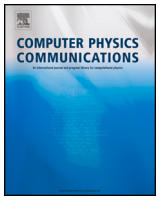
Computer Physics Communications 303 (2024) 109285

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc



Computer Programs in Physics

URANOS-2.0: Improved performance, enhanced portability, and model extension towards exascale computing of high-speed engineering flows ☆,☆☆

Francesco De Vanna [a,*], Giacomo Baldan [b]

[a] Department of Industrial Engineering, Università degli Studi di Padova, Padua 35131, Via Venezia 1, Italy
[b] Department of Computer Science, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm SE-10044, Sweden

ARTICLE INFO

Keywords:
GPU
OpenACC
Compressible flows
DNS
LES
WMLES
Open-source

ABSTRACT

We present URANOS-2.0, the second major release of our massively parallel, GPU-accelerated solver for compressible wall flow applications. This latest version represents a significant leap forward in our initial tool, which was launched in 2023 (De Vanna et al. [1]), and has been specifically optimized to take full advantage of the opportunities offered by the cutting-edge pre-exascale architectures available within the EuroHPC JU. In particular, URANOS-2.0 emphasizes portability and compatibility improvements with the two top-ranked supercomputing architectures in Europe: LUMI and Leonardo. These systems utilize different GPU architectures, AMD and NVIDIA, respectively, which necessitates extensive efforts to ensure seamless usability across their distinct structures. In pursuit of this objective, the current release adheres to the OpenACC standard. This choice not only facilitates efficient utilization of the full potential inherent in these extensive GPU-based architectures but also upholds the principles of vendor neutrality, a distinctive characteristic of URANOS solvers in the CFD solvers' panorama. However, the URANOS-2.0 version goes beyond the goals of improving usability and portability; it introduces performance enhancements and restructures the most demanding computational kernels. This translates into a 2× speedup over the same architecture. In addition to its enhanced single-GPU performance, the present solver release demonstrates very good scalability in multi-GPU environments. URANOS-2.0, in fact, achieves strong scaling efficiencies of over 80% across 64 compute nodes (256 GPUs) for both LUMI and Leonardo. Furthermore, its weak scaling efficiencies reach approximately 95% and 90% on LUMI and Leonardo, respectively, when up to 256 nodes (1024 GPUs) are considered. These significant performance advancements position URANOS-2.0 as a state-of-the-art supercomputing platform tailored for compressible wall turbulence applications, establishing the solver as an integrated tool for various aerospace and energy engineering applications, which can span from direct numerical simulations, wall-resolved large eddy simulations, up to most recent wall-modeled large eddy simulations.

Program summary

Program title: Unsteady Robust All-around Navier-StOkes Solver (URANOS)
CPC Library link to program files: https://doi.org/10.17632/pw5hshn9k6.2
Developer's repository link: https://github.com/uranos-gpu/uranos-gpu, https://github.com/uranos-gpu/uranos-gpu/tree/v2.0
Licensing provisions: BSD License 2.0
Programming language: Modern Fortran, OpenACC, MPI
Nature of problem: Solving the compressible Navier-Stokes equations in a three-dimensional Cartesian framework.
Solution method: Convective terms are treated with high-resolution shock-capturing schemes. The system dynamics is advanced in time with a three-stage Runge-Kutta method. Parallelization adopts MPI+OpenACC.

☆ The review of this paper was arranged by Prof. Weigel Martin.
☆☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: francesco.devanna@unipd.it (F. De Vanna).

https://doi.org/10.1016/j.cpc.2024.109285
Received 17 January 2024; Received in revised form 4 June 2024; Accepted 17 June 2024
Available online 21 June 2024

Contents

1. Introduction

In recent years, Computational Fluid Dynamics (CFD) has emerged as a crucial tool in examining, analyzing, and designing fluid devices and machinery for various engineering applications. Undeniably, the contemporary engineering design process relies heavily on CFD methods, resulting in a significant reduction in costs associated with expensive experimental tests and a notable decrease in the time-to-market for the most innovative and outstanding technical solutions in fluid engineering. Furthermore, CFD has enabled a deeper understanding of several fluid mechanics phenomena related to the internal and external aerodynamics of increasingly complex engineering devices, making progress induced by the use of CFD techniques not only linked to economic factors but also to physical aspects that enhance our understanding of the intrinsic complexities associated with the fluid motion. Two essential processes highlight the indispensable role of CFD methods in engineering: what we can address as a *parametric approach*, which involves repeatedly deploying CFD models across a wide range of design variables in order to determine the best design of a specific engineering device or the role of some physical variable concerning the device functioning; and what we can address as a *physical-analytical approach*, which entails a thorough examination of flow fields around specific engineering devices through a limited number of analyses but with high accuracy and precision. The distinction between these two approaches is often blurred, and in both cases, the computational load associated with CFD analyses rapidly increases with the number of variables to be traced or the accuracy demanded. Thus, passing through HPC infrastructures and advanced calculation tools is a next door issue, becoming not just a preference but a necessity to navigate the complexities and push the boundaries of what is possible in fluid dynamics simulations within contemporary engineering design and prototyping.

Thus, the drive for continuous improvement in computational models and the utilization of progressively more accurate computing tools is an enduring reality, extending beyond engineering fluid dynamics to various applied sciences sectors. Consequently, over the past decades, there has been a global shift towards concentrating computing power in specialized and dedicated centers, aiming to provide cutting-edge facilities to the international research community. A recent milestone in this trajectory is the achievement of supercomputers capable of transcending the exascale threshold, i.e., performing more than 10^{18} floating-point operations per second. This limit, as indicated by the Top500 ranking,[1] has currently been surpassed by Frontier and Aurora clusters, both located in the United States, marking a significant advancement for science and reflecting the relentless pursuit of computational capabilities on a global scale. While the United States currently leads the supercomputing market, Europe stands out with commendable representation, boasting three pre-exascale supercomputing machines ranked over the top 10 most powerful computing units globally: LUMI (CSC, Finland) holding the fifth position, Leonardo (CINECA, Italy) in sixth place, and Marenostrum 5 (BSC-CNS, Spain) holding the eighth position. This robust European presence over the top-rated architectures highlights the Old Continent's current prowess in HPC and is a compelling testament to Europe's unwavering commitment to further advancing and investing in supercomputing capabilities in the future.

