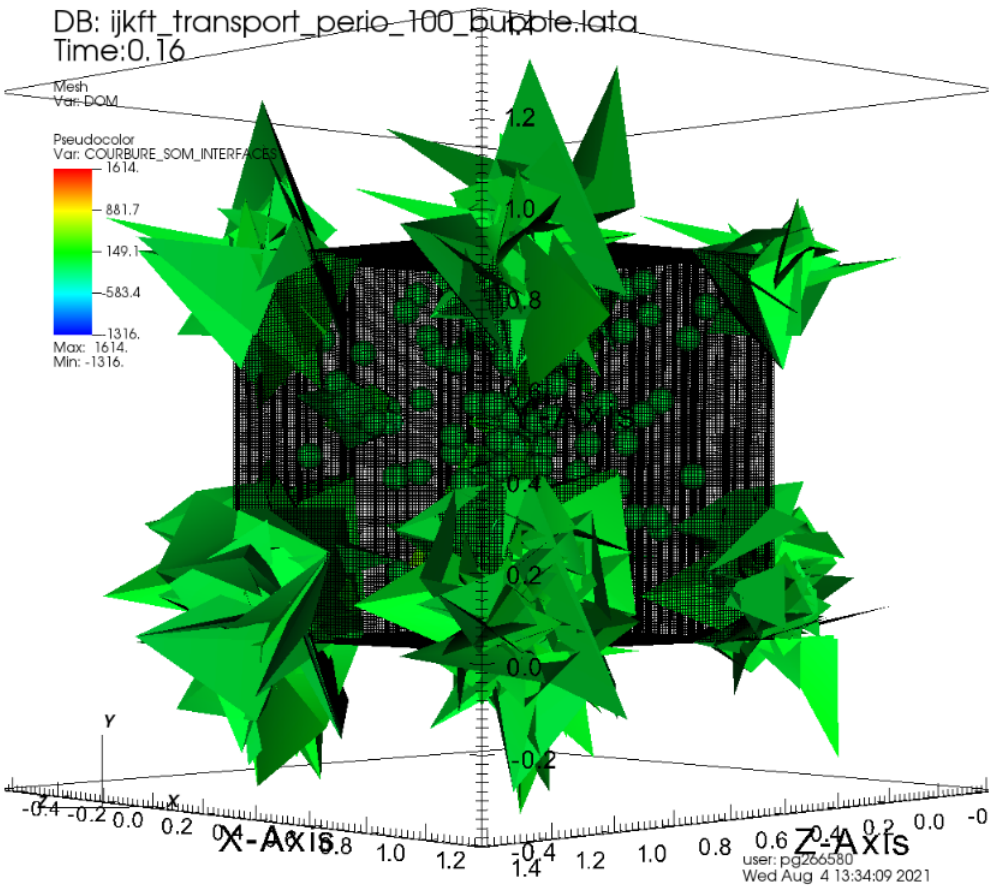

Figure 23 – One of the issues that happened to my predecessor

This was one of the issue mentioned in the report of Pierrick GUICHARD, who previously worked on this code. It never mentioned whether or not it was fixed but it was observed. Since I started with debugging and re implementing all the methods from the master branch of Trio 1.9.1, I never got to see this issue.

