ParCFD2024
35th International Conference on Parallel Computational Fluid Dynamics
Sep. 02-04, 2024, Bonn, Germany

A Portable Algebraic Implementation for Reliable Overnight Industrial LES

Marcial Mosqueda-Otero[a,*], Àdel Alsalti-Baldellou[a,b], Xavi Álvarez-Farré[c], Josep Plana-Riu[a], Guillem Colomer[a], Francesc Xavier Trias[a] and Asensio Oliva[a]

[a] *Universitat Politècnica de Catalunya, BarcelonaTech, Heat and Mass Transfer Technological Center, Carrer de Colom 11, 08222 Terrassa, Barcelona, Spain*
[b] *University of Padova, Department of Information Engineering, Via Giovanni Gradenigo, 6b, 35131 Padova, Italy*
[c] *SURF, High-Performance Computing Team, Science Park 140, 1098 XG Amsterdam, Netherlands*

ARTICLE INFO[†]

Keywords:
Message Passing Interface;
OpenMP;
OpenCL;
Heterogeneous Computing;
Large-Eddy Simulation;
Overnight Industrial Applications

ABSTRACT

This work assesses the feasibility of large-scale simulations for industrial applications using a symmetry-preserving discretization method for unstructured collocated grids in LES of turbulent flows. The method ensures stability without artificial dissipation and maintains portability through minimal algebraic kernels. Key challenges are addressed, such as the low arithmetic intensity of sparse linear algebra and efficient computation. A scalability analysis across MPI-only, MPI+OpenMP, and GPU architectures demonstrates the method's effectiveness in enhancing parallel efficiency and supporting large-scale simulations.

1. Introduction

The continuous development of novel numerical methods, coupled with the rapid evolution of high-performance computing (HPC) systems, has significantly expanded the role of computational fluid dynamics (CFD) in various industrial applications. Despite these advancements, the development of CFD faces persistent challenges. Early implementations were hindered by the compute-bound limitations of processors, which led to the adoption of compute-centric programming models. Over time, processor designs have evolved, addressing these limitations and resulting in a mismatch between computational power and memory bandwidth. This, in turn, forces the creation of complex memory hierarchies, complicating the optimization of traditional programs. At the same time, the widespread use of accelerators in diverse technological fields has driven the rise of hybrid architectures, offering greater computational throughput while improving power efficiency in large-scale applications [1]. However, this shift introduces a new challenge: ensuring the portability of legacy applications. This challenge requires versatile software architectures and the development of specialized APIs, such as CUDA, OpenCL, and HIP [2, 3].

In this context, the conservative discretization method for unstructured grids, as proposed by [4], has been adopted and implemented under TermoFluids Algebraic (TFA)—our in-house code based on an innovative algebra-dominant framework, HPC^2 [5]. This robust framework facilitates seamless integration into open-source codes [6] and hybrid supercomputing environments.

While computational power has improved, the time and resources required for detailed simulations remain a significant bottleneck. Achieving large-scale simulations is crucial for meeting industry demands for rapid decision-making, shorter product development cycles, and expanding CFD's industrial application fields. Thus, our research seeks to ensure the integration of modern CFD methodologies into industry practices, enabling precise and accurate simulations of complex processes while efficiently utilizing available resources and reducing simulation costs within limited timeframes.

2. Portability for CFD

The construction of codes based on a minimal set of algebraic kernels has become essential for ensuring portability, optimization, and ease of maintenance, particularly

[†] This paper is part of the ParCFD 2024 Proceedings. A recording of the presentation is available on YouTube. The DOI of this document is 10.34734/FZJ-2025-02469 and of the Proceedings 10.34734/FZJ-2025-02175.

[*] Corresponding author
marcial.francisco.mosqueda@upc.edu (M. Mosqueda-Otero); adel.alsalti@upc.edu (À. Alsalti-Baldellou); xavier.alvarezfarre@surf.nl (X. Álvarez-Farré); josep.plana.riu@upc.edu (J. Plana-Riu); guillem.colomer@upc.edu (G. Colomer); francesc.xavier.trias@upc.edu (F.X. Trias); asensio.oliva@upc.edu (A. Oliva)
ORCID(s): 0009-0003-8282-0300 (M. Mosqueda-Otero); 0000-0002-5331-4236 (À. Alsalti-Baldellou); 0000-0002-1684-7658 (X. Álvarez-Farré); 0000-0002-9086-0519 (J. Plana-Riu); 0000-0003-4592-0217 (G. Colomer); 0000-0002-5966-0703 (F.X. Trias); 0000-0002-2805-4794 (A. Oliva)

