27th INTERNATIONAL CONGRESS OF MECHANICAL ENGENEERING

Brazilian Society of Mechanical Sciences and Engineering
27th ABCM International Congress of Mechanical Engineering
December 04-08, 2023, Florianópolis-SC, Brazil

COB-2023-0819 REDUCING THE INITIAL COMPUTATIONAL COST OF COMPLEX TURBULENT FLOWS SIMULATIONS BY USING RECORDED DATA FROM AN EXISTENT SIMULATION

Johnatas Teixeira de Freitas
João Marcelo Vedovotto
Millena Martins Villar Vale
Laboratório de Mecânica dos Fluídos - Programa de Pós Graduação em Engenharia Mecânica - Universidade Federal de Uberlândia
johnatas@ufu.br, vedovotto@ufu.br, millena.villar@gmail.com

Ricardo Serfaty
Petróleo Brasileiro SA
rserfaty@petrobras.com.br

Flávio de Oliveira Silva
Programa de Pós Graduação em Ciências da Computação - Universidade Federal de Uberlândia
flavio@ufu.br

Abstract. *In the computational simulation of complex turbulent flows, for instance, involving fluid-structure interactions and/or turbulent combustion with detailed chemistry, the computational cost can be cumbersome, even using massively distributed computing. Normally these detailed simulations are transient flows and as initial conditions, they retain some kind of unperturbed uniform flow, hence, one must wait until the simulation reaches a state of statistical stability in terms of statistical moments (normally up to the second). As a form of reducing this initial computational cost, we proposed a tool that used recorded data from an existing simulation to start a new simulation, thus enabling methods that required stabilized simulations to run from the beginning instead of waiting until a certain point. This process is usual in commercial CFD software, however, to the best knowledge of the authors, this procedure does not exist for software that relies on distributed dynamic adaptive meshes. Here, it is implemented with domain re-decomposition into a structured-block adaptive mesh. The procedure is tested and validated for numerous cases using the MFSim platform.*

Keywords: *computational fluid dynamics, computational simulation, adaptive mesh refinement, complex turbulent flows, simulation optimizations*

1. INTRODUCTION

Computational simulations of complex turbulent flows usually start as statistically unstable simulations, requiring processing time to reach a more stable state. This can take days, weeks, or even months. To allow a simulation to start from the already stable state produced by a previous run of the same simulation, we developed a tool that allows a new simulation to import data and mesh from another and start its run using this information.

Nearly all computational fluid dynamic (CFD) software provides a way to store the current state, a **snapshot**, of a simulation from time to time. In this snapshot is registered all data of interest and also the mesh state. Depending on how the snapshot is built it can also be used for post-processing operations.

The computational framework employed to develop the present work is the MFSim (MFLab, 2023), a parallel CFD software written mostly in Fortran with some modules in C/C++, developed by the Fluid Mechanics Laboratory of the Federal University of Uberlândia, Brazil. This computational platform's development started with the work of Villar (2007), and has been continually developed through the years into a multi-disciplinary code. Nowadays, this platform application allows simulating 3D problems involving: turbulent flow (Vedovoto *et al.*, 2015; Damasceno *et al.*, 2015), fluid-structure interaction (Neto *et al.*, 2019; Souza *et al.*, 2022; Stival *et al.*, 2022), multi-phase (Pivello *et al.*, 2014; Barbi *et al.*, 2018; Pinheiro *et al.*, 2019, 2021), gas-solid and gas-liquid flows (Santos, 2019), reactive (Damasceno *et al.*, 2018; Castro *et al.*, 2021) and counts even with LES approaches considering isotropic and anisotropic modelings. Recently, the MFSim code was used for the assessment of disposal operations of hypersaline solutions, as the use of local environmental regulations is vital to minimize the impact on marine ecosystems (Mota *et al.*, 2023).

MFSim framework also possesses the **snapshot** functionally, storing the data on files according to user configuration. One possible configuration is to use HDF5 files for that purpose. Usually, an HDF5 file is generated from time to time, again according to user configuration, containing the snapshot of a simulation.

Since MFSim uses block-structured adaptive mesh with multilevel-multigrid as the main solver (Villar, 2007), with virtual and physical levels composed of **patches**, the HDF5 files generated are related to this structure. Hence the HDF5 files work as hierarchical containers with each physical level being represented on the file. All data for each level, from all processes, are stored on the file as an array of bytes, a **datastream**. The boundaries for each data on the datastream are stored in metadata, again, for each level. Additionally, data regarding the current mesh and the patches distribution throughout the domain for each level, are also stored as metadata in the HDF5 file.

The tool we present in this paper reads this HDF5 file and imports data and mesh from all physical levels that exist both in the current and previous simulation. It also allows domain re-decomposition in a new process configuration.