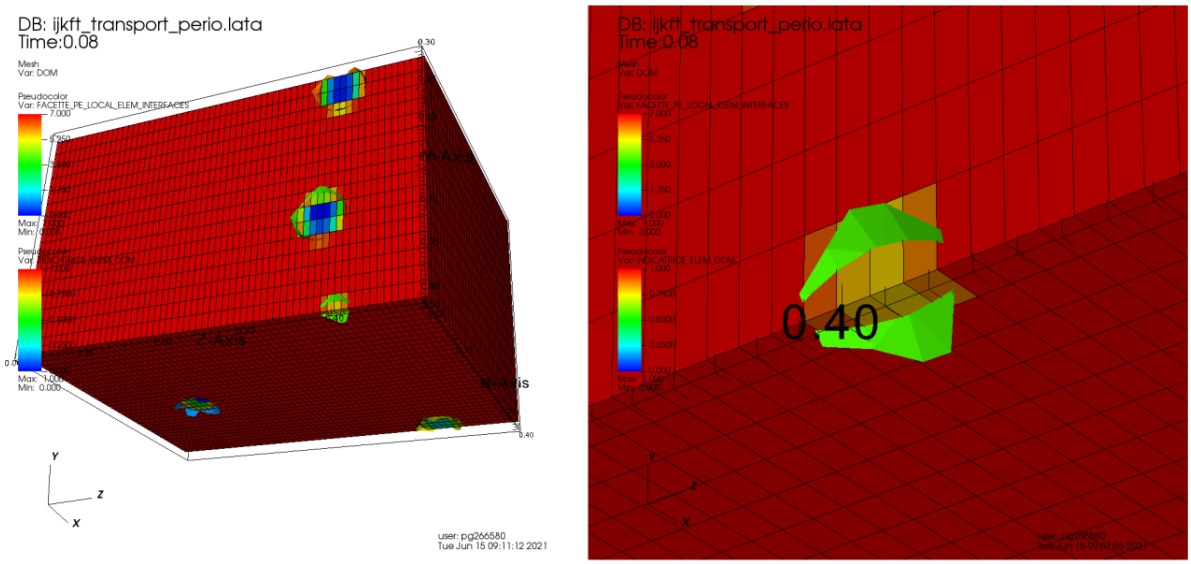

Figure 24 – Faces missing, one of the issues that happened to my predecessor

This is also an issue I never got to see. The bubbles I observed had issue but none of them had missing facets.

I had an issue where the different facets of the bubbles could change their compo id, which is the bubble's global id. I tried to see if the number of processors and/or the quality of the mesh had any impact on this issue but none of those factors were the cause of this. I fixed this issue multiple time but it would reappear if I wasn't careful with which data structure I was using in some methods.

