CFD-HOWL: I/O Library for Writing Higher-Order CFD Data

Rayan Dhib[a], Vatsalya Sharma[a,*], Andrea Lani[a], Stefaan Poedts[a,b]

[a] *Centre for Mathematical Plasma Astrophysics, KU Leuven, Celestijnenlaan 200B, Leuven, 3001, Belgium*
[b] *Institute of Physics, University of Maria Curie-Skłodowska, ul. Radziszewskiego 10, Lublin, PL-20-031, Poland*

Abstract

This paper introduces the Computational Fluid Dynamics High-Order Writer Library (CFD-HOWL), which performs I/O operations on solution data from high-order CFD simulations. The library centers around the novel mesh upgrade algorithm, which is coupled with the functionalities of the CGNS file system. Even though the library is in Python, users can readily apply this library *as-is* by integrating APIs that read their specific data formats. The flexible nature of the library allows for straightforward integration with existing CFD solvers. We integrate CFD-HOWL with the in-house COOLFluiD CFD solver and demonstrate a reduction in I/O operation times. The results show that the CFD-HOWL consistently outperforms other conventional writers, thus addressing a key bottleneck with the higher-order CFD data post-processing.

Keywords: High-order CFD, CGNS, I/O operations, Spectral methods.

1. Motivation and significance

Numerical methods such as Finite Difference (FD), Finite Volume (FV), and Finite Element (FE) have been fundamental in Computational Fluid Dynamics (CFD). While FD and FV methods are widely used for their robustness, they typically achieve only first or second-order accuracy in most production codes. High-order methods, defined as those of third-order accuracy or higher [1], are increasingly adopted to improve solution fidelity, particularly for complex geometries [2]. Over the past decades, significant advancements have extended these conventional methods to higher orders. For example,

*vatsalya.sharma@kuleuven.be

the Compact Finite Difference method [3] enhances accuracy on structured grids, and high-order extensions of FV methods handle unstructured grids, albeit with increased computational complexity [4, 5]. However, the most versatile high-order approaches are found in FE-type methods, which excel on unstructured grids. Among these, the Discontinuous Galerkin (DG) method [6] and its variants have been widely adopted. In 2007, Huynh introduced the Flux Reconstruction (FR) approach [7], a unifying framework that simplifies the implementation of high-order schemes and offers computational efficiency. FR, in particular, has emerged as a promising method for achieving high-order accuracy while handling complex geometries, making it a focus of this paper. Using these higher-order methods introduces several challenges to storing, handling, and post-processing simulation data. These simulation data inherently use a large amount of disk space, which is proportional to the polynomial degrees in simulation. This poses significant challenges in efficiently reading, writing, storing, and visualizing the simulation data, thus creating bottlenecks in the overall workflow.

Recently, visualization tools have begun to support high-order elements natively. Through ParaView, the Visualization Toolkit (VTK) has introduced capabilities for visualizing arbitrary-order Lagrange finite elements, supporting high-order polynomials up to degree 4 [8]. Similarly, Tecplot has integrated high-order element visualization in its latest updates [9]. These advancements highlight the growing need for data handling solutions that efficiently manage high-order CFD data without requiring element subdivision.

To address these issues, we introduce the Computational Fluid Dynamics High-Order Writer Library (CFD-HOWL), created to perform I/O operations on high-order CFD data.

The library uses a novel algorithm to write mesh and solution data that circumvent the constraints of conventional data storage techniques. These techniques typically depend on artificial mesh subdivision and linear interpolation, which may undermine solution fidelity and inflate data storage requirements. CFD-HOWL upgrades the elements' geometric order, directly integrating high-order solution data without subdividing the elements. This upgrade ensures the mesh has sufficient nodes to accommodate the detailed solution data. Figure 1 shows the summary of this concept by comparing our method with the conventional approach. The Bassi and Rebay convention [10], i.e. $PpQq$, is used to describe the elements, where P indicates the order of the numerical method, and Q denotes the geometric order.

The novel algorithm for encoding high-order CFD data uses the CGNS (CFD General Notation System) output format. CGNS is an open-source I/O file system jointly developed by NASA and Boeing in 1995 [11, 12] to promote a

standardized data storage and retrieval method in CFD codes. The CGNS files use a hierarchical data structure, allowing for complex data organization, which is particularly useful for managing data for large and complex CFD simulations. The output files can be read by free, open-source visualization software such as ParaView and their commercial counterpart such as Tecplot. CGNS can handle high-order geometric elements (from linear, $Q1$, to quadratic, $Q4$), which is crucial for integrating high-order simulation data directly without requiring element subdivision. Users can readily apply this library *as-is* by integrating APIs that read their specific mesh formats, which may include high-order data. Alternatively, the flexible nature of the library allows for seamless integration with existing CFD solvers. The users can develop language-specific wrappers or translate the library for straightforward incorporation as a module or class to fit within their solver.