2. TOOL FLUXOGRAM

Since MFSim generates one HDF5 file as a snapshot we start the new simulation by opening this file. To make better use of the HDF5 native API (which is written in the C language) and also the Operating System Filesystem API, we've implemented this part on C++ (Conic, 2023). All needed structures to extract levels, data, and patches from the file are allocated *on demand*.

After opening the HDF5 file, we read the metadata regarding the levels, the variables (data), and the mesh stored on it. Then, after checking that at least one variable and one physical level on the current simulation is also present on the file we proceed to import the mesh.

Usually, MFSim creates the mesh shortly after a few initial checks. As we needed to import the mesh from the HDF5 file, we created a bypass. First, we create the mesh topology considering the current simulation process distribution throughout the domain. Then we create the virtual levels required by the multigrid solver. After, we import the mesh from all physical levels that are present both in the current simulation and the HDF5 file, ignoring those that aren't. This is necessary because it's possible to change this setting between simulations: the first run can have, for example, 5 physical levels, and the second run, the one started with the HDF5 file, can have 3 or 6 physical levels. In case of a reduction in physical levels between simulations, the "excess" levels in the file (during the second run) are ignored and in case of an increase (again, in the second run), the additional levels are created sequentially by a combination of the refinement criteria and the level directly below.

If the physical level on the current simulation is found on the file, all metadata regarding the patches from that level are then retrieved. If the current process distribution is the same contained in the HDF5, then the patches are created on the current simulation as they are described in the file. If not, the tool interpolates the patches redistributing them throughout the new domain decomposition. Section 2.1 describes this feature.

After the mesh is properly imported/created we proceed to the data. As occurred with the mesh, it's possible to change the variable settings between simulations. Since there is a large number of possible combinations of which variables can be used in a simulation we make the availability test from the file to the simulation in opposition to what we did when importing the mesh. Therefore, we read the metadata describing the variables (or datafields) contained in the file and check if the current simulation is using the same variable. If it is, we import it. If not, we ignore it. As for the variables existing in the current simulation but not in the file, we initialize them in the same way they would be when we are just starting the simulation without the use of the HDF5 file.

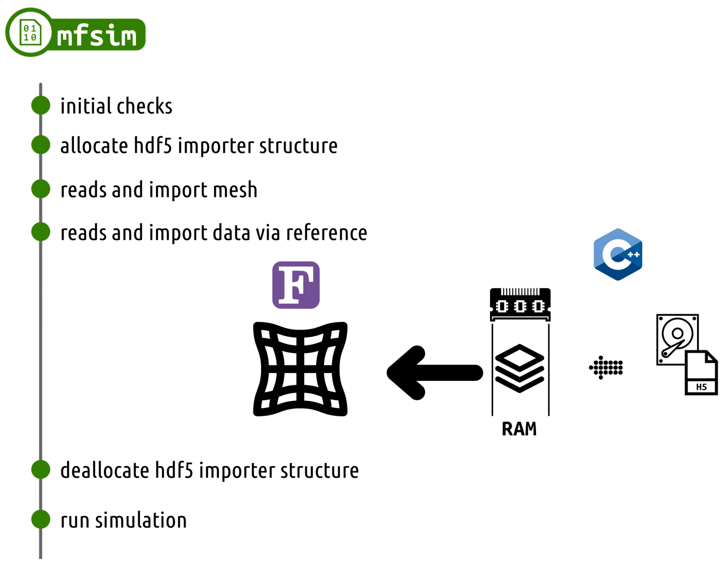
mfsim
initial checks
allocate hdf5 importer structure
reads and import mesh
reads and import data via reference
RAM
H5
deallocate hdf5 importer structure
run simulation

Figure 1. Tool fluxogram

Another important feature to describe here is how the variables from the HDF5 file are managed. In MFSim all data (variables) are defined in Fortran with references being shared with C and C++ code. This prevents double allocations for data which ends up optimizing memory use. But, in this tool, we have implemented the majority of the code in C++ and we need to share the data from C++ to Fortran. The use of C++ is not by mere choice, but because as explained before, C++ provides better API to access both the HDF5 file and the Operating System Filesystem. There's another motive for the use of C++ which is to take advantage of the language's dynamic container structures like vectors (cplusplus, 2023). Since we are going to import data of varying sizes stored in a datastream, dynamic containers like the C++ vector can become very handy. So to optimize memory consumption we take the strategy already implemented in MFSim and share with the Fortran code only the reference to the data stored in the C++ vector. A more detailed description of this feature is provided in Section 2.2 For now we only need to know that the data is extracted in parallel from the HDF5, stored in a C++ vector (in each process) and its reference is shared with the Fortran code.

The order used to store data in the vector (its iteration direction) coincides with the patch-filling order (its iteration direction). This ensures that we can iterate over the vector reference in Fortran in the same way we iterate over the patch, allowing us to inject the data from the vector directly into the patch without the need for additional operations. This process is executed variable by variable, for all patches and levels that exist on the file and the current simulation. For levels or patches that do not exist in the HDF5, the data are interpolated from the level immediately below.