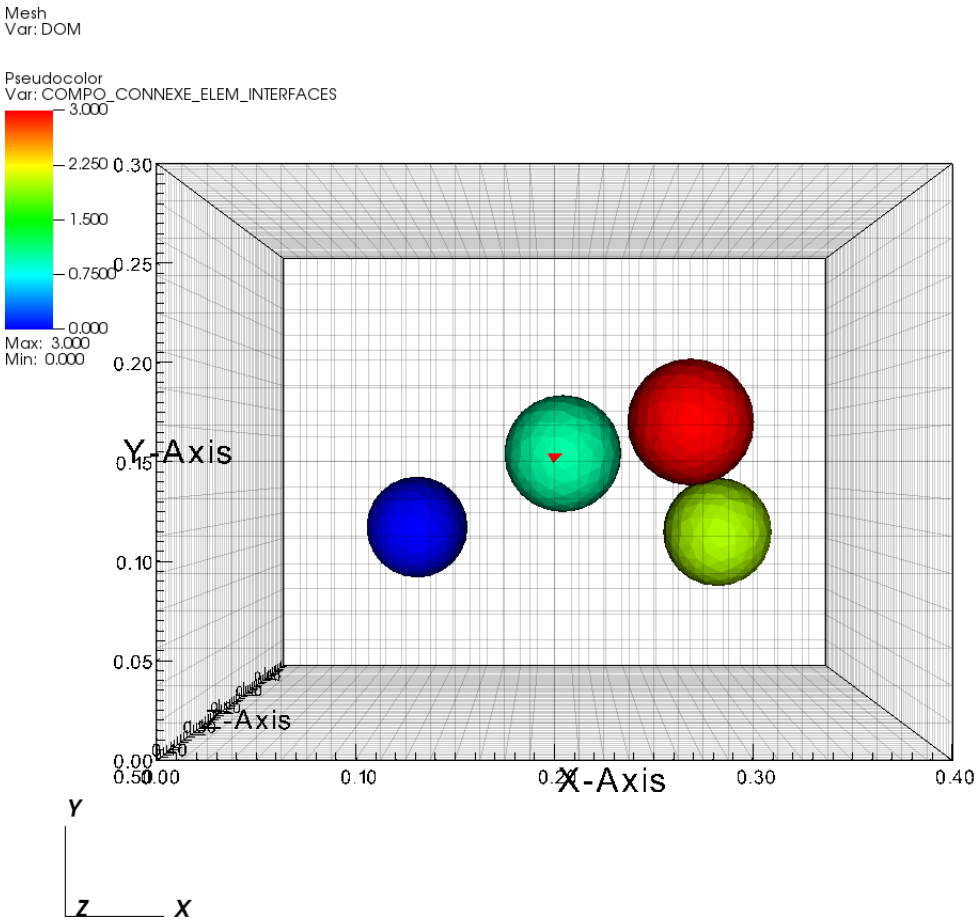

Figure 25 – Facets having the wrong compo id : with 8 processors

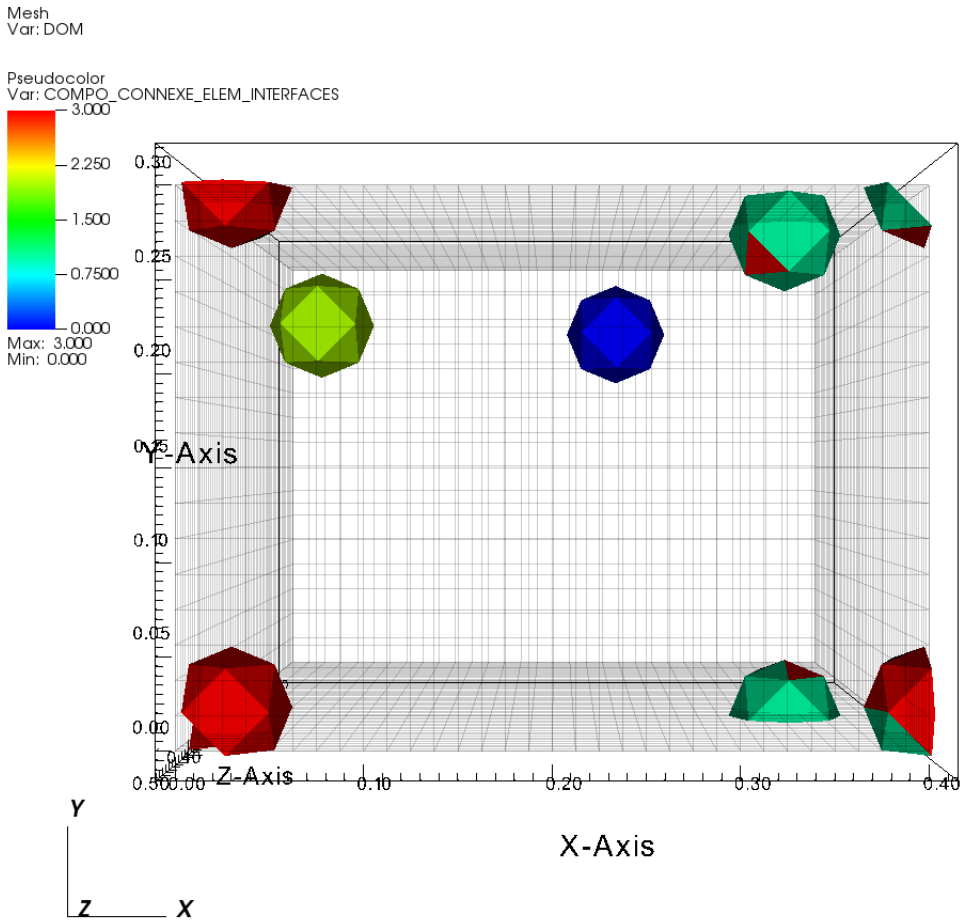

Figure 26 – Facets having the wrong compo id : with 8 processors and a coarser mesh

Among the list of issues I've encountered, some were also dependent on the position of the bubbles at the initial state (t = 0). If any bubble is too close to the periodic boundary, the program will run but it's going to raise a lot of errors in the debug mode.

The bubbles can be wrongly counted in multi process with the methods implemented in TRUST. These methods, `search_connex_components_local` and therefore `compute_global_connex_components`, count the number of bubbles by checking the number of structures that have all the facets connected. This issue happens when bubbles come close to the domain space of another processor, they do not get counted properly. I have yet to find a fix for this issue. My guess is that either these methods do not have access to `sommets_etendus_` or there is something wrong in that structure in some cases.

The other issue that is complex to fix is computing the curve on each bubble in parallel. The algorithm goes through all vertices in its domain, but when they cross the border toward another processor, it gets complicated. While this bug is very sporadic, it only happens in specific cases, I need to do more testing to really figure out a fix for it.