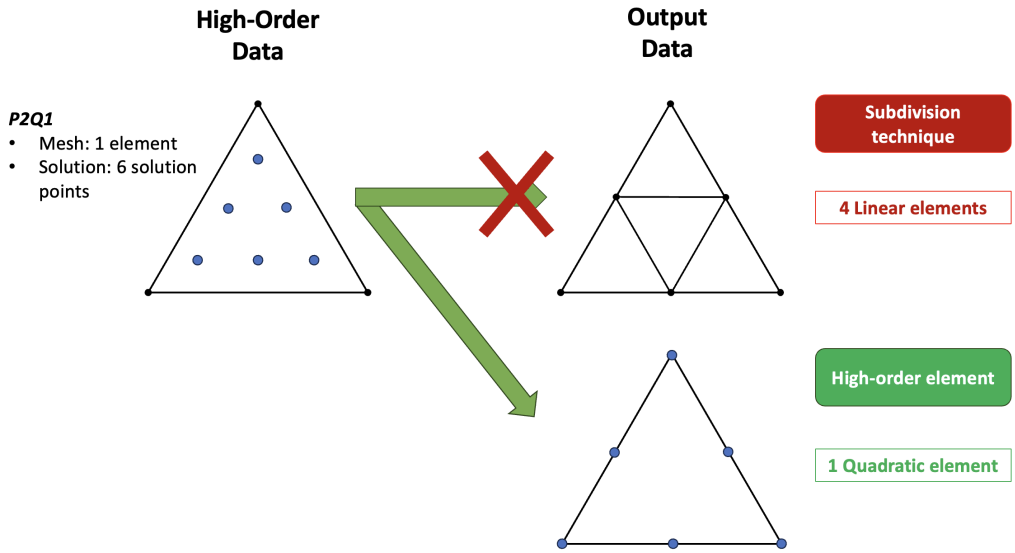


Figure 1: Conventional subdivision technique vs. geometric order upgrade of mesh elements. The example shows a $P2Q1$ triangle, i.e. a second-order polynomial (P) solution on a first-order geometric (Q) element.

2. Software description

First, we describe the structure of the library. The three folders within the root directory of the library are `writer`, which contains the essential APIs; `tests`, which contains unit tests for the library APIs; and `example`, which contains input meshes used as an example in the code to demonstrate its functionality. The `main.py` file in the root folder showcases the deployment of different APIs in the library with a given input mesh from the `example`

folder.
CFD-HOWL is centred around classes in the `writer` folder, which is briefly explained in this section. There are two types of API in this folder: ones that form the core of the library and others that are used for support and auxiliary work to demonstrate the library's application.

Nr.	**Code metadata description**	**Details**
C1	Current code version	v2.1
C2	Permanent link to code/repository used for this code version	`https://github.com/RayanDhib/CFD-HOWL`
C3	Permanent link to Reproducible Capsule	N/A.
C4	Legal Code License	GNU-LGPL 3.0
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python, C++
C7	Compilation requirements, operating environments & dependencies	pyCGNS $\leq$ v 6.6.2, CGNS $\leq$ v 4.4.0, tested in Ubuntu 23.10 and MacOS 14.6.1 .
C8	Link to developer documentation/manual	File README.md in the repository
C9	Support email for questions	rayan.dhib@kuleuven.be

Table 1: CFD-HOWL metadata.

2.1. Core APIs

Table 1 shows the metadata for CFD-HOWL. The library is written in *Python*. It depends on the open-source *pyCGNS* library that works as an interface to import the native functions from the CGNS library for writing output solution files. Other dependencies such as *numpy* and *scipy* are open source Python libraries that accelerate the I/O process. These dependencies are not required if the library is ported to C++. We now briefly describe different core APIs located in the `writer` folder.

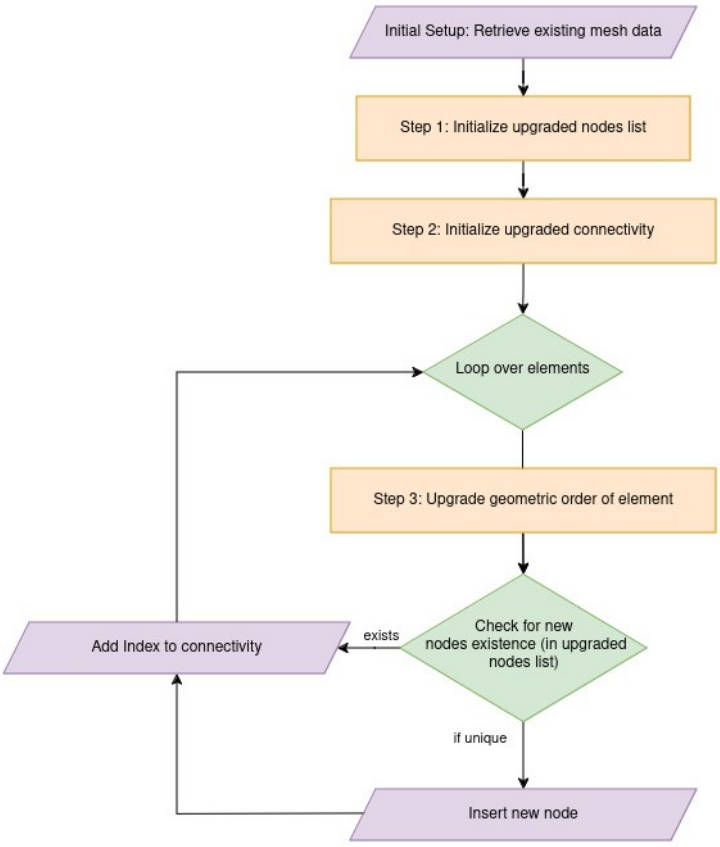


Figure 2: Flowchart depicting mesh upgrade algorithm used in the library.

2.1.1. Mesh upgrade

Figure 2 shows the flowchart logic of the novel mesh upgrade class in the library, implemented inside `writer/mesh_upgrade.py`.