Inside such a landscape, it becomes evident that recent advancements in HPC present a unique opportunity for computational initiatives in engineering design. However, a substantial challenge emerges in the requisite specialized skills and technical prerequisites necessary to unlock the full potential of HPC resources. Primarily, developing and managing solvers and software adept at effectively capitalizing on the capabilities proffered by the HPC market is an imperative mandate. The absence of tailored tools capable of harnessing the complete HPC potential would confine computational engineering design practices to traditional approaches, limiting avenues for improvement and understanding phenomena that remain elusive. Additionally, even with cutting-edge tools and software proficient in fully exploiting HPC architectures, the architectural heterogeneity of the top-ranked HPC infras-

[1] The Top500 project ranks and provides detailed information on the 500 most powerful computer systems globally. Launched in 1993, the project updates the list of supercomputers biannually, offering a comprehensive snapshot of the current landscape of high-performance computing. Details can be found at https://www.top500.org.

tructures is a persistent issue. Focusing specifically on Europe and concentrating solely on data processing units without delving into details related to compilers and networks, it is noteworthy that LUMI, currently standing out as the most powerful resource, employs AMD Graphics Processing Units (GPUs). In contrast, Leonardo utilizes NVIDIA A100 cards, and Marenostrum 5 is equipped with NVIDIA H100 GPUs. Thus, each HPC facility is based on distinct architectural logic, posing challenges to portability since each software is often designed and calibrated for the architecture most commonly adopted or preferred by individual research centers, introducing complexities in cross-platform usability. To illustrate with tangible examples, particularly within the realm of CFD, substantial progress has been achieved in developing and implementing state-of-the-art solvers designed to exploit the potential of contemporary HPC facilities. Nonetheless, a common characteristic among these solvers is their optimization and customization tailored to specific architectures, notably favoring NVIDIA-based clusters and programming languages derived from CUDA. Examples of this trend include AFiD [2] and CaNS [3], both structured incompressible flow solvers designed for massively parallel Direct Numerical Simulation (DNS) of canonical flows: The former employing CUDA Fortran, while the latter using OpenACC [4] NVIDIA-tailored directives. OpenSBLI [5], instead catering to the compressible CFD community, serves as a DNS Python framework, streamlining the automated derivation of finite difference solvers for both CPUs and GPU architectures. Additionally, PyFr [6,7] and ZEFR [8] present themselves as general-purpose DNS unstructured flow solvers, incorporating high-order flux reconstruction. In such a framework we can mention also STREAmS [9] and CharLES [10], which introduce a DNS high-order paradigm based on CUDA Fortran, and, very recently, we can dispose of STREAmS-2.0 [11], that extends STREAmS-1.0 to exploit AMD architectures, HORSES3D [12] and AMFlow [13]. Hence, we can assert that the current state of cutting-edge CFD solvers available in the European landscape closely mirrors the scenario presented in 2023 when the inaugural version of the URANOS solver [1] is introduced. Despite the presence of numerous high-performance CFD solvers, these tools are frequently fine-tuned for specific architectures, with DNS solvers designed for NVIDIA clusters taking the predominant role in the field.

In this context, the URANOS project has always prioritized vendor neutrality, positioning itself as a portable CFD solver compatible with a wide range of HPC architectures. However, the initial release of the solver had limitations, as it was predominantly tuned for NVIDIA architectures, driven by resource constraints and the development context within the Italian HPC framework at the time of its development. Recognizing the dynamic nature of the HPC landscape, characterized by diverse solutions and specific coding requirements, URANOS-2.0 keeps its commitment to vendor neutrality and multi-platform compatibility through the OpenACC standard but extends its support to the two most outstanding contemporary GPU architectures, NVIDIA and AMD, with dedicated compiler support, `nvhpc` and `cray`, respectively. This adaptability allows URANOS-2.0 to navigate the diverse landscape of supercomputers available to the European community within the EuroHPC JU. However, the portability improvement due to the support of AMD cards is not the solely technical augmentation of URANOS-2.0. Our moral commitment was also to provide the community with a solver that renewed in performance. The result is a deeply refactored solver, achieving a 2× speedup compared to its precursor if tested over an equivalent architecture. Finally, URANOS-2.0 boasts a broader array of numerical schemes and options, surpassing the flexibility offered by its predecessor. As a result, the current solver release offers a wider computing landscape with enhanced numerical schemes, improved portability, and performance. Additionally, it retains its status as a solver capable of handling compressible flows using three different modeling approaches: DNS, Wall-Resolved Large-Eddy Simulation (WRLES), and Wall-Modeled LES (WMLES) strategies. These improvements have firmly established URANOS-2.0 as a leading solver for advanced CFD simulations in the contemporary European panorama.

The paper is structured as follows: Section 2 provides a comprehensive overview of the URANOS structure, along with the available options and schemes. Section 3 delves into the specifics of the portability enhancements and outlines the options offered by the URANOS makefile. Our efforts to optimize GPU performance on contemporary cutting-edge architectures, including a detailed examination of the most demanding computational kernels, are thoroughly discussed in Section 4. Section 5 offers a comparative analysis between URANOS-2.0 and URANOS-1.0, considering multiple architectures and outlining performance over the two top-ranked multi-GPU environments currently accessible to the European scientific community. Finally, Section 6 provides conclusive remarks and summarizes the key findings of this study.

2. Numerical model description

2.1. Numerical schemes description

URANOS is a high-fidelity CFD solver that was developed explicitly for wall flow applications at the Industrial Engineering Department of the University of Padova. The solver deals with the filtered compressible Navier-Stokes system of equations in a conservative formulation, and it is capable of addressing three different fluid dynamics modeling frameworks: DNS, WRLES, and WMLES. URANOS employs a high-order finite-difference framework and supports both uniform and non-uniform structured Cartesian meshes. It offers a variety of convective schemes, including a central, nominally zero-dissipative, and fully split convective Energy-Preserving (EP) method for dealing with shock-free or smooth flows; three increasingly high-order Weighted Essentially Non-Oscillatory (WENO) methods with improved spectral resolution (-Z version); two increasingly accurate low-dissipative Targeted Essentially Non-Oscillatory (TENO) schemes, and a fifth-order TENO adaptive (TENO-A) version. Shock-capturing techniques can be specifically targeted at shock/shocklet sites to restrict the introduction of numerical viscosity and reduce nonlinear operations per time step. This enables the nominally zero-dissipative EP approach to handle smooth flow zones effectively. The outcome is a set of hybrid schemes formed by integrating each WENO/TENO shock-capturing approach with the central-like EP method, resulting in the hybrid-WENO-Z/EP, hybrid-TENO/EP, and hybrid-TENO-A/EP schemes. Three potential methods are employed to determine the shock locations in advance during each iteration. These include a density-gradient-based approach, a density-jump method, and an enhanced Ducros sensors. The first two methods are recommended for situations in which the flow pattern is already known, such as in debugging operations. They assume that regions with significant density gradients or jumps indicate the presence of shocks. The improved Ducros sensor, instead, considers both local rotations and dilatations of the flow, making it particularly suitable for scenarios involving wall turbulence within shock-dominated frameworks. The management of viscous fluxes is a distinct feature of the solver, which extends the viscous terms to isolate the incompressible from compressible components. By employing a high-order finite-difference method, the concept tackles incompressible terms with a conservative approximation, thereby enhancing the prediction of highly variable viscosity/diffusivity flows. Unlike URANOS-1.0, the solver's treatment of viscous terms has been expanded to include a standard Laplacian formulation. This alternative approach yields outcomes equivalent to those of the conservative method when the diffusive coefficients exhibit smooth flow variations. Consequently, this formulation can be effortlessly integrated into DNS frameworks without encountering issues. However, it is not compatible with WMLES applications and it is advisable to refrain from using it in WRLES. Ultimately, temporal integration employs a third-order Total-Variation-Diminishing (TVD) low-storage Runge-Kutta method, which is regarded for convective-dominated problems in compressible fluid dynamics.