After importing the data all structures allocated by the tool are liberated and the simulation is free to run its course as summarized in the fluxogram shown in Figure 1.

2.1 MESH INTERPOLATION

To understand how and why the tool interpolates mesh we must first understand how multigrid levels and patches work in MFSim. The simulation's physical domain is represented as a grid of points and vertices, a mesh. This is the first physical level. Inside it, there can be others sub-grids that represent refinement areas that compose the level above the first and allow for more accuracy in whatever equations the simulation is trying to solve. Each one of those sub-grids can have other sub-grids composing levels above and so on. Figure 2 shows an illustration with a 3-level multigrid mesh.

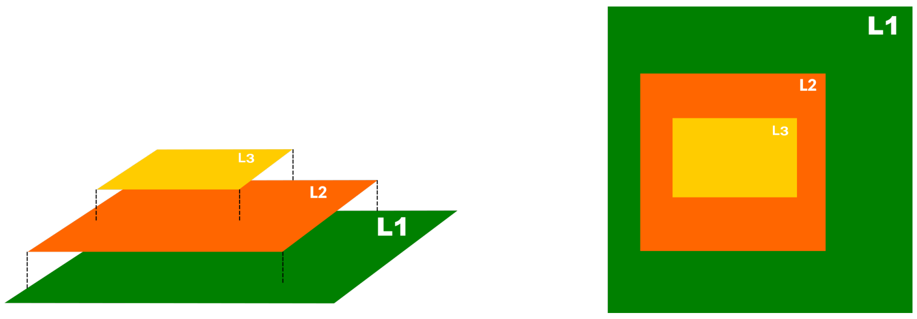

Figure 2. Levels on multigrid mesh

Each level is also composed of at least one patch considering the geometry of the problem solved and some rules developed by Marsha Berger on her adaptive mesh refinement algorithm (Berger and Oliger, 1984). Below the first physical level, are the virtual levels required by the multilevel-multigrid solver (Villar, 2007). This whole structure is distributed amongst the process used in the simulation in such a form that all process has all virtual levels and the first (physical) level, while refinement levels may or may not be present in all processes. The HDF5 file reflects this structure.

Hence, when there is a change in process configuration on the simulation started with the HDF5, the mesh structure stored in the file must be redistributed between the new processes of the current simulation. Therefore, all patches and levels (from the HDF5 file) must be remapped according to the new setting. Since we cannot remap patches (or levels) to something that does not exist, we must first create the mesh topology of the current simulation, which will provide us with the **coordinate space** for all possible levels contained in each process.

Following that, we read the mesh metadata contained in the HDF5 file. This must be done by all processes as we do not know yet which levels and/or patches are to be remapped. With this metadata, we create a tridimensional model of the mesh contained in the HDF5 file. This model is lightweight since only contains the coordinate space (two sets of coordinates, where the space starts and where it ends) of each patch and level, and in what process they were originally assigned to.

Since we now know the process distribution used when the HDF5 was generated (previous simulation) we can also mount a map relating the old process distribution with the new one. Subsequently, we iterate over the levels and the patches from the HDF5 to test if they fit totally, partially, or not at all on the current process coordinate space. If they do not fit they're ignored. If they fit totally, then we create the same patch on the same level in the current simulation. If they fit partially, we cut the patch considering the boundaries imposed by the current process coordinate space, effectively

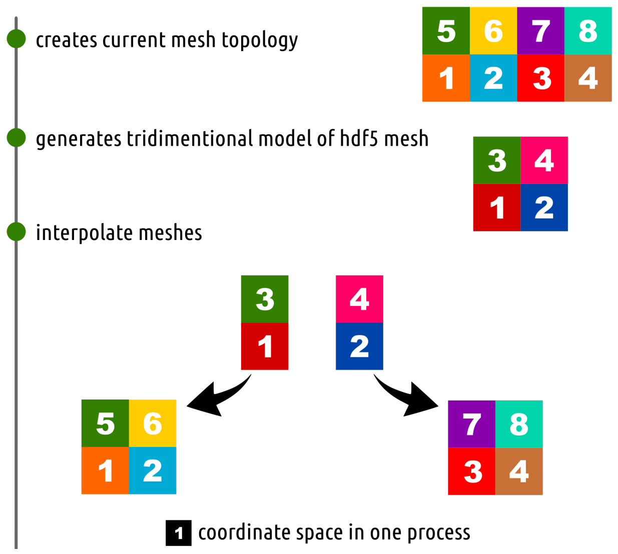
creates current mesh topology
5
6
7
8
1
2
3
4
generates tridimentional model of hdf5 mesh
3
4
1
2
interpolate meshes
3
1
4
2
5
6
1
2
7
8
3
4
1
coordinate space in one process

Figure 3. Mesh interpolation

creating a sub-patch that fits the process. As this is done in parallel, all patches contained in the HDF5 are remapped by a direct copy or a split copy of the patch in the current domain decomposition. Figure 3 shows a graphical representation of this whole process.