In this class, the primary function, `upgrade_geoOrder`, accepts original mesh data, including nodes, connectivity, parameters specifying the current and desired geometric orders, and the element type. It begins by creating a base version of upgraded nodes and connectivity, simplifying the connectivity to fit a base geometric order as groundwork for further refinement. This function uses a *KD-tree* data structure, constructed via the `build_node_tree` function, to locate and verify the existence of nodes during the upgrade. The `compute_new_nodes_position` function calculates the positions of new nodes using geometric shape functions, which define how node positions are interpolated within each element based on the new geometric order. These positions are calculated for additional, non-corner nodes of each element,

enhancing the simulation data fidelity. Finally, the `clean_data` function is used to post-upgrade to remove any unused nodes and update connectivity indices, ensuring the upgraded mesh maintains integrity and aligns with expected topological configurations. Together, this suite of functions facilitates the conversion of a mesh to a higher resolution, which is pivotal for achieving finer, more accurate computational results in engineering and scientific analyses.

2.1.2. Mesh data

The `writer/MeshData.py` file defines the `MeshData` class, which serves as a central repository for mesh data, including nodes, connectivity, and both geometric and solution orders. This class provides methods to manage and query mesh properties. A key method in this class is `upgrade_mesh`, which integrates closely with the mesh upgrade process described in Section 2.1.1. It checks if the current geometric order of the mesh is sufficient to represent the high-order CFD solution. If an upgrade is necessary, it utilizes the `upgrade_geoOrder` and other functions from `writer/mesh_upgrade.py` to enhance the mesh resolution by refining connectivity and potentially adding more nodes. This ensures the mesh can accurately represent higher-order solutions without compromising data integrity.

2.1.3. Solution processing

The `solution_processing.py` file contains the `interpolate_to_nodes` function. This function accepts a `MeshData` object, which holds mesh specifics and a list of solution data arrays for each element, along with variable names and the number of equations corresponding to these names. The shape functions for solution interpolation are determined based on the mapped coordinates of solution points and the geometric and solution orders specified in the `MeshData` object. The `interpolate_to_nodes` function iterates over each element, retrieves the solution data, and computes the interpolated values at the nodes using the precomputed shape functions. This process involves accumulating contributions from each element's solution points to their corresponding nodes. Finally, the accumulated values are normalized by the number of contributions to each node to finalize the node-based solution data. This method ensures that the element-based solution data is accurately transformed into node-based data.

2.2. Support APIs

A brief description of support API in different files within the `writer` folder is given as follows:

- `input_data.py`: The function inside this file reads input data from a file by choosing the appropriate reader based on file extension. Currently, COOLFluiD [13, 14, 15] native *CFmesh* is supported, but the function can be extended to support other file formats;

- `cgns_operations.py`: The file includes different functions designed for working with CGNS protocol, specifically for creating, modifying, and saving CGNS data structures;

- `cgns_utils.py`: The file contains several utility functions designed to work with CGNS protocol that assists in integrating CFD data into CGNS file;

- `utils.py`: The function `compute_solShape_function` inside this file calculates solution shape functions based on their type and the solution order specified. This function transforms solution data from a computational model into a form that can be accurately represented on the mesh. It should be noted that the shape functions are normally part of the CFD solver and can be directly retrieved.

3. Software functionality and examples

We consider two examples to demonstrate CFD-HOWL deployment as an independent code and within a large CFD code such as COOLFluiD.

3.1. Standalone library

To demonstrate the deployment of CFD-HOWL as a standalone library, different APIs described in the previous section are assembled within the `main.py` script, located within the root folder. The mesh used in this script is located in `example` folder and is read using a `config` file. At first, the support APIs such as `read_input_data`, `MeshData` obtain the mesh parameters from the input mesh file, and the mesh metadata is initialized as an object. The utility APIs are then called to create a new CGNS tree populated according to mesh and solution order. The core APIs such as `upgrade_mesh` are called to perform mesh upgrade and write the output CGNS file with the appropriate order. An example of deploying CFD-HOWL as a standalone library with the open-source higher-order solver PyFR [16] is shown in Appendix C.

3.2. Coupling with a CFD solver

To demonstrate the ease of integration and porting into other programming languages, we demonstrate the integration of CFD-HOWL with our in-house, open-source Flux Reconstruction solver in the COOLFluiD platform [13, 14, 17]. The implementation can be found in the COOLFluiD GitHub repository [18] under the directory `/plugins/CGNSWriter/`, while the solver is available in `plugins/FluxReconstruction` [19, 20]. The library is ported to C++ and uses the MPI capabilities within COOLFluiD to accelerate data handling and writing. The key files of interest within the `CGNSWriter` folder are `CGNSHighOrderWriter` and `ParCGNSHighOrderWriter` for serial and parallel I/O operations. Please note that for porting in C++, the CGNS library is accessed through native `cgnslib.h` rather than using *pyCGNS* library. `ParCGNSHighOrderWriter` uses the COOLFluiD platform's parallel capabilities and the CGNS library to efficiently manage and produce a singular output file from simulations run across multiple cores. This makes the implementation ideal for modern high-performance computing (HPC) environments.

4. Impact

In this section, we evaluate the performance of the newly developed CFD-HOWL writer coupled with the COOLFluiD platform (Section 3.2). We focus on three key performance indices: how much time COOLFluiD takes to write an output file, the final size of the file on the disk, and how much time it takes to load it in post-processing software. These metrics quantify the improvements proposed by our approach under varying computational loads and complexity. Each test evaluates the performance against the existing writers in COOLFluiD for Tecplot and ParaView files. It is important to note that the conventional file writers in COOLFluiD are based on storing data using the aforementioned element subdivision method, shown in Figure 1. The *.vtu* files used in these tests are produced through subdivision, where high-order elements are split into linear sub-elements for visualization purposes. This method is different from a true high-order representation, which is available in newer *.vtu* formats. The comparison presented here highlights the benefits of using CFD-HOWL without relying on such subdivisions, directly writing high-order elements using the CGNS format. The test case used for evaluation consists of a 3-dimensional inviscid flow in a channel with a bump at Mach 0.5, which has been previously analyzed in Dhib et al. [20] using COOLFluiD's high-order FR solver. The computational domain is discretized using a mesh with $53,820$ $Q2$ prism elements, as shown in Figure 3. The library was tested on a machine equipped with the AMD