Table 1
Comprehensive overview of numerical schemes and modeling options in URANOS-2.0. (For interpretation of the colors in the table, the reader is referred to the web version of this article.)

	Scheme	References	Application	Ord. of accuracy
Convective terms discretization	EP	[17–20]	smooth flows	2/4/6
	WENO-Z	[21–26]	shock-dominated flows	3/5/7
	TENO	[27–31]	shock-dominated flows	5/7
	TENO-A	[32–34]	shock-dominated flows	5
	Hybrid-WENO-Z/EP	[35,36]	shock-dominated flows	3/5/7 - 2/4/6
	Hybrid-TENO/EP	-	shock-dominated flows	5/6 - 2/4/6
	Hybrid-TENO-A/EP	-	shock-dominated flows	5 - 2/4/6
Viscous terms discretization	Semi-conservative	[37]	Intensively varying diffusive properties	2/4/6
	Laplacian	[38]	Smoothly varying diffusive properties	2/4/6
Time Integration	TVD Runge-Kutta	[39]	Convective-dominated PDE	3
Shock-detection	Density Jump	-	A priori known flow patterns	2
	Density Gradient	-	A priori known flow patterns	2
	Ducros Sensor	[17,40–42]	Wall-turbulence	2/4/6
Laminar viscosity modeling	Sutherland's law	[43]	Wide temperature ranges	-
	Power law	[44]	Temperature clustered to reference	-
Turbulence modeling	Classical Smagorinsky	[45–47]	Free-shear turbulence	2/4/6
	WALE	[48]	wall turbulence	2/4/6
	Sigma	[49]	wall turbulence	2/4/6
	MXTS	[50]	wall turbulence	2/4/6
Wall modeling	TBL equations	[14,51–53]	-	2

2.2. Turbulence modeling

Regarding turbulence modeling, URANOS-2.0 follows in the footsteps of the first solver release, with no significant changes in brooding the available choices. Thus, four algebraic turbulence models are disposed of: the classical Smagorinsky model, the Wall-Adaptive Large-Eddy (WALE) model, the Sigma model, and the MiXed Time Scale (MXTS) model. Except for the Smagorinsky model, the last three are all *wall-turbulence consistent*, i.e., they give a smooth transition to the SubGrid-Scale (SGS) viscosity and diffusivity between the bulk of the flow and the wall, vanishing the turbulent parameters while automatically approaching the wall location. To cope with WMLES, URANOS-2.0 keeps using the equilibrium-based wall model by Larsson et al. [14] by determining the proper wall shear stress and heat flux across an under-resolved wall portion and feeding the information to the outer solution as a boundary condition according to De Vanna et al. [15,16]. Even though no improvements in terms of widening the turbulence modeling spectrum have been included in URANOS-2.0 concerning 1.0, the WM-LES modeling section has been refactored to increase its performance, as will be discussed later in the paper.

2.3. Numerical model extension

To concisely delineate the modeling improvements and summarize the features of URANOS-2.0, Table 1 presents the comprehensive array of numerical schemes and models incorporated in the current version of the solver. Each entry includes minimal indications of its applicative range, order of accuracy, and relevant references. Notably, the table highlights in *light blue* the additions made in comparison to URANOS-1.0. As evident, URANOS-2.0 expands the repertoire of available numerical schemes by incorporating TENO-7 for convective scheme discretization and introducing the Laplacian formulation for treating viscous terms. Beyond these additional scheme options, URANOS-2.0 offers a choice between two models for laminar flow viscosity. The first adheres to Sutherland's law, a feature inherited from URANOS-1.0, while the second aligns with a Power Law framework. The respective equations are as follows:

$$\left(\frac{\tilde{\mu}}{\mu_{ref}}\right)_S = \left(\frac{\tilde{T}}{T_{ref}}\right)^{1.5} \left(\frac{1 + S/T_{ref}}{\tilde{T}/T_{ref} + S/T_{ref}}\right) \tag{1a}$$

$$\left(\frac{\tilde{\mu}}{\mu_{ref}}\right)_P = \left(\frac{\tilde{T}}{T_{ref}}\right)^{0.75} \tag{1b}$$

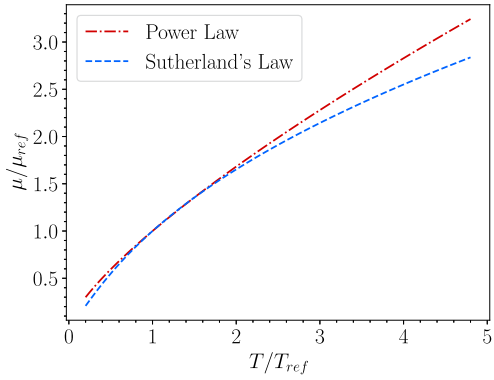
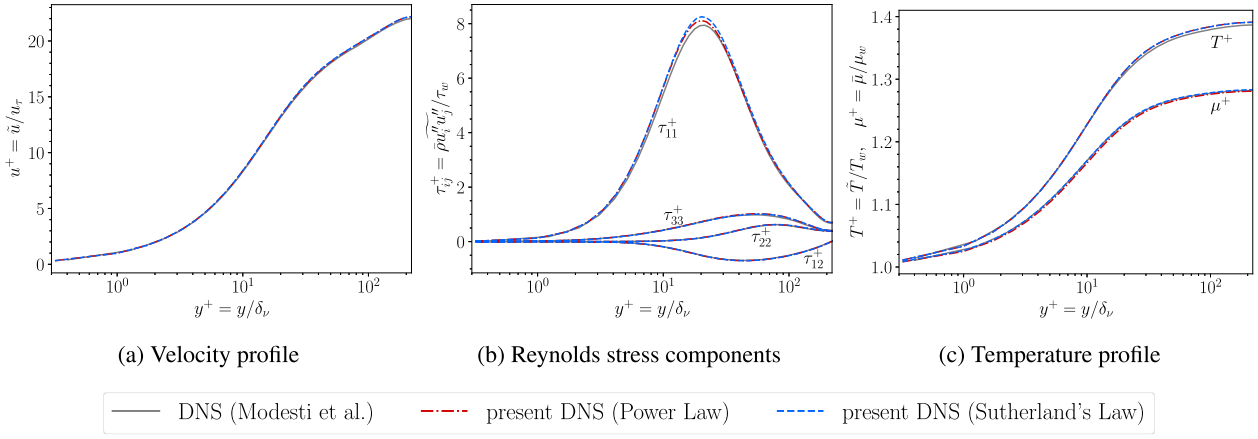


Fig. 1. Comparison between Power Law and Sutherland's Law laminar viscosity modeling as a function of the non-dimensional flow temperature.