in light of the growing diversity of computational architectures and hardware vendors. The hybrid nature of modern high-performance computing (HPC) systems presents additional challenges, as effective utilization of both processors and parallel accelerators often requires heterogeneous computations and complex data exchanges. Traditional CFD codes, however, rely on intricate data structures and specialized computational routines, which complicates portability. To address this, algorithms centered on algebraic kernels, such as the sparse matrix-vector product (`SpMV`), linear combination of vectors (`axpy`), element-wise product of vectors (`axty`), and the `dot` product, emerge as promising solutions [5].

However, this approach introduces two primary challenges: (i) computational, particularly the low arithmetic intensity of the `SpMV` operation. This limitation can be mitigated by employing the more computationally intensive sparse matrix-matrix product (`SpMM`), which is advantageous in various scenarios such as matrices $\hat{\mathsf{A}} \in \mathbb{R}^{N\times N}$ that can be decomposed as the Kronecker product of a diagonal matrix, $\mathsf{C} \equiv diag(\boldsymbol{c}) \in \mathbb{R}^{K\times K}$, and a sparse matrix, $\mathsf{A} \in \mathbb{R}^{N/K\times N/K}$, *i.e.*, $\hat{\mathsf{A}} = \mathsf{C} \otimes \mathsf{A}$. The following transformation is achieved:

$$\boldsymbol{y} = \hat{\mathsf{A}}\boldsymbol{x} \implies \left(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K\right) = \mathsf{A}\left(c_1\boldsymbol{x}_1, \ldots, c_K\boldsymbol{x}_K\right), \quad (1)$$

where $\boldsymbol{x}_i, \boldsymbol{y}_i \in \mathbb{R}^{N/K}$. Substituting `SpMV` with `SpMM` in such cases significantly reduces memory access demands and the memory footprint by reusing matrix coefficients. (ii) Algorithmic challenges, such as redefining boundary conditions, can be naturally addressed within an algebraic framework using affine transformations, e.g.,

$$\boldsymbol{\varphi}_h \rightarrow \mathsf{A}\boldsymbol{\varphi}_h + \boldsymbol{b}_h, \quad (2)$$

which enables algebraic treatments suitable for both explicit and implicit time integration methods [7].

Beyond portability, algebra-based CFD implementations offer distinct numerical benefits. For example, they facilitate the development of efficient CFL-like conditions, reducing computational cost by up to 4x compared to classical approaches. The algebraic CFL method relies on constructing stable eigenvalue bounds during preprocessing, requiring only minimal vector updates over time [8]. Additionally, algebraic frameworks allow for the streamlined incorporation of advanced techniques such as flux limiters, yielding compact and efficient implementations by utilizing incidence matrices and local operations to control gradient ratios, thus reducing the number of computing kernels required for porting [9].

Building on these principles, the work in [10] proposed an effective approach to accelerate Poisson solvers by exploiting domain symmetries. By carefully ordering the unknowns, `SpMV` operations could be replaced with `SpMM`, leading to a 2.5x increase in the performance of compute-intensive kernels while significantly reducing the solver's memory footprint and setup costs. This enhancement highlights the potential of algebra-based methods for large-scale simulations on diverse HPC architectures.

3. Algorithm scalability analysis

The objective of the numerical experiments is to evaluate the performance and scalability of TFA's base algorithm under different parallel computing paradigms. Specifically, we compare the performance of an MPI-only configuration – where each CPU core is assigned a single task – against a hybrid MPI+OpenMP configuration, which utilizes 2 MPI processes and 56 multi-threaded executions per node. This hybrid approach is intended to reduce communication overhead by leveraging shared memory, which is particularly advantageous in multicore environments. Furthermore, the scalability of the code on hybrid HPC systems is analyzed, taking advantage of TFA's underlying structure, which is based on minimal algebraic kernels. This architecture enables broad portability across diverse GPU hardware using the OpenCL API.

The numerical test case solves a turbulent channel flow using a conjugate gradient solver with a Jacobi preconditioner for Poisson's equation, combined with an explicit time integration scheme and a variable time step. Since the primary objective is to assess the scalability of TFA+HPC^2 kernels, each case is limited to 10 time steps, with 800 solver iterations per step. The test problem is solved over the entire domain without exploiting symmetries, thereby isolating the key algebraic kernels – `SpMV`, `axpy`, `axty`, and `dot` product – for performance measurement and analysis.