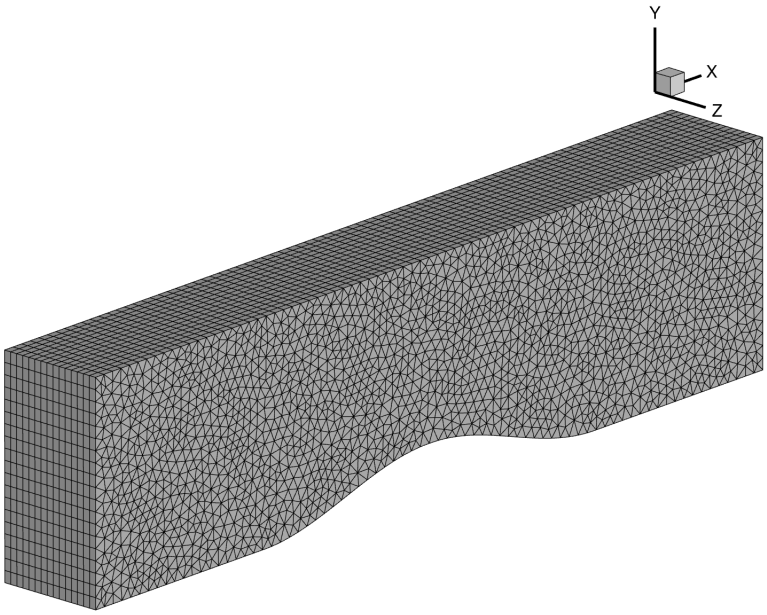
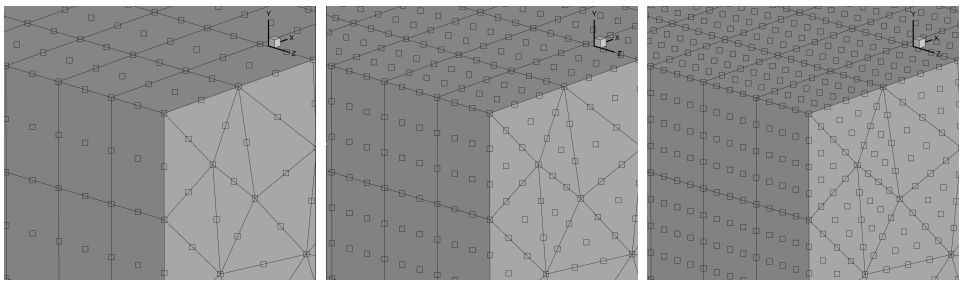


Figure 3: Mesh comprised of $53,820$ $Q2$ prism elements.

Ryzen Threadripper PRO 5995WX CPU running Ubuntu 20.04. Tecplot [21] and ParaView [22] versions 2023.2.0.42957 and 5.12, respectively, are used as the post-processing software in these tests. Please note that the results for each evaluation test presented in this section represent the average of values obtained from 10 individual runs.

4.1. Serial I/O results

Different polynomial orders ($P1$ to $P4$) for the test case are simulated on the single core of the hardware setup. Figure 4 shows the corresponding prismatic mesh. Table 2 shows the number of solution points per element and the total number of degrees of freedom for each polynomial order case. The total number of degrees of freedom is calculated by multiplying the number of solution points per element by the number of elements $(53,820)$.It is worth noting that despite the increasing polynomial order P, the number of elements written remains the same as that of CFD-HOWL. At the same time, it rapidly grows if we rely on a subdivision technique. This increase occurs because the subdivision technique involves breaking down each high-order element into multiple linear sub-elements, which inflates the total number of elements needed to represent the same data. To compare the time taken to write the file, the percentage reduction in file writing times by the solver is



(a) P2 (b) P3 (c) P4

Figure 4: Output .cgns files loaded in Tecplot displaying high-order elements for different simulations.

	P1	P2	P3	P4
Number of solution point per element	6	18	40	75
Total Number of degrees of freedom	322920	968760	2152800	4036500
Number of elements written by CFD-HOWL	53820	53820	53820	53820
Number of elements using subdivision	53820	430560	1453140	3444480

Table 2: Number of elements and degrees of freedom for each polynomial order.

calculated as follows:

$$\Delta W(\%) = \left(\frac{W_{\text{old}} - W_{\text{new}}}{W_{\text{old}}} \right) \times 100$$

Where:

- W_{old} represents the writing time using the built-in high-order writers (.plt or .vtu) within COOLFluiD;
- W_{new} denotes the writing time with the interfaced CFD-HOWL writer.
- ΔW denotes the change in file write times.

A positive value of ΔW denotes a reduction in writing times, while a negative value represents the opposite effect.

	$P1$	$P2$	$P3$	$P4$
$\Delta W\%$ wrt *.plt* writer	89.2	93.75	67.53	46.14
$\Delta W\%$ wrt *.vtu* writer	90.104	93.55	67.72	45.27

Table 3: Reduction in writing times compared to built-in algorithms.

Table 3 shows the percentage reduction in writing time for each polynomial order. A substantial reduction is obtained across all the cases. Over 80% reduction is observed for $P1$ and $P2$, and significant reductions of over 60% for $P3$ and 40% for $P4$.

	$P1$	$P2$	$P3$	$P4$
$\Delta W\%$ wrt *.plt* writer	61.69	87.27	66.04	62.99
$\Delta W\%$ wrt *.vtu* writer	74.09	86.97	59.81	53.7

Table 4: Reduction in output file size compared to built-in algorithms.