Here μ_{ref} and T_{ref} are the fluid reference viscosity and temperature, respectively, while $S = 110.4$ K is the Sutherland's empirical fitting parameter [43]. The limitation of Eq. (1a) becomes evident in its explicit dependence on the reference temperature (T_{ref}), necessitating the expression of the S/T_{ref} ratio, even within the framework of purely non-dimensional modeling as employed by URANOS. This dependency introduces uncertainty during model settings, particularly when the reference temperature is not known. To mitigate this concern, the alternative power-law-like laminar viscosity behavior is introduced, allowing users to choose the laminar viscosity model based on specific case requirements.

Fig. 1 comprehensively compares Sutherland's law with Power law in a wide range of non-dimensional temperatures. Here T_{ref} is set to 273.15 K. The analysis reveals that the distinction between the two formulations is primarily confined to the extremal ranges, specifically for values where $T/T_{ref} < 0.5$ and $T/T_{ref} > 2.0$. Thus, within temperature clustered around $T/T_{ref} \approx 1$, the outcomes yielded by the two formulations exhibit strong similarity. This observation underscores the fidelity of both laminar viscosity models within the typical operating conditions in applications.

In order to substantiate this assertion, a fully resolved channel flow in a compressible regime is undertaken, addressing the two distinct laminar viscosity formulations. The DNS framework is deliberately chosen to serve as the exclusive avenue where the laminar viscosity formulation significantly influences the outcomes, exerting control over turbulent dissipation at the grid level. To this end, the simulation setup con-



(a) Velocity profile (b) Reynolds stress components (c) Temperature profile

Fig. 2. Comparison of turbulent channel flow DNS at $M_b = 1.5$ and $Re_\tau = 215$ modeling molecular viscosity with Power Law or Sutherland Law. Distribution of (2a) mean velocity profile, (2b) Reynolds stress components and (2c) temperature profile as a function of y^+, compared with reference DNS data of Modesti and Pirozzoli [38] (gray lines).

sists of a compressible flow confined between two isothermal walls at a bulk Mach number, $M_b = u_b/c_w$ of 1.5, and friction Reynolds number, $Re_\tau = h/\delta_\nu = 215$. Here $u_b = 1/(\rho_b V)\int_V \rho u\, dV$ is the bulk velocity, and $\rho_b = 1/V \int_V \rho dV$ is the bulk density, c_w is the speed of sound at the wall location, h denotes the channel half-width and δ_ν is the viscous length at the wall location; the latter defined as $\delta_\nu = \mu_w/(\rho_w u_\tau)$, where ρ_w and μ_w are the wall density and laminar viscosity, respectively, while $u_\tau = \sqrt{\tau_w/\rho_w}$ is the friction speed. The computation is performed within a three-dimensional domain, characterized by dimensions $L_x/h \times L_y/h \times L_z/h = 6\pi \times 2 \times 2\pi$ along the x, y, and z coordinates. In the wall-parallel directions, a uniform mesh spacing is employed, while in the wall-normal direction, a non-uniform distribution is implemented using the Gauss' error function, as outlined by De Vanna et al. [15]. The total number of nodes, denoted as $N_x \times N_y \times N_z$, is configured to be $512 \times 128 \times 256 \approx 17 \times 10^6$. The grid stretching in the Gaussian distribution is fixed at 3. The discretization process yields mesh spacings in inner units, normalized by the viscous length δ_ν, given by $\Delta x^+ \times \Delta y_w^+ \div \Delta y_c^+ \times \Delta z^+ = 7.91 \times 0.31 \div 5.88 \times 5.27$. Regarding the boundary conditions, periodicity is enforced in the wall-parallel directions, while a no-slip isothermal condition is applied to the two walls setting $T_w/T_{ref} = 1$. The grid spacing is staggered, aligning with the first and last cells such that the wall coincides with an intermediate node. The approach ensures mass conservation as discussed by Modesti and Pirozzoli [38]. For the initial condition, we follow the procedure outlined by Henningson and Kim [54]. This involves superimposing a vortex pair onto the analytical solution of the Poiseuille flow, inducing an early transition to turbulence. To sustain the flow rate within the channel, a forcing term $f_i = \{0, f, 0, 0, fu\}^T$ is incorporated into the right-hand side of the Navier-Stokes equations. The forcing term is evaluated at each time step.

Fig. 2 compares the mean streamwise inner-scaled velocity profile, $u^+ = \tilde{u}/u_\tau$, the density-scaled Reynolds stress components, $\bar{\tau}_{ij}^+ = \widetilde{\bar{\rho} u_i'' u_j''}/\tau_w$, and the inner-scaled temperature and viscosity profiles, $T^+ = \tilde{T}/T_w$, $\mu^+ = \tilde{\mu}/\mu_w$, between the two adopted laminar viscosity formulations as functions of the inner-scaled wall distance, $y^+ = y/\delta_\nu$. Upon observation, it is evident that no discernible differences exist between the two laminar viscosity formulations, as the temperature variations fall within the range of superposition for both Sutherland's and the Power law.

3. Portability improvements

3.1. Multi-GPU architectures compatibility

As outlined in the introduction, one of the critical objectives of this research was to improve the portability of the URANOS solver across the latest multi-GPU systems. Initially designed for the NVIDIA V100 architecture [1], the solver has been optimized to take full advantage of the OpenACC paradigm. As a result of this initiative, URANOS now boasts enhanced compatibility with the dominant supercomputing architectures in Europe. Specifically, the solver has been tailored for seamless integration with the LUMI cluster at CSC, Finland, which features AMD MI250X graphics cards, and the Leonardo supercomputer at CINECA, Italy, which utilizes NVIDIA A100 graphics cards. Measures have also been taken to anticipate the forthcoming release of the NVIDIA H100 architecture, which is now available for the European HPC community over Marenostrum 5 architecture. It is crucial to emphasize that both the LUMI and Leonardo architectures hold integral roles within the EuroHPC JU, ensuring accessibility for researchers and research centers affiliated with the most prominent European institutes. This widespread accessibility to state-of-the-art European architectures positions URANOS as a user-friendly and portable platform across the Old Continent. Furthermore, at the time of this writing, the two targeted architectures hold global ranks of fifth and sixth in peak performance, as validated by the Top500 ranking. However, a notable distinction arises between the LUMI-G partition and Leonardo: the former utilizes OpenACC exclusively upon compilation with the Cray compiler for Fortran codes, whereas the latter extends full support to OpenACC through the NVHPC SDK. Consequently, extensive efforts have been invested to guarantee the solver's compatibility with the Cray compiler, a consideration not addressed in the previous release. To delve deeper into these advancements, we shall explore how URANOS-2.0 navigates this diverse array of architectures.