6 Conclusion

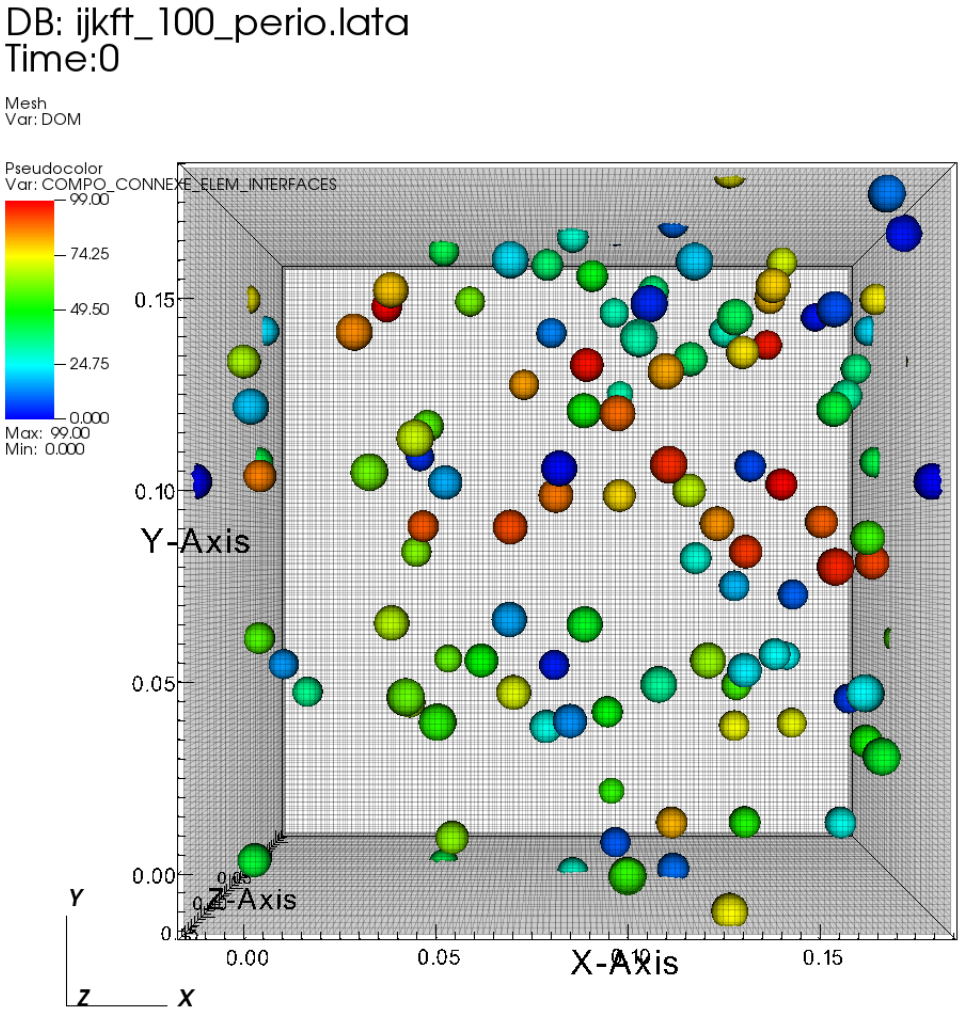

Figure 27 – 100 bubbles in a 128x128x128 grid

The new method is fully implemented on TrioIJK and still a work in progress for TrioCFD. A lot of computation time could be saved by using an hybrid approach in the algorithm in theory. A lot of methods in TrioIJK and TrioCFD could use multi-threading to improve the performance even further and drastically reduce computation time. However, this has to be tested for more certainty. The new implementation managed to improve the data consumption by a lot. Furthermore, the extended domain still "exist" and has not been entirely removed yet. Thus, TrioIJK will be even more efficient from a memory standpoint in the future, enabling its users to create more complex and intricate models.

Bibliography

[1] *CEA*. URL: https://www.cea.fr/.

[2] *Github for the source code of TRUST and TrioCFD*. URL: https://github.com/cea-trust-platform.

[3] Stéphane Zaleski Grétar Tryggvason Ruben Scardovelli. *Direct Numerical Simulations of Gas–Liquid Multiphase Flows*. Cambridge University Press, 2011.

[4] Cyril W. Hirt and Billy D. Nichols. "Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries". In: *JOURNAL OF COMPUTATIONAL PHYSICS* (1981).

[5] DOMINIQUE LEGENDRE and JACQUES MAGNAUDET. "The lift force on a spherical bubble in a viscous linear shear flow". en. In: *Journal of Fluid Mechanics* 368.11 (1998), pp. 81–126. DOI: https://doi.org/10.1017/S0022112098001621. URL: https://www.cambridge.org/core/journals/journal-of-fluid-mechanics/article/lift-force-on-a-spherical-bubble-in-a-viscous-linear-shear-flow/6B8108A588A8E562EBB99A6558BAB774.

[6] *Marker and Cell*. URL: http://plaza.ufl.edu/ebrackear/.

[7] *Master Course for Fluid Simulation Analysis of Multi-phase Flows by Oka-san*. URL: https://www.cradle-cfd.com/media/column/a105.

Glossary

DNS Direct numerical simulation is a technique for the study of turbulence in which the Navier-Stoke equations, the governing equations of fluid flow, are solved with sufficient resolution to represent all the scales of turbulence. 6

RANS Numerical method to model a turbulent flow wherein the flow quantities are decomposed into their time-averaged and fluctuating components. 6

VOF Volume of Fluid. Ratio of a certain fluid in a two-phase problem. 2, 8, 9