The change in output file size follows a similar analysis. The results in Table 4, show consistent reductions, with the most significant decrease of over 80% observed in $P2$ case. This is attributed to the fact that the input mesh is originally $Q2$, thus eliminating the need for mesh upgrades for the $P2$ simulations.
To further assess the impact of the CFD-HOWL writer, we measured the time taken to load the files created using element sub-division by Tecplot and ParaView for *.plt* and *.vtu* files, respectively, and compared it with our

.cgns files. Tables 5 and 6 present the reduction in loading times across different polynomial orders. In these tables, the reduction factor is calculated as the ratio of writing times of the files produced by various built-in plug-ins in COOLFluiD (CF) to the CGNS files produced by CFD-HOWL. We observe that the reduction in loading times is directly proportional to the order, reaching up to 28 times faster for $P4$ in Tecplot and over 400 times in ParaView. This demonstrates a significant improvement in post-processing efficiency for higher-order simulations.

	$P1$	$P2$	$P3$	$P4$
CF built-in Tecplot Writer (*.plt*)	6.546	11.128	19.805	34.380
CFD-HOWL (*.cgns*)	1.099	0.986	1.036	1.210
Reduction factor	5.957	11.288	19.121	28.406

Table 5: Loading time performance in Tecplot (in seconds).

	$P1$	$P2$	$P3$	$P4$
CF built-in ParaView Writer (*.vtu)*	4.591	23.153	73.639	172.859
CFD-HOWL (.cgns)	0.241	0.251	0.349	0.362
Reduction factor	19.083	92.243	210.880	477.510

Table 6: Loading time performance in ParaView (in seconds).

4.2. Parallel I/O results

CFD simulations are usually carried out with multiple cores using MPI in an HPC environment, which affects the file I/O operations. The CFD-HOWL writer, integrated within COOLFluiD, utilizes its MPI capabilities and makes use of the I/O capabilities of the CGNS library to produce a single output file, irrespective of the number of cores employed. In this section, we evaluate the influence on the parallel file writing and loading operations as the number of CPU cores increases. It is worth noting that the reduction in file size remains consistently independent from the number of cores used, as the total data volume to be written remains consistent across the same simulation with different CPU core counts.

We compared the performance of the CFD-HOWL writer to the traditional *.vtu* and *.plt* writers as a function of the number of cores. While the CFD-HOWL writer effectively reduces writing times across all CPU core counts, the trend slightly decreases with increased no. of cores, as illustrated in Table 7 for the $P4$ case. The table presents the percent change in writing times compared to the built-in *.plt* and *.vtu* writers, given by Eq. 4.1, across different core counts.

# Cores	$\Delta W\%$ wrt *.plt* writer	$\Delta W\%$ wrt *.vtu* writer
2	29.60	27.91
4	26.97	21.16
8	21.78	21.96
16	8.10	13.06
32	15.15	18.84

Table 7: Writing time performance compared to built-in libraries.

This gradual decrease is particularly evident starting from 16 cores. It can be attributed to the overhead associated with managing a larger number of cores or the specific computational characteristics of the system at higher core counts. To investigate this behaviour further, we conducted a scaling study in an HPC environment with core counts of up to 256. In this study, we measured the writing times of the integrated version of CFD-HOWL within COOLFluiD. Although we observed weak scaling, the writing times remained consistent across all core counts, with a moderate decrease at higher core counts. The details of this study are presented in Appendix B. The results for the $P4$ case represent the most data-intense scenario, thus underscoring the robustness of the CFD-HOWL writer even under demanding conditions. The impact on file loading times varies significantly between different post-processing environments, as shown in Table 8 and 9, respectively. In Tecplot, the reduction factor in loading times notably increases as the number of cores used rises, highlighting a marked improvement. The file loading time for *.plt* files written by the conventional COOLFluiD Tecplot writer increase with the number of cores, while the file loading time for the *.cgns* files generated by the CFD-HOWL writer remains relatively stable. As a result, the reduction factor escalates from about 31.50 at 2 cores to approximately 44.54 at 32 cores.
Conversely, in ParaView, the trend in reduction factors shows a decline with increasing core counts when loading *.vtu* files compared to *.cgns* files. For 2 CPU cores, the reduction factor is 309.70, which highlights a drastic reduction in loading time due to the CFD-HOWL writer. However, this factor decreases as the number of cores increases, reaching 76.25 at 32 cores. This trend can be attributed to ParaView better handling multiple smaller *.vtu* files as each core produces its own file. On the other hand, loading times for the *.cgns* files remain stable across all the tests. Despite these varying trends, the performance improvement across all scenarios is significant, underscoring the effectiveness of the CFD-HOWL writer in enhancing loading times. Both environments demonstrate substantially faster loading of *.cgns*

files compared to other formats, reaffirming the advantages of integrating CFD-HOWL within COOLFluiD in high-performance computing settings.

# Cores	Loading time of (*.plt*) files (sec)	Loading time of (*.cgns*) file (sec)	Reduction factor (%)
2	35.282	1.120	31.498
4	37.060	1.131	32.780
8	41.247	1.195	34.528
16	45.250	1.367	33.108
32	52.966	1.189	44.544

Table 8: Tecplot loading time performance.

# Cores	Loading time of (*.vtu*) files (sec)	Loading time of (*.cgns*) file (sec)	Reduction factor (%)
2	93.591	0.302	309.698
4	55.264	0.347	159.170
8	37.511	0.313	119.727
16	27.181	0.317	85.691
32	22.913	0.301	76.250

Table 9: ParaView loading time performance.