Strong scalability tests were conducted using both MPI-only and MPI+OpenMP configurations on the MareNostrum 5 GPP supercomputer at BSC. The experiments were run on nodes equipped with two Intel Xeon Platinum 8480+ processors (56 cores, 2 GHz, 105 MB L3 cache, and 307.2 GB/s memory bandwidth) with 256 GB of RAM, interconnected via ConnectX-7 NDR200 InfiniBand. Additionally, GPU-accelerated tests were conducted on the Snellius supercomputer at SURF, utilizing nodes with two Intel Xeon Platinum 8360Y processors (36 cores, 2.4 GHz, 54 MB L3 cache, and 204.8 GB/s memory bandwidth), 512 GB of RAM and interconnected through dual ConnectX-6 HDR100 cards.

Figure 1 illustrates the strong scalability results for both parallel paradigms using two baseline configurations: (i) a single-node configuration (left plot) with a $305 \times 480 \times 350$

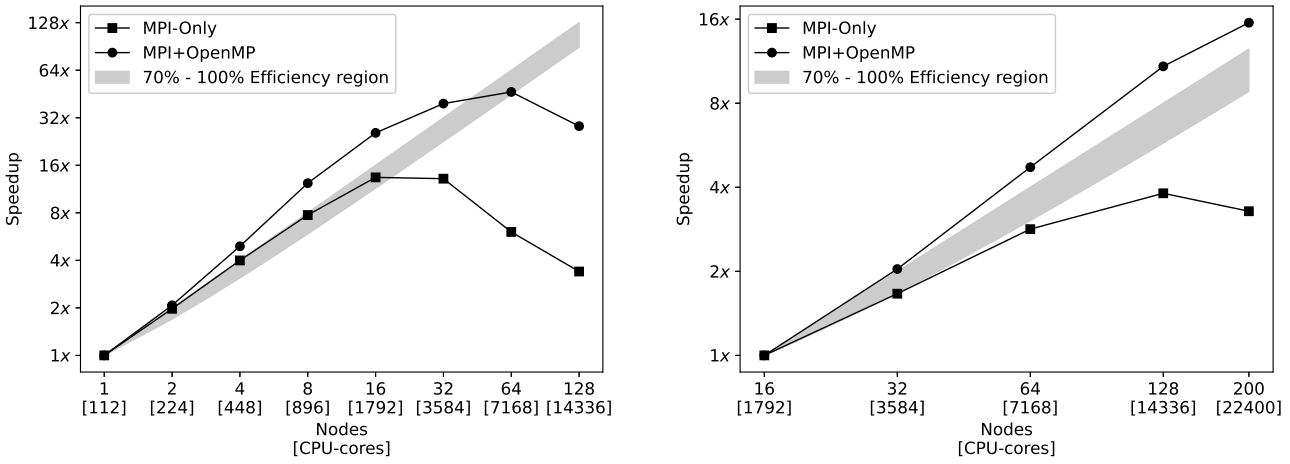

Figure 1: MPI-only vs. MPI+OpenMP strong scalability; with a 350×480×350 - 58.8M CVs - grid (left plot) and a 800×1470×800 - 940.8M CVs - grid (right plot).

grid and (ii) a 16-node configuration (right plot) with a 800× 1470×800 grid. Both configurations maintain a workload of 525k control volumes (CV) per CPU core. The results show a marked super-linear speedup for hybrid MPI+OpenMP processes, which can be attributed to enhanced cache utilization, a pattern consistently observed across both baselines.

Moreover, the MPI-only solution exhibits significant communication overhead, particularly when using 16 or more computational nodes in the single-node baseline, where a drop in performance is observed. In contrast, the 16-node baseline for the MPI-only configuration depicts a higher overhead due to the increased number of halo cells.