3.2. Compiling options

The OpenACC standard does not require specific compiling options for NVIDIA and AMD products. However, integrating the solver across various architectures necessitates including explicit instructions during the pre-processing phase. Notably, distinctions are required for profiling and random number generation, resulting in two definitions, namely `-DNVIDIA` and `-DAMD`. These flags are applied during the compilation process with NVHPC and Cray compilers, respectively. The makefile underwent comprehensive restructuring to facilitate seamless transitions between architectures. As a result, the

Table 2
Comprehensive overview of the compiling options and related architectures offered by URANOS-2.0.

compiler	mode_option	Intended supported architectures	tested GPUs
nvhpc	gpu gpu_debug gpu_profiling	NVIDIA architectures	V100 [55], A100 [56], H100 [57]
cray	gpu gpu_profiling	AMD architectures	MI250X [58]
cray	cpu	Cray multi/many core architectures	-
gnu	cpu cpu_debug	Vendor neutral multi/many core architectures	-

compilation process can be executed using the following command:

```
make comp=<compiler> mode=<mode_option>
```

This approach allows users to choose the appropriate compiler based on the available architecture. Furthermore, the solver is also accessible to users who do not possess GPUs by supporting the Cray and GNU compiler on CPU modes. A resume outlining the specific details of the compilation options and intended architectures associated with each combination is provided in Table 2.

3.3. Profiling options

In addition to the differentiation between NVIDIA and AMD products, profiling options and activities are instrumental in visualizing the execution of kernels and pinpointing the most time-consuming segments of the solver. Consequently, the latest version of URANOS expands upon the profiling module, `src/profiling_module.f90`, designed as an interface for external profiling libraries. This module facilitates monitoring any code segment by enclosing it between the `StartProfRange` and `EndProfRange` calls. The resulting algorithmic appears as in Algorithm 1:

```
call StartProfRange(name)
...
! portion to be monitored
...
call EndProfRange
```

Algorithm 1. Pseudocode associated URANOS profiling interface.

Therefore, we examine the computationally intensive sections of the code, prioritizing critical routines. This process involves the utilization of the NVIDIA NVHPC SDK 23.1 [59], employing the NVTX API for NVIDIA products, and leveraging ROCTX 5.2.3 [60] to analyze code performance on AMD architectures. Profiling options are activated through user-defined compiler flags, specifically `-DNVTX` and `-DROCTX`, in conjunction with `-DNVIDIA` and `-DAMD`, while utilizing the `mode=gpu_profiling` compilation setting.

3.4. Random number generation

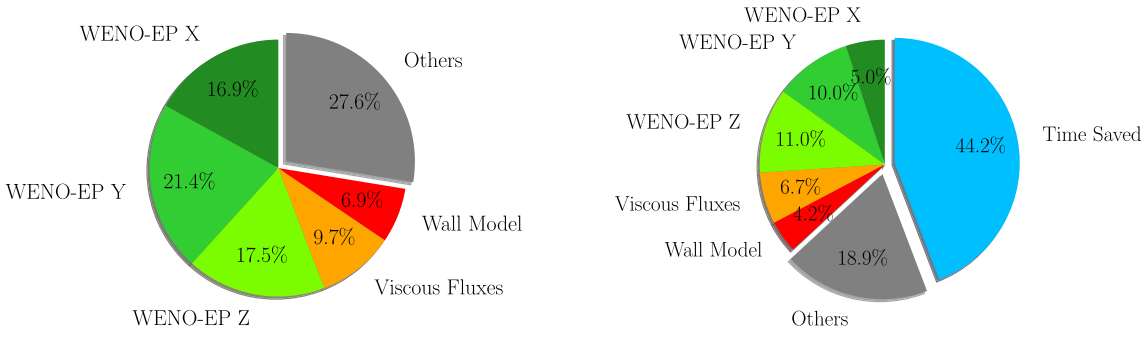
Ultimately, it is essential to emphasize a unique factor in the context of random number generation when employing URANOS with NVIDIA or AMD architectures. Although the OpenACC framework does not inherently support random number generation in a thread-safe manner, it is necessary to make specific calls to the respective CUDA (NVIDIA) or ROC (AMD) libraries, depending on the chosen architecture. More specifically, the `curand` library, integrated into the NVHPC SDK package, can be invoked during compilation using the `-cudalib=curand` flag. Conversely, random number generation suites for AMD cards are currently housed in the `rocrand` library, accessible for Fortran-based codes in LUMI by installing the `hipfort` interface through the EasyBuild suite. Once this environment is configured, the `-lhipfort-amdgcn -lrocrand` compilation flags enable the utilization of essential routines for random numbers generation. This complexity remains entirely transparent to the user, as it is exclusively managed through the compilation process governed by the makefile.

4. GPU performance optimization

4.1. Bottlenecks identification

Following the examination of the model extensions and the efforts to enhance URANOS's portability across various GPU architectures, this section provides a detailed account of the optimization process for computational kernels. The primary goal is to compare the performance of the current solver version with that of the initial release. The optimization strategy adhered to a Pareto logic, emphasizing that a select subset of functions or portions of the code significantly influences the overall calculation time per iteration. Thus, a comprehensive comparison between the old code version and the new release, particularly concerning AMD cards, was not feasible due to URANOS-1.0's lack of support for AMD cards. Consequently, the subsequent sections will present 1:1 comparisons for NVIDIA graphics cards. Simultaneously, the performance evaluation of URANOS-2.0 will cover both NVIDIA and AMD cards. This dual approach ensures a comprehensive understanding of the improvements made in the solver's performance across a diverse range of GPU architectures.

To conduct a performance analysis under computationally realistic conditions, we chose a practical scenario involving the simulation of a boundary layer in hypersonic conditions. Despite being a nominally shock-free flow, such a configuration requires the activation of shock-capturing schemes in spatial regions that are not identifiable *a priori*, introducing an additional layer of complexity to the simulation and mirroring a realistic simulation scenario where hybrid schemes (WENO/TENO + EP) operate with varied distributions in both time and space. Thus, a spatial deploying turbulent boundary layer at $M_\infty = 5.86$ over a flat plate is addressed, featuring a nominal inflowing friction Reynolds number of $Re_{\tau,0} = \delta_0/\delta_\nu = 1100$. The test has already been documented by De Vanna et al. [1] (case BL03); thus, here we provide concise details. δ_0 denotes the incoming boundary layer thickness, computed at the height where the boundary layer recovers 99% of the free-stream speed. Meanwhile, $\delta_\nu = \mu_w/(\rho_w u_\tau)$ is the viscous length. The test employs a three-dimensional box of dimensions $L_x/\delta_0 \times L_y/\delta_0 \times L_z/\delta_0 = 100 \times 18 \times 9$, mirroring a realistic spatially-deploying turbulent boundary layer. Domain discretization uses a uniform Cartesian grid distribution along the x, y, and z coordinates, ensuring a total number of computational points equal to $N_x \times N_y \times N_z = 2048 \times 256 \times 96 \simeq 50 \cdot 10^6$. This domain size is designed to sufficiently load a single H100 GPU's compute capabilities, while not surpassing the main memory limit disposed by the V100 architecture also considering the profiling overload. The A100 architecture is chosen as a reference, given its status as an intermediate architecture among the available NVIDIA products and, as previously mentioned, not disposing of an AMD-based for URANOS-1.0. As the grid is not clustered near the wall, and the number of points is insufficient to reach a WRLES at such a Reynolds regime, wall modeling is kept active, and its computational load is also assessed in the optimization process.



(a) URANOS-1.0 (b) URANOS-2.0

Fig. 3. Distribution of time per iteration percentage: Comparison between baseline and optimized version of URANOS. Data are collected over a single NVIDIA A100 architecture.