5. Conclusions

The paper presents CFD-HOWL for I/O operations for higher-order CFD codes. The library is centred around the mesh upgrade algorithm, described in detail, along with its file and data structure. The flexible structure of CFD-HOWL allows it to be used *as-is*, or integrate with existing CFD solvers as demonstrated here by porting and coupling it with the FR solver in COOLFluiD. The performance of CFD-HOWL for I/O operations is evaluated for three key performance indices: how much time COOLFluiD takes to write an output file, what is the final size of the output file on the disk, and how much time it takes to load the file in post-processing software. From the results, it is seen that the library produces significant improvement over conventional writers on all three performance indices. CFD-HOWL reduces data size, improves I/O operations, and improves data fidelity, thus removing the key bottleneck in CFD data post-processing.

Funding Sources

The research is supported by the Research Foundation Flanders (FWO) and the Flemish Government – Department EWI, with grant number FWO G0B5823N. RD's fellowship is funded by the FWO SB PhD fellowship, grant number 1S66823N. SP acknowledges support from the projects C14/19/089 (C1 project Internal Funds KU Leuven), G0B5823N and G002523N (WEAVE) (FWO-Vlaanderen), 4000145223 (SIDC Data Exploitation (SIDEX2), ESA Prodex), and Belspo project B2/191/P1/SWiM.

References

[1] Z. Wang, K. Fidkowski, R. Abgrall, F. Bassi, D. Caraeni, A. Cary, H. Deconinck, R. Hartmann, K. Hillewaert, H. Huynh, N. Kroll, G. May, P.-O. Persson, B. van Leer, M. Visbal, High-order cfd methods: current status and perspective, International Journal for Numerical Methods in Fluids 72 (8) (2013) 811–845. doi:10.1002/fld.3767.

[2] Z. J. Wang, High-order methods for the Euler and Navier–Stokes equations on unstructured grids, Progress in Aerospace Sciences 43 (1) (2007) 1–41. doi:10.1016/j.paerosci.2007.05.001.

[3] S. K. Lele, Compact finite difference schemes with spectral-like resolution, Journal of Computational Physics 103 (1) (1992) 16–42. doi:10.1016/0021-9991(92)90324-R.

[4] C. Shu, High-order Finite Difference and Finite Volume WENO Schemes and Discontinuous Galerkin Methods for CFD, International Journal of Computational Fluid Dynamics 17 (2) (2003) 107–118. doi:10.1080/1061856031000104851.

[5] X. Deng, M. Mao, G. Tu, H. Zhang, Y. Zhang, High-Order and High Accurate CFD Methods and Their Applications for Complex Grid Problems, Communications in Computational Physics 11 (4) (2012) 1081–1102, publisher: Cambridge University Press. doi:10.4208/cicp.100510.150511s.

[6] W. H. Reed, T. Hill, Triangular mesh methods for the neutron transport equation, Los Alamos Report LA-UR-73-479 (10 1973).

[7] H. Huynh, A Reconstruction Approach to High-Order Schemes Including Discontinuous Galerkin for Diffusion, Collection of Technical Papers - 18th AIAA Computational Fluid Dynamics Conference 1 (06 2009). doi:10.2514/6.2007-4079.

[8] Kitware, Modeling arbitrary order lagrange finite elements in the visualization toolkit (2018).
URL https://shorturl.at/9VLQs

[9] I. Tecplot, High-order elements in tecplot 360 (2023).
URL https://tecplot.com/2023/06/21/hoe-in-tecplot-360/

[10] F. Bassi, S. Rebay, High-Order Accurate Discontinuous Finite Element Solution of the 2D Euler Equations, Journal of Computational Physics 138 (2) (1997) 251–285. doi:10.1006/jcph.1997.5454.

[11] CGNS, Cgns github repository (2024).
URL https://github.com/CGNS/CGNS

[12] D. Poirier, S. Allmaras, D. McCarthy, M. Smith, F. Enomoto, The cgns system, in: 29th AIAA, Fluid Dynamics Conference, 1998, p. 3007.

[13] A. Lani, N. Villedie, K. Bensassi, L. Koloszar, M. Vymazal, S. M. Yalim, M. Panesi, COOLFluiD: an open computational platform for multi-physics simulation and research, in: 21st AIAA Computational Fluid Dynamics Conference, 2013, p. 2589.

[14] V. Sharma, V. F. Giangaspero, A. Munafò, S. Poedts, A. Lani, Validation and verification of the implicit thermo-chemical non-equilibrium cfd solver in coolfluid with plato, in: AIAA SCITECH 2024 Forum, 2024, p. 2728.

[15] V. Sharma, V. F. Giangaspero, S. Poedts, A. Lani, Influence of magnetohydrodynamics configuration on aerothermodynamics during martian reentry, Physics of Fluids 36 (3) (2024).

[16] F. D. Witherden, A. M. Farrington, P. E. Vincent, PyFR: An open-source framework for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach, Computer Physics Communications 185 (11) (2014) 3028–3040. doi:https://doi.org/10.1016/j.cpc.2014.07.011.

[17] D. Kimpe, A. Lani, T. Quintino, S. Poedts, S. Vandewalle, The COOLFluiD Parallel Architecture, in: B. D. Martino, D. Kranzlmüller, J. Dongarra (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 520–527.

[18] A. Lani, Coolfluid github repository (2024).
URL https://github.com/andrealani/COOLFluiD/wiki

[19] R. Vandenhoeck, A. Lani, Implicit high-order flux reconstruction solver for high-speed compressible flows, Computer Physics Communications 242 (2019) 1–24. doi:10.1016/j.cpc.2019.04.015.