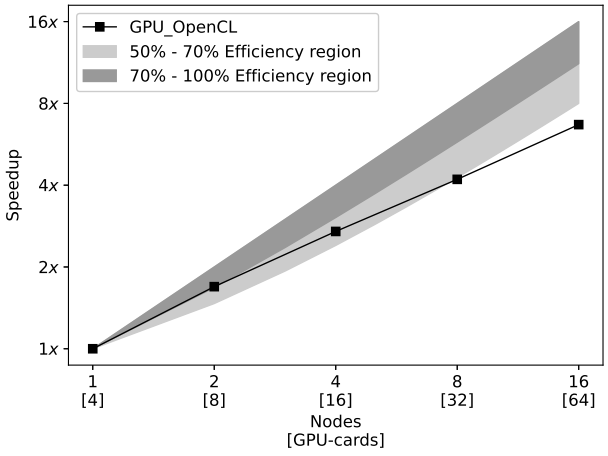

Figure 2: GPU-based strong scalability analysis on Snellius GPU island with a $400 \times 640 \times 400$ - 102.4M CVs - grid.

The strong scalability analysis for TFA+HPC^2 on a hybrid architecture was conducted on a domain of $400 \times 640 \times 400$, using a single-node configuration as the baseline on the Snellius GPU island. As shown in Fig. 2, a steady speedup is observed as the system scales. However, despite the memory-bound nature of CFD applications, the efficiency remains between 50% and 70% up to 8 nodes. For a 16-node implementation, the efficiency drops to approximately 45%, indicating the increasing impact of communication overhead in hybrid architectures as the scale increases.

Figure 3 presents the weak scalability analysis[1] for both MPI+OpenMP and GPU-based tests. The hybrid parallel paradigm (left plot) shows an 11% drop in performance when scaling up to 200 nodes, compared to the 16-node baseline. In contrast, the GPU-accelerated weak scalability (right plot) displays a significant initial drop in efficiency, approximately 10%, when transitioning from intra-node to inter-node execution. This performance decrease is attributed to the increased latency associated with inter-node communication. However, once scaling beyond two nodes, the GPU weak scaling stabilizes, showing only a minor decline in performance. At the 16-node scale, the efficiency drop remains modest at around 5%, indicating relatively consistent performance as the system scales across nodes.

[1]The main objective of this study is to assess the scalability of TFA+HPC^2 kernels. Thus, the number of iterations per time step was fixed to ensure a consistent number of kernel calls as the problem scales.

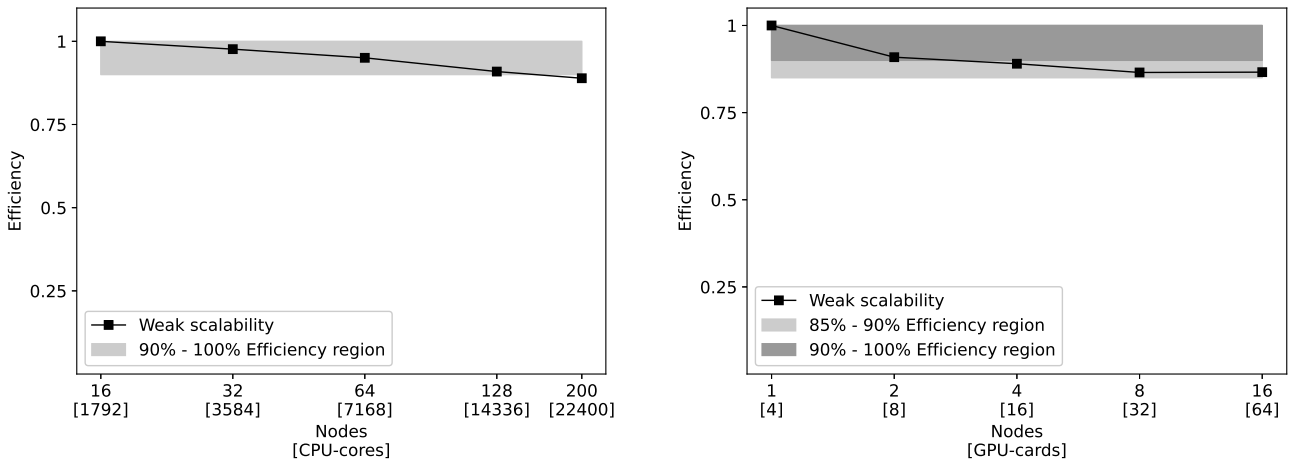

Figure 3: Weak scalability analysis; MPI+OpenMP paradigm with 525k CVs per CPU-core, starting with 16 nodes up to 200 nodes (left plot) and GPU-based implementation with 25.6M CVs per GPU card, starting with 1 node up to 16 nodes (right plot).