After all levels and patches in the file that could be retrieved have been interpolated to the current simulation, the tool proceeds to import the data.

2.2 PARALLEL DATA RETRIEVAL

The data stored in the HDF5 files are organized as an array of bytes, a datastream, for each physical level existing in the file. This single datastream, per level, contains the data from all variables of interest from all processes on that level and is generated part by part, by each process. Subsequently, metadata regarding the boundaries of each patch, per process, are also stored on the file.

So, to retrieve this data we must first identify the variables. Then, calculate the boundaries for each patch in each process (process information contained in the HDF5 file, not the current simulation process distribution) which will serve as boundaries for the variables as well. With those boundaries, we can calculate the *shift* required to extract each variable data for each patch for each process from the datastream. Since this is a parallel operation, each process (the ones in the current simulation) uses the process map mounted during mesh interpolation to discover which patches from the previous simulation are now assigned to them (even those patches who was splitted). Once that is known, the shifts are properly calculated and we can begin extracting the data from the file to the C++ vector.

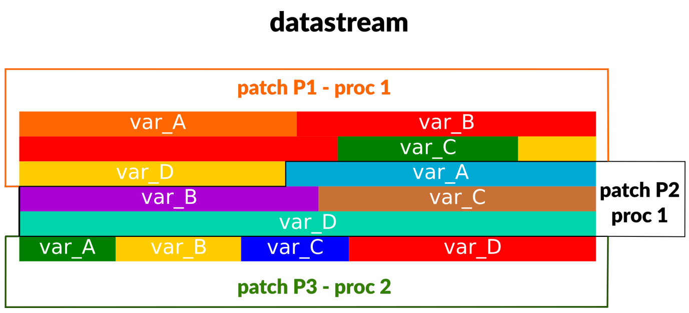

Figure 4. datastream containing 4 variables and 3 patches distributed in 2 processes

Figure 4 illustrates how a datastream is stored in the HDF5, exemplifying a datastream that contains patches P1 and P2 belonging originally to process 1 and patch P3 originally belonging to process 2. As an exercise to understand the procedure described before, let us assume process 1 is to be remapped in the current simulation to processes 1 and 2 and the former process 2 is to be remapped to the current processes 3 and 4. When retrieving data, both current processes 1 and 2 are going to retrieve variables A, B, C, and D from former patches P1 and P2 while current processes 3 and 4 are going to retrieve variables A, B, C, and D from former patch P3. The *shifts* for each variable A, B, C, and D are going to be calculated considering the original patch and process distribution contained in the HDF5 file, and the variables data will be extracted from the file and put into the C++ vector. Then the reference for this vector is passed to the Fortran code where the data are going to be assigned to the proper patch, level, and processes in the current simulation.

Since the vector can contain data that are not bound to the current setting of patches and processes, another *shift* must be calculated separating the data that are bound to the current patch and process for those that aren't. Taking the example again, the datastream shown in Figure 4 was originally generated from 3 patches distributed in 2 processes and we are now redistributing them to 6 patches in 4 processes. In that scenario var_A in patch P1 is to be now splitted between patches P1_PROC1 in process 1 and P1_PROC2 in process 2. That's why we need to calculate and use the second shift, otherwise, data bounded for patch P1_PROC1 will be wrongly assigned to patch P1_PROC2.

After finishing this splitting the Fortran code proceeds to inject the data in the current patch and level in each process. This procedure then is repeated for each variable in all patches in all processes stored in the HDF5 file that are going to be imported to the new simulation. That means a lot of loops and data structures been created, reused, or destroyed which creates a need for a properly organized code.

Beyond that, to effectively extract data from a HDF5 file we must dive into its internal structure. A physical level is stored as a group while metadata are stored as attributes and variables data, as datasets. Combining this structure with the structure the file represents, levels, and patches from the multigrid solver, we have a series of different objects that share some characteristics and differ in others. That, combined with the execution flow already explained, brings the necessity of *design patterns* (Erich Gamma and Vlissides, 2000) to properly organize the code, maximize code reuse, and avoid waste in memory or processing time. In that regard we make use of two *design patterns*, the Abstract Factory and the Façade.

We've used Abstract Factory (Guru, 2023a) in the objects that represent HDF5 internal structure that shares some characteristics, are of interest, and have the same execution flow, like groups, attributes, and datasets. For instance, they all need access to the file. They also need iterators to access them as there are multiple instances of them in the file. At the same time, they represent different information with different structures: attributes are containers-like structures while datasets are datastreams. So we create an object to describe the common operations between those objects. This object is inherited by the specialized objects that implement those common operations considering their specialty. Thus we provide a single unified interface for all HDF5 internal objects.