[20] R. Dhib, F. B. Ameur, R. Vandenhoeck, A. Lani, S. Poedts, Development of an implicit high-order flux reconstruction solver for high-speed flows on simplex elements, Computer Physics Communications 295 (2024) 109006. doi:10.1016/j.cpc.2023.109006.

[21] Tecplot, User manual (2024).
URL https://tecplot.azureedge.net/products/360/current/360-users-manual.pdf

[22] J. Ahrens, B. Geveci, C. Law, Paraview: An end-user tool for large data visualization, in: Visualization Handbook, 2005, pp. 717–731. doi:10.1016/B978-012387582-2/50038-1.

Appendix A. Performance analysis on hexahedral elements for higher order simulations.

The performance of CFD-HOWL for the test case in Section 4 with hexahedral mesh elements is analyzed in this appendix. All the parameters, results, and images have been explained previously and reproduced here for hexahedral elements instead of prism elements.

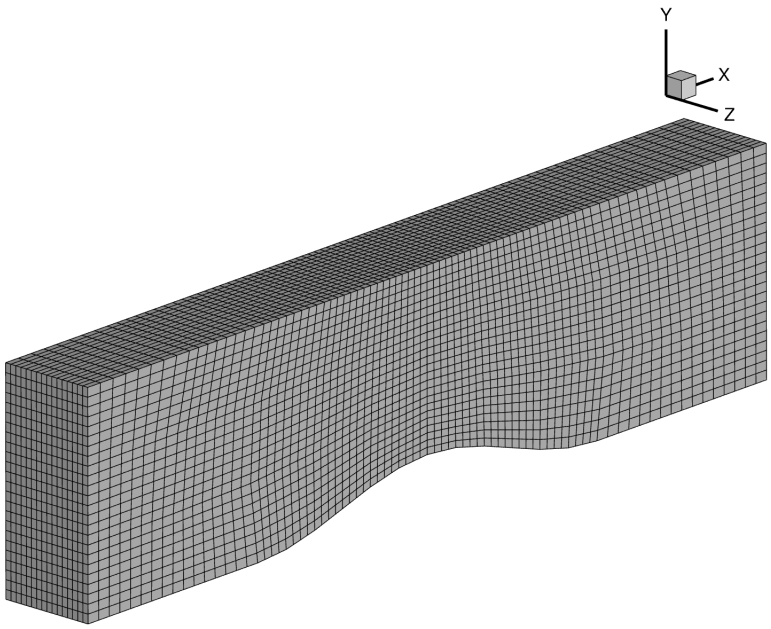
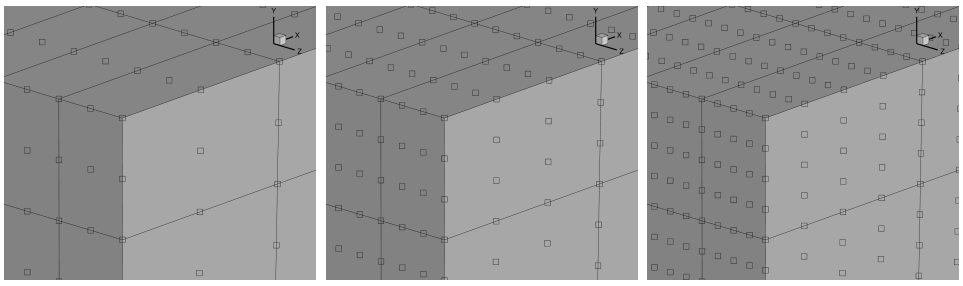


Figure A.5: Mesh comprised of $30,720$ hexahedral elements.



(a) P2 (b) P3 (c) P4

Figure A.6: .cgns files loaded in Tecplot displaying high-order elements for different simulations.

	$P1$	$P2$	$P3$	$P4$
$\Delta W\%$ wrt *.plt* writer	83.54	86.94	36.43	10.18
$\Delta W\%$ wrt *.vtu* writer	83.8	86.52	34.67	10.29

Table A.10: Reduction in writing times compared to built-in algorithms.

	$P1$	$P2$	$P3$	$P4$
$\Delta W\%$ wrt *.plt* writer	46.7	84.2	63.74	60.44
$\Delta W\%$ wrt *.vtu* writer	58.96	82.48	55.42	49.4

Table A.11: Reduction in output file size compared to built-in algorithms.

	$P1$	$P2$	$P3$	$P4$
CF Built-in Tecplot Writer (*.plt*)	4.107	7.757	15.263	27.991
CFD-HOWL (*.cgns*)	1.019	0.995	1.047	1.103
Reduction factor	4.030	7.797	14.573	25.374

Table A.12: Loading time performance in Tecplot (in seconds).

	$P1$	$P2$	$P3$	$P4$
CF Built-in ParaView Writer (*.vtu*)	2.226	8.684	25.878	59.188
CFD-HOWL (*.cgns*)	0.261	0.263	0.301	0.311
Reduction factor	8.517	33.019	86.058	190.194

Table A.13: Loading time performance in ParaView (in seconds).

Appendix B. Performance analysis on HPC

In this section, we present the additional performance results obtained from testing CFD-HOWL in an HPC environment. This study was conducted to investigate the scalability of the library with larger core counts. Therefore, the tests were carried out on a cluster with up to 256 cores. The cluster uses Intel CPUs, including Skylake and Cascade Lake Intel Xeon Gold processors. These tests focused specifically on the writing times of CFD-HOWL within COOLFluiD for a polynomial order of $P4$. The mesh used in this analysis is a refined version of the previous tests, consisting of 180,224 prism elements. The results are summarized in Table B.14.

# Cores	Writing time (sec)
2	58.149
4	37.254
8	30.755
16	26.241
32	28.026
64	28.861
128	29.370
256	33.0159

Table B.14: Writing time performance.