4. Performance analysis

In order to measure the implementation performance, an equivalent arithmetic intensity (AI_{eq}) and equivalent Performance (P_{eq}) were defined by applying a weighted average:

$$\mathrm{AI}_{eq} = \frac{\sum_{k \in K} \alpha_k \mathrm{FLOPS}_k}{\sum_{k \in K} \alpha_k \mathrm{BYTES}_k}, \tag{3}$$

and

$$\mathrm{P}_{eq} = \frac{\sum_{k \in K} \mathrm{P}_k \mathrm{N}_k}{\sum_{k \in K} \mathrm{N}_k}, \tag{4}$$

where K is the set of kernels ($K = \{\texttt{SpMV}, \texttt{axpy}, \texttt{axty}, \texttt{dot}\}$), N_k and P_k corresponds with the number of operations and the performance of each kernel, respectively, while α_k, FLOPS_k, and BYTES_k represents the operations ratio, the number of floating-point operations, and the number of memory transfers for each kernel, respectively.

Further, $\mathrm{AI}_{\texttt{SpMV}}$ is computed by the expression proposed in [10]:

$$\mathrm{AI}_{\texttt{SpMV}} = \frac{2nnz(\mathsf{A}) + 1}{8nnz(\mathsf{A}) + 4nnz(\mathsf{A}) + 4(n+1) + 8n + 8m + 8}, \tag{5}$$

where $nnz(\mathsf{A})$, n and m correspond with the number of non-zeros, 7 in the current implementation, and the number of rows and columns of matrix A, respectively.

Figure 4 illustrates the roofline model analysis of TFA + HPC^2 performance on two distinct HPC architectures, highlighting their behavior in the memory-bound region. Both solutions demonstrate an equivalent arithmetic intensity (AI_{eq}) of approximately 0.125, with a noticeable gap between the theoretical peak performance (P_{peak}) of the supercomputers and the achieved performance (P_{eq}). Despite this, the results show that TFA+HPC^2 efficiently utilizes the available resources for its given arithmetic intensity, as the performance points for both architectures are located near the memory bandwidth limit. Notably, the GPU-based implementation (right plot) exhibits higher efficiency than the MPI+OpenMP configuration (left plot).

Furthermore, while both architectures are constrained by memory bandwidth, the GPU-accelerated system consistently outperforms the MPI+OpenMP solution at lower arithmetic intensities, benefiting from higher throughput and more optimized parallel execution routines. This highlights the advantage of GPU architectures in achieving better performance despite the inherent memory-bound limitations.

5. Closing remarks

The TFA+HPC^2 framework demonstrates strong portability across various HPC architectures, leveraging its minimal algebraic kernel design to ensure broad compatibility and efficiency. The MPI+OpenMP hybrid paradigm shows superior strong scalability over MPI-only, benefiting from better cache utilization and reduced communication overhead, making it ideal for large-scale CPU-based simulations. While the GPU-accelerated implementation is promising,

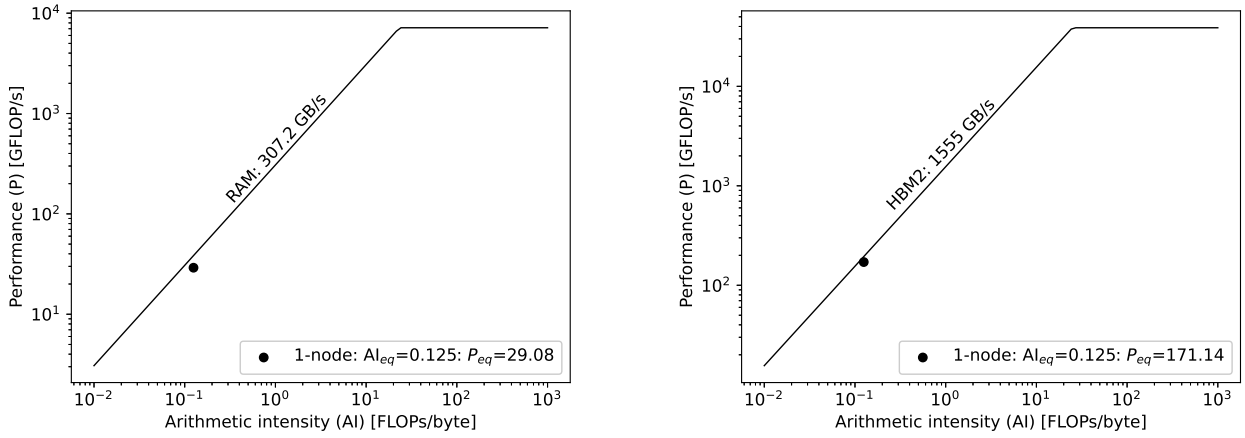

Figure 4: Roofline model; MPI+OpenMP paradigm on 1 node (112 CPU-cores) with a $350\times480\times350$ - 58.8M CVs - grid solution (left plot) and GPU-based implementation on 1 node (4 GPU cards) with a $400\times640\times400$ - 102.4M CVs - grid solution (right plot).

it faces performance limitations due to inter-node communication overhead, indicating the need for increased computational load per GPU. Nonetheless, the current implementation is highly optimized, with performance primarily constrained by memory bandwidth.