Simulations make use of the fifth-order accurate WENO scheme combined with the sixth-order EP method within the framework of a hybrid WENO-Z5/EP6 advection scheme. Diffusive terms are handled with the sixth-order semi-conservative approach, while shocks/shocklets are detected with the Ducros sensor approach. The numerical receipt aligns with a common practice in the solver standard usage. Finally, the analysis and profiling of the solver are executed leveraging the whole set of GPU architectures we dispose of, i.e., V100 [55], A100 [56], and H100 [57] from NVIDIA and MI250X [58] from AMD.

Fig. 3 illustrates the bottleneck analysis associated with URANOS-1.0 and -2.0 versions. Here, the percentage of time per iteration allocated to the primary functions is highlighted. Data are collected over a single NVIDIA A100 GPU leveraging the Leonardo supercomputing facility. Even when utilizing a single GPU, the test incorporates the complexities inherent in multi-GPU logic, given the implementation of periodic boundary conditions along the z-direction leveraging MPI calls. Performance metrics are collected by averaging one hundred iterations to ascertain statistical reliability in the obtained data. Thus, Fig. 3a provides results for URANOS-1.0, indicating that the time-intensive routines, as expected, are associated with the computation of Navier-Stokes fluxes. These encompass the three hybrid WENO/EP schemes along the Cartesian directions, the diffusive terms computation, and, in addition, the wall model kernel. Similar results and considerations emerge from employing the same analysis for the other architectures. These additional findings have not been explicitly reported here for simplicity in the text. Thus, adhering to Pareto logic, these five routines account for nearly 75% of the total time per iteration, underscoring the need to optimize such specific segments of the solver to enhance most of the code performance. Nevertheless, something should be mentioned about the gray slice, which, though not insignificant, constitutes approximately 25% of an iteration. These include an innumerable array of routines such as the time integration algorithm, updating of primitive variables, computation of the SGS terms, computation of the shock sensor, and data movements associated with MPI calls, among others. While the cumulative cost of these components is not negligible, their optimization margins prove to be modest unless a refactoring of the entire solver is put into play. Derived from these observations on the URANOS baseline, Fig. 3b delineates the distribution of time percentages within the optimized solver version. Data are here rescaled concerning the time per iteration gathered in URANOS-1.0. The rescaling approach highlights the percentage of the time saved per iteration (azure region). As evident, optimizing the solver in its version 2.0 has yielded a noteworthy advantage, manifesting in a substantially reduced calculation time. This translates to a time per iteration approximately halved compared to that recorded for version 1.0. Subsequently, in the progress of this paper, we will delve into the details of the optimization process applied to the five highlighted routines.

4.2. Optimization of advection fluxes computation

We start our discussion by delving into the optimization process for the computation of convective terms. Notably, our investigation reveals a pronounced correlation between the computation time and the specific advection component. Precisely, the computational efficiency experiences a decrease when calculating y and z advection components as opposed to the x. The trend is depicted in Fig. 3. This discrepancy stems from the nature of the hybrid WENO(TENO)+EP algorithm, which necessitates extracting one-dimensional arrays from the three-dimensional data structures in order to compute advection fluxes components. The extraction aligns with Fortran data sorting for the x fluxes components. Conversely, it experiences performance penalties with non-contiguous data in memory for y and z directions. Thus, while the operations for computing convective flows remain consistent across all directions, a substantial performance degradation is observed in streamwise-orthogonal terms. Given this observation, and to maintain conciseness in our dissertation, we present and analyze the pseudocode associated with y flux components only. Structurally and operationally, the algorithm mirrors the one employed for calculating convective flows along the z direction and just a small departure from x terms computation is kept.

Algorithm 2 illustrates the pseudocode governing the computation. The key factor contributing to a substantial enhancement in the performance of convective flux calculations is implementing chunking for local one-dimensional variables and making them available for the device's shared memory. This list of variables plays a crucial role in the flux computation process, necessitated by the imperative to extract essential values from field variables and granting the conservative nature of convective terms discretization. Thus, the optimization approach involves statically declaring all local one-dimensional variables. The primary issue consists in determining their optimal size. The latter hinges on two critical factors: An expansion in chunk size would theoretically lead to enhanced performance, attributed to the reduced overlap between chunks (the latter required for computing derivatives through a high-order finite difference approach, i.e., accounting for chunk ghosts). However, the counteractive consideration is that an excessive enlargement of chunk size results in warp/wavefront stalling, attributable to the if statement governing the WENO/TENO or EP entries, leading to an occupancy reduction. A series of numerical tests highlight that the most favorable compromise manifests with arrays size of `vec_size=64`. This carefully chosen parameter strikes a balance, maximizing performance gains while avoiding the pitfalls of excessive chunk dimensions and warp/wavefront stalling constraints. Being all one-dimensional variables required for flux computations defined statically, the `!$acc cache(list_of_variables)` directive becomes instrumental. This directive compels the compiler to allocate *cached variables* in the device shared memory (96 kB/SM for V100, 192 kB/SM for A100, and 256 kB/SM for H100, and 16 kB/CU for MI250X, where SM and CU stand for

```
subroutine PseudoCodeOfAdvectionFluxesComputation
! ...
integer , parameter                              :: vec_size  = 64
integer , parameter                              :: chunkSize = vec_size - 2*GN
real(rp), dimension(0:3,1-GN:chunkSize+GN) :: chanked_variable
! ...

!$acc parallel vector_length(vec_size) default(present) &
!$acc private(list_of_private_variables)
!$acc loop gang collapse(2)
do k   = sz,ez
  do i = sx,ex,4

     iimax = min(3,ex-i)

     do str_y = lby,uby,chunkSize
        end_y = min(str_y+ChunkSize, uby-2*GN+1) - str_y

        !$acc cache(list_of_lcl_chunked_variables)

        !$acc loop vector collapse(2)
        do j = 1-GN,end_y+GN
           do ii = 0,iimax
              jgbl = j + str_y-1+GN

              ! storing data in chunked and cached variables
              lcl_chunked_variable(ii,j,1) = gbl_variable(i+ii,jgbl,k,1)

           enddo
        enddo

        ! computing advection fluxes using lcl chunked variables
        do ii=0,iimax
           !$acc loop vector
           do j = 1-GN,end_y+GN

              if(weno_flag_chunk(j) == is_smooth) then
                ! Perform EP algorithm
              else
                ! Perform WENO/TENO algorithm
              endif

           enddo
       enddo

   enddo
enddo

return
end subroutine PseudoCodeOfAdvectionFluxesComputation
```

Algorithm 2. Pseudocode associated with advection fluxes computation.