As for the Façade (Guru, 2023b) design pattern, we encapsulated all functions regarding the tool into a single interface that MFSim can use without needing to know any details of it. For example, to discover which variables the HDF5 file can provide to the current simulation, we need to access the file, iterate over its groups and attributes, obtain the information as it is stored, and then organize it in a way that can be used by MFSim later to request those variables. That's a lot of operations and procedures to get that, so we grouped all needed procedure calls to achieve that into a single procedure that can be called by MFSim and return a list with the variable's names. That is the Façade design pattern.

3. TESTING THE TOOL

To test the tool we run the flow past a stationary sphere test case described in MFSim's user guide (MFLab, 2023). This test case solves velocity, one per axis (x/u, y/v, z/w), and pressure, runs for more than two thousand timesteps, has immersed boundary physics, and a turbulent flow been formed in the back of the sphere after the 500th timestep. Our main objective is to take the simulation in a state where the turbulent flow is already happening and is statistically stable and start a new simulation with this data.

To achieve that, we identified an adequate timestep where the turbulent flow is already happening in a state of statistical stability. Since the turbulent flow starts being formed in the back of the sphere after the 500th timestep and remains in that state at least to the 800th timestep, we decide to use the 1000th, as the turbulent flow is fully formed (as we needed) and the simulation is somewhat half executed.

Then we proceeded to configure a new simulation using the same test case but this time, changing the start conditions to use the data contained in the HDF5 file generated in the 1000th timestep. We also changed the process distribution from 4 processes to 8 processes, doubling the number of processes assigned to the x/u axis.

The test will be considered successful if the tool is capable at the start of the new simulation import both the mesh and data from the previous one.

4. RESULTS

Figure 5 depicts how the flow past a stationary sphere test case usually starts (rendered on velocity in axis x/u). The flow has not started yet, the sphere is inside the center of the boxes and there is no refinement area in the back of the sphere, indicating there is nothing of interest happening there at that timestep.

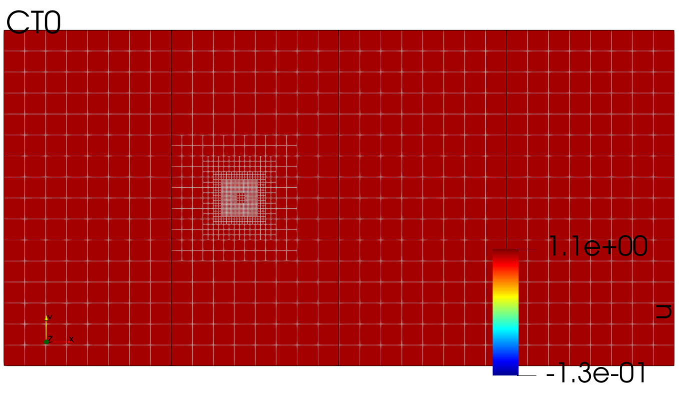

Figure 5. sphere test case start

That changes entirely when we reach the 1000th timestep as shown by Figure 6. Instead of a mostly stable velocity gradient, we can see more velocity variability throughout the mesh. Also, there are more refinement areas in the back of the sphere who wasn't present when the case started. Since we are using adaptive refinement mesh, that indicates that something is happening in that region.

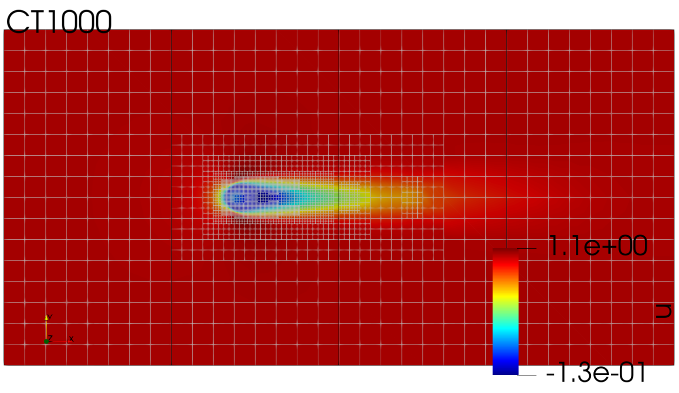

Figure 6. sphere test case 1000th timestep

Now let us see if we can import this same state to the new simulation.