Future work will focus on increasing the arithmetic intensity of TFA+HPC^2 to overcome its memory-bound limitations, e.g., by exploiting domain symmetries in large-scale urban simulations; replacing `SpMV` operations with `SpMM` to enhance solver performance. The weak scaling analysis shows excellent efficiency, confirming the framework's robustness and scalability for demanding industrial applications.

Acknowledgements

This work is supported by the Ministerio de Economía y Competitividad, Spain, SIMEX project (PID2022-142174OB-I00). M.MO. is supported by the Catalan Agency for Management of University and Research Grants (2024 FI-1 00684). In addition, J.PR is also supported by the Catalan Agency for Management of University and Research Grants (2022 FI_B 00810). Calculations were performed on the MareNostrum 5 GPP at BSC and Snellius supercomputer at SURF. The authors thankfully acknowledge these institutions.

References

[1] 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 Communication 185 (11) (2014) 3028–3040. doi:10.1016/j.cpc.2014.07.011.

[2] H. Carter Edwards, C. R. Trott, D. Sunderland, Kokkos: Enabling manycore performance portability through polymorphic memory access patterns, Journal of Parallel and Distributed Computing 74 (12) (2014) 3202–3216. doi:10.1016/j.jpdc.2014.07.003.

[3] Y. Zhang, M. Sinclair, A. A. Chien, Improving performance portability in opencl programs, in: J. M. Kunkel, T. Ludwig, H. W. Meuer (Eds.), Supercomputing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 136–150. doi:10.1007/978-3-642-38750-0_11.

[4] F. X. Trias, O. Lehmkuhl, A. Oliva, C.D. Pérez-Segarra, R. W. C. P. Verstappen, Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids, Journal of Computational Physics 258 (2014) 246–267. doi:10.1016/j.jcp.2013.10.031.

[5] X. Álvarez Farré, A. Gorobets, F. X. Trias, R. Borrell, G. Oyarzun, HPC^2 – A fully portable algebra-based framework for heterogeneous computing. Application to CFD, Computers & Fluids 173 (2018) 285–292. doi:10.1016/j.compfluid.2018.01.034.

[6] E. Komen, J. A. Hopman, E. M. A. Frederix, F. X. Trias, R. W. C. P. Verstappen, A symmetry-preserving second-order time-accurate PISO-based method, Computers & Fluids 225 (2021) 104979. doi:10.1016/j.compfluid.2021.104979.

[7] A. Alsalti-Baldellou, G. Colomer, J. A. Hopman, X. Álvarez-Farré, A. Gorobets, F. X. Trias, C. D. Pérez-Segarra, A. Oliva, Reliable overnight industrial LES: challenges and limitations. application to CSP technologies, in: 14th International

ERCOFTAC Symposium on Engineering, Turbulence, Modelling and Measurements, European Research Community on Flow, Turbulence, and Combustion (ERCOFTAC), 2023.
URL https://www.fxtrias.com/docs/ETMM14_TF+HPC_vs_OF_abstract.pdf

[8] F. X. Trias, X. Álvarez-Farré, À. Alsalti-Baldellou, A. Gorobets, A. Oliva, An Efficient Eigenvalue Bounding Method: Cfl Condition Revisited (2023). doi:10.2139/ssrn.4353590.

[9] N. Valle, X. Álvarez-Farré, A. Gorobets, J. Castro, A. Oliva, F. X. Trias, On the implementation of flux limiters in algebraic frameworks, Computer Physics Communications 271 (2022) 108230. doi:10.1016/j.cpc.2021.108230.

[10] A. Alsalti-Baldellou, X. Álvarez-Farré, F. X. Trias, A. Oliva, Exploiting spatial symmetries for solving poisson's equation, Journal of Computational Physics 486 (2023) 112133. doi:10.1016/j.jcp.2023.112133.