Streaming Multiprocessor according to NVIDIA and CU for Computing Unit according to AMD), reducing L1/L2 pressure and yielding tangible enhancements in memory access and computational performance. As observed from Algorithm 2, variables segmented into chunks are represented as three-dimensional structures with an inner co-dimension of size $0:3$, a second dimension of a size $1-GN:chunkSize+GN$, and a third co-dimension packing different fields (i.e., density, velocity, pressure, and more). Here, we highlight that the x direction misses the first inner co-dimension. This additional feature is aimed for y and z flux computation only to optimize the exploitation of global data continuity along the primary direction, effectively mitigating cache miss issues and minimizing memory accesses. This distinction in data management is the sole difference between x and y/z fluxes computations. Ultimately, chunked variable implementation, even enhancing the performance, would enable computation solely for domains featuring a point count that is either less than `vec_size`=64 or a multiple thereof. To achieve full generality for domains with cardinalities not constrained by multiples of `vec_size`, minimum operations on indices are incor-

```
subroutine spread_hybrid_convective_scheme_flag
implicit none
integer :: i,j,k
integer :: bL, bR

bL = weno_num - 1
bR = weno_num

!$acc parallel default(present)
!$acc loop gang, vector collapse(3)
do k       = sz-1,ez
   do j    = sy-1,ey
      do i = sx-1,ex

         weno_flag_xyz(i,j,k) = minval(weno_flag(i-bL:i+BR,j,k))
         weno_flag_xyz(i,j,k) = weno_flag_xyz(i,j,k) + 2*minval(weno_flag(i,j-bL:j+bR,k))
         weno_flag_xyz(i,j,k) = weno_flag_xyz(i,j,k) + 4*minval(weno_flag(i,j,k-bL:k+bR))

      enddo
   enddo
enddo
!$acc end parallel

return
end subroutine spread_hybrid_convective_scheme_flag
```

Algorithm 3. Pseudocode associated with the spreading of data structure collecting shocks locations.

```
integer, parameter :: bit = ! 1,2,4 dipending on the Cartesian direction

weno_flag_chunk(i) = iand(weno_flag_xyz(igbl,j,k),bit)
! ...
if(weno_flag_chunk(k) == bit) then
        ! the flow is smooth
else
        ! the flow is shocked
endif
```

Algorithm 4. Pseudocode associated with bitwise operations associated structure collecting shocks locations.

porated, rescaling of indices within the do loops, as exemplified in lines 14 and 17 in Algorithm 2.

Our optimization efforts extended beyond the chunking methodology outlined above. In particular, to enhance the efficiency of convective flux computation and mitigate risks associated with warp/wavefront stalling, a crucial consideration in hybrid schemes involves spreading data structures containing information about shocked/non-shocked nodal positions. In URANOS-1.0, the spreading of the three-dimensional `integer(kind = 4)` structure demanded for this task, `weno_flag_xyz`, was dynamically determined during flux computation. However, this approach proved incompatible with the operational characteristics of GPUs, chiefly due to issues related to warp/wavefront stalling. Spreading such data structures is fundamental for properly working the whole algorithm since identifying shocks/shocklets' location through Ducros-like detection methods can determine solitary nodes due to the noise associated with the fields or turbulent fluctuations. This scenario poses a significant threat to simulation stability when such a singular nodal location is subsequently addressed with a shock-capturing scheme. Thus, expanding the coverage of shocked locations becomes imperative to keep simulation stability, ensuring a minimum extension equivalent to the stencil of the employed WENO/TENO scheme.

In light of this consideration, the decision is made to execute the spreading of the three-dimensional variable responsible for managing shock positions as a distinct operation from the computation of convective flux. Algorithm 3 reports the function adopted for this purpose. Notably, the kernel orchestrates a localized reduction of the variable by internally defining its minimum value within a stencil of size `bL:bR` - recall that a cell flagged as shocked corresponds to a value of zero in the `weno_flag_xyz` variable, whereas a cell marked by a smooth flow is represented by a value of 1. Thus, the `bL` and `bR` parameters are contingent upon the order of accuracy and, consequently, on the stencil necessary for computing the WENO/TENO schemes. To achieve the localized reduction, the intrinsic Fortran function `minval` is employed. Lastly, it is noteworthy that the summation of the `weno_flag_xyz` in the y and z directions is organized by multiplying its values by 2 and by 4 along the y-axis and z-axis, respectively. This aspect constitutes a secondary optimization to minimize the field variables required to store the shock-affected nodal locations. Finally, in the convective flux calculation, we implemented a bitwise flag selection procedure to selectively define the chunked variable associated with `weno_flag_xyz`. Algorithm 4 provided the code excerpts for this implementation.

4.3. Optimization of viscous fluxes computation

According to the outcomes highlighted in the bottlenecks investigation (see Fig. 3), the second optimization objective focused on enhancing the viscous terms kernel. Viscous fluxes rank as the fourth function experiencing the lengthiest time per iteration. However, their computational structure significantly differs from that employed for convective contributions. Specifically, the viscous terms involve a singular three-dimensional loop that accesses several `i,j,k` structured data and computes the necessary derivatives of fluid quantities via finite difference operations. Drawing from the insights gained while optimizing convective terms, employing two or three-dimensional chunks for such computations would be beneficial. This approach would involve defining static `nx,ny,nz` data structures to leverage the potential offered by the `!$acc cache` directive, hoping for a benefit comparable to that observed for the computation of convective terms. Given that the kernel operates purely in three dimensions, the chunking approach can be automatically performed by the compiler, leveraging the `tile` option of the `!$acc loop` directive over the kernel loop, i.e., according to the following syntax:

```
!$acc loop gang, vector tile(nx,ny,nz)
```

Specifically, the `tile` option instructs the compiler to handle two- or three-dimensional data structures through chunks of size $nx \times ny \times nz$, replacing the `collapse(3)` option and, in specific scenarios, yielding enhanced performance [61]. Numerous attempts have been undertaken to identify the optimal chunk sizes for the `tile` option. The exploration involved testing both three-dimensional and two-dimensional combinations. Unfortunately, performance deterioration is observed in all configurations compared to the baseline version. This outcome can be clarified by considering the following rationale: in loops with a three-dimensional structure, the chunks' overlap - mandatory to ensure finite difference operations - becomes non-negligible as the chunk size decreases. Consequently, small chunks lead to performance degradation. On the contrary, relative to the inner-chunk computational nodes, the percentage of chunk ghosts rapidly diminishes when working with large chunks. However, large chunks cannot be fully accommodated in the shared memory, resulting in performance degradation due to L1/L2 cache misses. Tests revealed the absence of a viable compromise, and the `tile`-based strategy implementation exhibited a performance reduction compared to the `collapse(3)` baseline approach.