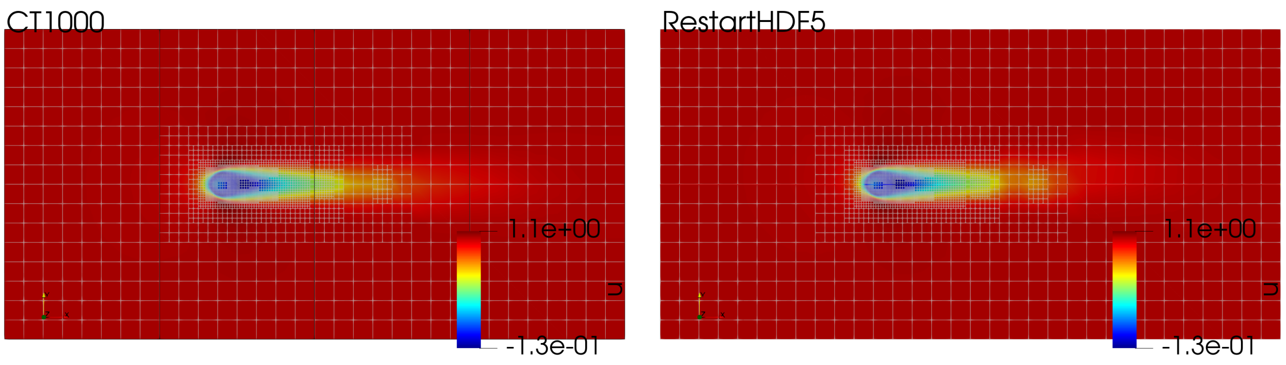

Figure 7. Left: original mesh in hdf5; Right: imported mesh

As Figure 7 presents, the mesh from the 1000th timestep was properly imported into the start timestep of the new simulation. Also, considering that we changed process distribution, we can see that the tool adequately remapped all patches and levels to the new setting as expected.

Evaluation of the variable's gradients can provide an additional check. If the imported variables can match or be very close to the original ones, then the tool indeed imported the data as expected.

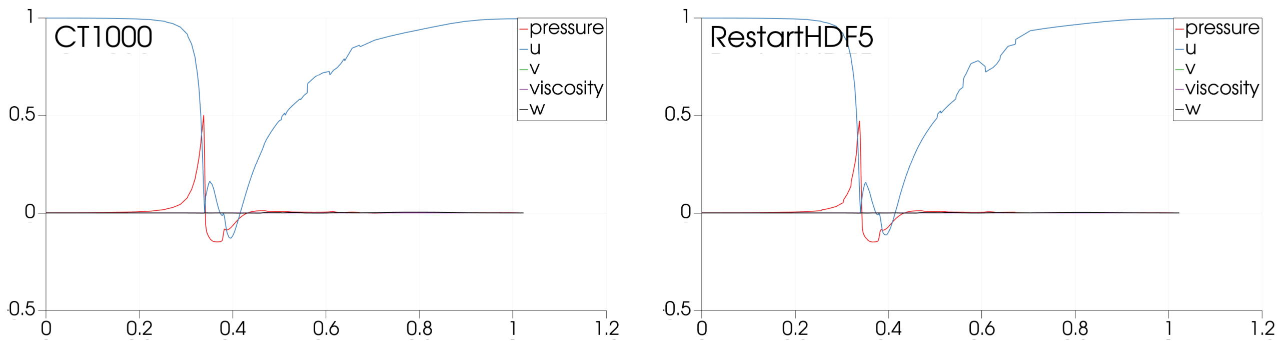

Figure 8. Variables gradients in the 1000th timestep (left) and imported (right)

In that sense, Figure 8 shows side by side the velocities and pressure gradients present in the HDF5 and the ones imported by the tool. We can see a small disturbance in one of the gradients which requires a more close inspection. With that in mind, we presented the gradients separately, variable by variable, in Figures 9 and 10.