The results show that the writing times initially decrease, particularly as the number of cores increases from 2 to 16, but then tend to stagnate as more cores are added, remaining between 26 and 33 seconds. This pattern suggests weak scaling behavior, at least for this particular implementation of CFD-HOWL within COOLFluiD, where adding more cores does not lead to significant further reductions in writing times. It is important to note that our computational resources were constrained to a maximum of 256 cores, which prevented us from conducting further scalability tests. Despite this limitation, the current results provide valuable insights into the behavior of CFD-HOWL within COOLFluiD under increased core counts, and they suggest that the weak scaling characteristics observed in smaller setups persist even in larger environments.

Appendix C. Integration with PyFR and Comparison Results

This appendix details the integration of CFD-HOWL with PyFR [16], an open-source high-order CFD solver, and presents the results of the comparisons made between the files written by CFD-HOWL and PyFR. The goal is to demonstrate the flexibility of CFD-HOWL when used with different solvers and the efficiency of its high-order data output capabilities in terms of file size and loading times in visualization software.

Appendix C.1. Integration with PyFR

To integrate CFD-HOWL with PyFR, we updated the script `input_data.py` to read PyFR's proprietary format. Specifically, two functions were modified:

- `read_input_data`: This function extracts the nodes, element connectivity, and states from the PyFR output files (`.pyfrs`) and mesh files (`.msh`).
- `get_info_mesh`: This function extracts metadata such as the solution order, number of dimensions, and number of elements, which are necessary for setting up the high-order data writer.

Modifying these functions ensures that the library can interpret PyFR's output format and use it directly with CFD-HOWL. By making changes only to `input_data.py`, developers from other CFD solvers can quickly adapt CFD-HOWL to their solver-specific formats. `input_data.py` currently supports both COOLFluiD and PyFR formats, demonstrating the flexibility of the library.

Appendix C.2. Results

We conducted a comparison of the high-order `.cgns` files written by CFD-HOWL with the `.vtu` files produced by PyFR's native high-order writers. PyFR offers the possibility to write to high-order VTK (`.vtu`) files, as well as subdivided `.vtu` files, where elements are subdivided into linear VTK cells. This comparison focused on two key performance metrics: file size and loading times in post-processing tools.

Appendix C.2.1. Test Setup

- **Test Case**: The tests were performed using the 2D Euler Vortex case, as an example in PyFR's documentation [**?**]. We used the provided mesh for this case and refined it to create two different mesh densities for our tests.

- **Mesh Details**: The mesh used for these tests consisted of quadrilateral elements with two different mesh densities: 6.4k elements and 25.6k elements.
- **Polynomial Order**: The simulations were run at a polynomial order of P3.
- **File Formats Compared**:
  - `.cgns` **(CFD-HOWL)**: Written using CFD-HOWL, maintaining high-order data without subdivision.
  - **High-Order** `.vtu` **(PyFR)**: PyFR's native high-order writing capability using high-order VTK cells.
  - **Subdivided** `.vtu` **(PyFR)**: Traditional `.vtu` files produced by subdividing high-order elements.

Appendix C.2.2. Results and Analysis

The following table summarizes the file sizes of the different output formats:

Mesh Density	`.cgns` **(CFD-HOWL)**	**High-Order** `.vtu` **(PyFR)**	**Subdivided** `.vtu` **(PyFR)**
6.4k elements	2.7 MB	3.3 MB	4.3 MB
25.6k elements	10.8 MB	13.3 MB	17.2 MB

Table C.15: Comparison of file sizes for different formats and mesh densities.

The results indicate that the `.cgns` files written by CFD-HOWL are smaller in size compared to both the high-order and the subdivided `.vtu` files. This is attributed to the handling of high-order elements without the need for subdivision, which leads to a more compact representation of the data.

Appendix C.2.3. Loading Times in Post-Processing Tools

To assess the practicality of using different output formats for high-order data, we measured the time to load each file type into two widely used post-processing tools: Tecplot and ParaView. The measured time represents the duration of loading the data and displaying it in an already opened Tecplot or ParaView window. The results are presented separately for the two mesh sizes: $6.4k$ elements and $25.6k$ elements.

Tecplot:. For Tecplot, the `.cgns` files generated by CFD-HOWL demonstrated significantly faster loading times compared to the high-order `.vtu` files produced by PyFR for both mesh sizes. This suggests that the `.cgns` format is highly efficient for loading high-order CFD data in Tecplot.

File Type	Tecplot	ParaView
`.cgns` (CFD-HOWL)	0.04	0.17
High-Order `.vtu` (PyFR)	0.14	0.10

Table C.16: Loading Times in Post-Processing Tools for $6.4k$ Elements (in seconds)

File Type	Tecplot	ParaView
`.cgns` (CFD-HOWL)	0.11	0.54
High-Order `.vtu` (PyFR)	0.30	0.31

Table C.17: Loading Times in Post-Processing Tools for $25.6k$ Elements (in seconds)

ParaView:. In ParaView, the high-order `.vtu` files showed better performance compared to the `.cgns` files, which is expected as ParaView is inherently optimized for VTK formats. However, the performance differences were not substantial, indicating that both formats are viable options for visualizing high-order data in ParaView.

Appendix C.3. Summary

The integration of CFD-HOWL with PyFR demonstrates the library's flexibility in handling different CFD solvers. By modifying the `input_data.py` script, users can adapt CFD-HOWL to their solver-specific formats. The results from the comparisons show that the `.cgns` format produced by CFD-HOWL is efficient in terms of file size and performs competitively in loading times for post-processing, highlighting its value as a high-order data output solution.