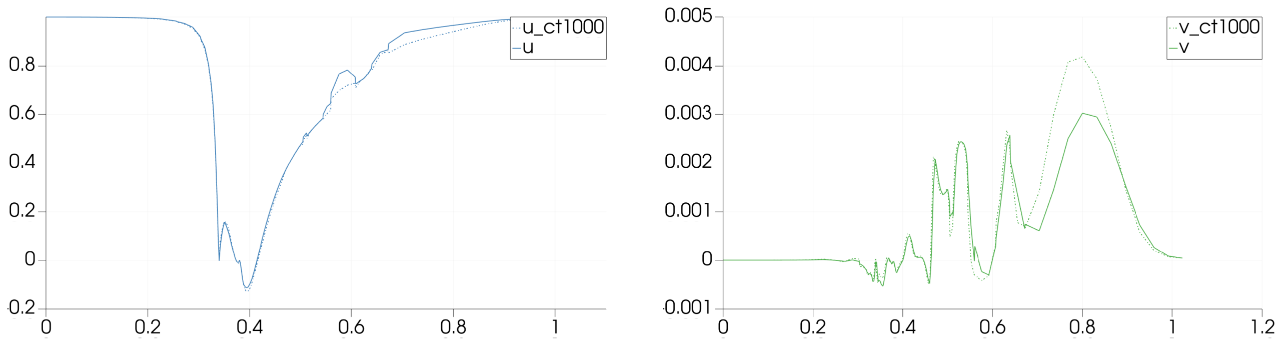

Figure 9. Velocity gradient in axis x/u and y/v

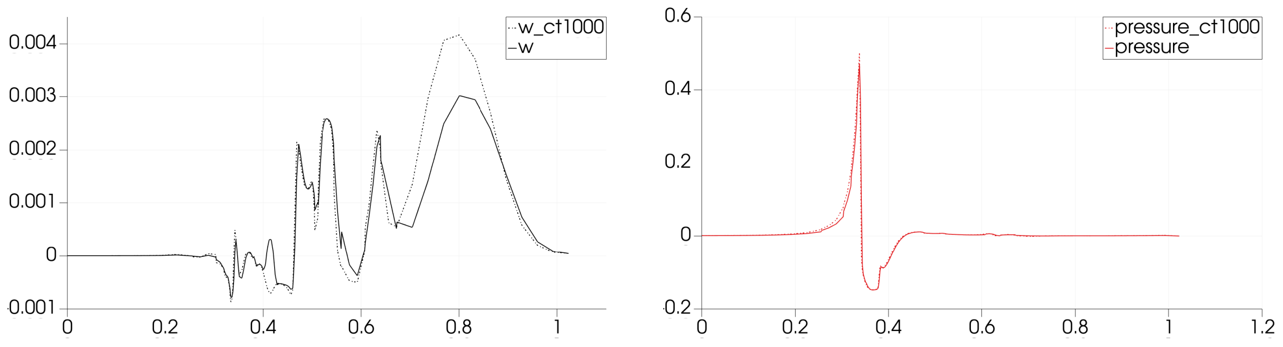

Figure 10. Velocity gradient in axis z/w and pressure

As shown by Figures 9 and 10 the velocity gradient for the x/u axis and pressure differs very little between the 1000th timestep and the imported version while gradients for velocity in y/v and z/w axis shows a small disturbance in a portion of the gradient. This disturbance is not caused by the process redistribution as we repeated the test without changing the process distribution and the results were the same.

A more adequate explanation for this is the combination of interpolations from virtual levels, created anew on the second simulation, and the data on the physical levels injected from the HDF5. In fact, we've implemented a restart tool using the same mesh interpolation and parallel data retrieval code from the tool described in this work, but for a restart propose instead of a start. Elaborating, this other tool allows the continuation of a simulation stored in a snapshot with process redistribution (hence why this other tool uses parts of code from this tool). It differs from this tool though, when it doesn't allow for a change in the number of physical levels or variable settings, in other words, besides process redistribution, everything isn't changed between runs, which includes the retrieval of virtual levels. With virtual levels being transported from one run to another, and with no changes on the physical levels, the disturbances in the gradients disappear, providing good evidence that the disturbances observed here are caused by the changes in level, patches, and process configuration between runs.

On the other hand, the differences observed in the variables when starting a new simulation with data and mesh recorded from another simulation weren't enough to change the expected results. After the tool finishes its work, the simulation runs its course as expected and MFSim's subsequent operations (including remeshing and interpolations) fix any disturbance found after importing data and mesh from the HDF5 file.

5. CONCLUSIONS AND FUTURE WORK

We've demonstrated in this work that is possible to implement a tool capable of importing both data and mesh from an HDF5 file and use that information as the start conditions of a new simulation. We tested our tool against the canonical flow past a stationary sphere test case and as shown in Section 4. it appears to be working as was designed to. As future work, we are testing this tool against more complex simulations, with more physics embedded, more complex meshes, and process redistribution to assess the full capability of this tool.

6. REFERENCES

Barbi, F., Pivello, M.R., Villar, M.M., Serfaty, R., Roma, A.M. and Silveira Neto, A.d., 2018. "Numerical experiments of ascending bubbles for fluid dynamic force calculations". *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, Vol. 40, No. 11, p. 519. ISSN 1806-3691. doi:10.1007/s40430-018-1435-7. URL `https://doi.org/10.1007/s40430-018-1435-7`.

Berger, M.J. and Oliger, J., 1984. "Adaptive mesh refinement for hyperbolic partial differential equations". *Journal of Computational Physics*, Vol. 53, No. 3, pp. 484–512. ISSN 0021-9991. doi:10.1016/0021-9991(84)90073-1.

Castro, L.P.d., Pinheiro, A.P., Vilela, V., Magalhães, G.M., Serfaty, R. and Vedovotto, J.M., 2021. "Implementation of a hybrid lagrangian filtered density function–large eddy simulation methodology in a dynamic adaptive mesh refinement environment". *Physics of Fluids*, Vol. 33, No. 4, p. 045126. doi:10.1063/5.0045873.

Conic, D., 2023. "Fortran filesystem blues". https://degenerateconic.com/fortran-filesystem-blues.html. Acessed 27 June 2023.

cplusplus, 2023. "std::vector". https://cplusplus.com/reference/vector/vector/. Acessed 27 June 2023.

Damasceno, M., Vedovoto, J. and Silveira-Neto, A., 2015. "Turbulent inlet conditions modeling using large-eddy simulations". *CMES - Computer Modeling in Engineering and Sciences*, Vol. 104, pp. 105–132.

Damasceno, M.M.R., de Freitas Santos, J.G. and Vedovoto, J.M., 2018. "Simulation of turbulent reactive flows using a fdf methodology – advances in particle density control for normalized variables". *Computers & Fluids*, Vol. 170, pp. 128 – 140. ISSN 0045-7930. doi:https://doi.org/10.1016/j.compfluid.2018.05.004. URL `http://www.sciencedirect.com/science/article/pii/S0045793018302494`.

Erich Gamma, Richard Helm, R.J. and Vlissides, J., 2000. *Padrões de Projeto: soluções reutilizáveis de software orientado a objetos*. Bookman. ISBN 9788573076103.

Guru, R., 2023a. "Abstract factory". https://refactoring.guru/pt-br/design-patterns/abstract-factory. Acessed 27 June 2023.

Guru, R., 2023b. "Façade". https://refactoring.guru/pt-br/design-patterns/facade. Acessed 27 June 2023.

MFLab, 2023. "Site oficial do mfsim". Laboratório de Mecânica de Fluídos, Programa de Pós Graduação em Engenharia Mecânica, Universidade Federal de Uberlândia, https://www.mflab.mecanica.ufu.br/mfsim/. Acessed 27 June 2023.

Mota, P., Henrique, Augusto, V., João Marcelo, S.N. and Aristeu, 2023. "Assessment of critical brine disposal operations conditions by cfd modeling and a kriging metamodel". *Environmental Fluid Mechanics*, Vol. 167, pp. 1573–1510. doi:10.1007/s10652-023-09911-7.

Neto, H.R., Cavalini, A., Vedovoto, J., Neto, A.S. and Rade, D., 2019. "Influence of seabed proximity on the vibration responses of a pipeline accounting for fluid-structure interaction". *Mechanical Systems and Signal Processing*, Vol. 114, pp. 224–238.

Pinheiro, A.P., Rybdylova, O., Zubrilin, I.A., Sazhin, S.S., Sacomano Filho, F.L. and Vedovotto, J.M., 2021. "Modelling of aviation kerosene droplet heating and evaporation using complete fuel composition and surrogates". *Fuel*, Vol. 305, p. 121564. ISSN 0016-2361. doi:https://doi.org/10.1016/j.fuel.2021.121564. URL `https://www.sciencedirect.com/science/article/pii/S0016236121014459`.

Pinheiro, A.P., Vedovoto, J.M., da Silveira Neto, A. and van Wachem, B.G., 2019. "Ethanol droplet evaporation: Effects of ambient temperature, pressure and fuel vapor concentration". *International Journal of Heat and Mass Transfer*, Vol. 143, p. 118472. ISSN 0017-9310. doi:https://doi.org/10.1016/j.ijheatmasstransfer.2019.118472. URL `https://www.sciencedirect.com/science/article/pii/S0017931019309214`.

Pivello, M., Villar, M., Serfaty, R., Roma, A. and Silveira-Neto, A., 2014. "A fully adaptive front tracking method for the simulation of two phase flows". *International Journal of Multiphase Flow*, Vol. 58, pp. 72–82. ISSN 0301-9322. doi:https://doi.org/10.1016/j.ijmultiphaseflow.2013.08.009. URL `https://www.sciencedirect.com/science/article/pii/S0301932213001286`.

Santos, J.G.d.F., 2019. "Mathematical and computational modeling of gas-solid flows in dynamic adaptive mesh". *Master thesis, Universidade Federal de Uberlândia*.

Souza, P.R.C., Neto, H.R., Villar, M.M., Vedovotto, J.M., Cavalini, A.A. and Neto, A.S., 2022. "Multi-phase fluid–structure interaction using adaptive mesh refinement and immersed boundary method". *Journal of the Brazilian Society of Mechanical Sciences and Engineering*. doi:10.1007/s40430-022-03417-x.

Stival, L.J., Brinkerhoff, J.R., Vedovotto, J.M. and de Andrade, F.O., 2022. "Wake modeling and simulation of an experimental wind turbine using large eddy simulation coupled with immersed boundary method alongside a dynamic adaptive mesh refinement". *Energy Conversion and Management*, Vol. 268, p. 115938. ISSN 0196-8904. doi:https://doi.org/10.1016/j.enconman.2022.115938. URL `https://www.sciencedirect.com/science/article/pii/S0196890422007348`.

Vedovoto, J.M., Serfaty, R. and Silveira Neto, A.D., 2015. "Mathematical and numerical modeling of turbulent flows". *Anais da Academia Brasileira de Ciências*, Vol. 87, No. An. Acad. Bras. Ciênc., 2015 87(2), p. 1195–1232. ISSN 0001-3765. doi:10.1590/0001-3765201520140510. URL `https://doi.org/10.1590/0001-3765201520140510`.

Villar, M., 2007. "Análise numérica detalhada de escoamentos multifásicos bidimensionais". *Ph.D. thesis, Universidade Federal de Uberlândia*.

7. RESPONSIBILITY NOTICE