Given the limited storage capacity of shared memory, another path is followed. In particular, we tried to define small arrays buffer - dynamically extracted during the loop from three-dimensional variables - that encapsulate the essential values required for finite difference operations. Such arrays, effectively collecting the field values necessary for a computational stencil and being statically declared, can be directed to the shared memory using the cache directive according to the following instructions:

```
real(rp), dimension(-bL:bR) :: uxbfr
do k,j,i
   !$acc cache(uxbfr)
   uxbfr = U(i-bL:i+bR,j,k)
```

This implementation, too, failed to yield overall benefits. While the loops dedicated to finite difference calculations experienced a significant reduction in computational cost, the buffers' extraction for all variables across the three Cartesian directions nullified this advantage. Consequently, the overall kernel computational cost remained comparable to the baseline solution. From these attempts, it has been concluded that the kernel responsible for computing viscous terms is inherently memory-bounded, providing minimal room for optimization through less intrusive refactoring actions. Attempts with straightforward combinations of OpenACC optimization directives have not yielded appreciable improvements. Hence, we chose a more conventional optimization paradigm by minimizing division operations, and reusing variables to stress the registers. These precautions contributed to the noteworthy results already depicted in Fig. 3b, which we will discuss in detail in the following of this paper.

```
subroutine ComputeWallModeling

use wmles_module, only: OdeWMLES

! looping over a wall

!$acc parallel default(present)
!$acc loop gang, vector collapse(2)
do     k = sz,ez
   do  i = sx,ex

       inputs = ! i,k dependent

       call OdeWMLES(inputs,tauWall,qWall)

   enddo
enddo
!$acc end parallel

end subroutine ComputeWallModeling
```

Algorithm 5. Pseudocode associated the wall modeling computation.

4.4. Optimization of the wall model computation

Finally, substantial revisions are applied to the wall modeling kernel. Unlike previous sections, the refactoring approach did not adhere to specific strategies since the primary factor contributing to the computational time of wall modeling is closely tied to the iterative solution of the thin boundary layer equations. The solution of these translates into solving two non-linear systems of equations — one for momentum and one for energy conservation within the boundary layer — in a segregated manner. The process is genuinely serial for each wall location, and it is demanded to the `OdeWMLES` subroutine as sketched in Algorithm 5. Given this implementation, the parallelization granularity is exhausted by distributing threads over the `OdeWMLES` calls. Consequently, the subroutine is subjected to serial operations. Thus, in pursuit of enhancing this kernel, the refactoring primarily concentrates on mitigating bottlenecks and minimizing idle operations of the `OdeWMLES` routine. The following measures are specifically implemented:

1. *Removal of idle division and exponential operations.* The WMLES interface is dynamically established, considering local resolution and flow characteristics. Thus, the grid required for discretizing thin boundary layer equations is computed point by point along the wall-normal direction. Specifically, the grid adopts a geometric progression distribution of nodes, necessitating iteration for calculating the ad-hoc stretching factor. In URANOS-1.0, this process involved resource-intensive iterative steps employing multiple exponential operations. In the solver's current version, we have actively worked to streamline and reduce such operations.
2. *Function calls inlining.* Within the `OdeWMLES` subroutine, iterative calls to the functions solving the tri-diagonal system of equations arise from the second-order finite volume method discretization of thin boundary layer equations. These processes utilize a standard Thomas Algorithm, now condensed into a few embedded lines.
3. *Elimination of idle outputs.* Finally, in URANOS-1.0, the `OdeWMLES` function generated velocity (`u_wm`) and temperature (`T_wm`) profiles derived from the solution of thin boundary layer equations.

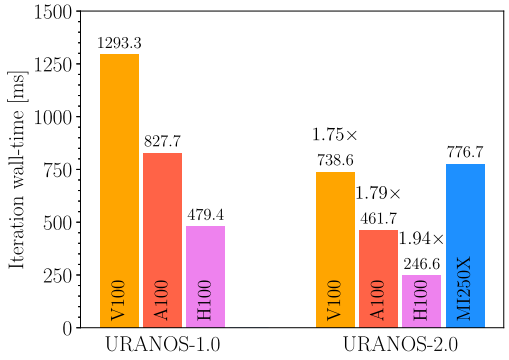


Fig. 4. Time per iteration for URANOS-1.0 and 2.0 using different GPU architectures. For the AMD MI250X, tests are run on a single GCD and reported time are halved to consider the potential of a full GPU.

This required declaring `u_wm` and `T_wm` arrays as `private` structures within the `$acc loop`, creating individual copies for each GPU thread. Since `u_wm` and `T_wm` are solely utilized for determining wall friction and heat fluxes, `OdeWMLES` in URANOS-2.0 exclusively outputs the wall shear stress and heat flux scalar values. This design eliminates the need for private copies of `u_wm` and `T_wm`, streamlining the computational process.

5. GPU performance comparison

5.1. Global time per iteration comparison

After examining the methodologies employed to optimize the code's computationally intensive sections, this section presents the performance speedup achieved by URANOS-2.0 compared to its initial release. The focus is mainly on evaluating how the refactored code performs across various cutting-edge GPGPU architectures, including the NVIDIA V100, A100, and H100, as well as the AMD MI250X. Performance evaluations are conducted in a realistic flow scenario using the previously discussed test case involving the 50 million points hypersonic boundary layer. For NVIDIA cards, simulations are carried out on a single GPU; however, a different approach is warranted for the MI250X card, which features two separate Graphics Compute Die (GCD). Relying on a single MPI rank may be limited in assessing the performance of this architecture. Therefore, the MI250X architecture is tested with one and two MPI ranks sharing the same GPU. Only one MPI rank is utilized for profiling activities to compare with the NVIDIA profiling process and avoid the complexities of simultaneously profiling two MPI ranks. The obtained times are halved to account for the scaling factor, considering an ideal linear scaling between one and two processes. For completeness, the linear scaling is verified by running the same test over two MPI ranks, namely the whole MI250X architecture, and a difference of less than 1% of the time-per-iteration is observed compared to single-rank execution. Finally, it is worth noting that using a single MPI rank does not cause a loss in generality as it introduces the complexities related to MPI data exchange due to the management of periodic boundary conditions.

Fig. 4 presents the time per iteration as a function of the solver version and chosen architecture. The left-hand-side columns depict the results for URANOS-1.0, whereas the right-hand-side bars report the outcomes for URANOS-2.0. Focusing on the performance obtained on NVIDIA cards, the time per iteration demonstrates nearly linear scaling with technological advancements, regardless of the solver version. For instance, transitioning from URANOS-1.0 on V100, where the time per iteration stands at 1293.3 ms, to URANOS-1.0 on H100 yields a 2.7× speedup solely attributable to technological advancements. This observation underscores the consistent impact of technological progress on the computational efficiency of URANOS across various present and, hopefully, upcoming platforms. In the rightmost set of columns, the times per iteration for the enhanced version of URANOS are presented. Specifically, compared to URANOS-1.0, there is a 1.75-fold acceleration on V100, a 1.79-fold improvement on A100, and a 1.94-fold increase in H100. Thus, it is noteworthy to bring to the reader that the computation of 50 million fluid points utilizing high-order schemes on a single GPU has experienced an intense acceleration if we look at the whole picture. This goal results from a synergy between technological advancements and optimization efforts. Specifically, the time per iteration decreased from 1293.3 ms in URANOS-1.0 on the V100 to 246.6 ms in URANOS-2.0 on the H100. This improvement amounted to an overall speedup of 5.24x. It is essential to mention that data for AMD cards is not included for URANOS-1.0, as the earlier version of the solver did not support these architectures. Thus, a historical trend cannot be traced for AMD products. Nevertheless, it is noteworthy that the performances of URANOS-2.0 on AMD cards are comparable to those attainable on an NVIDIA V100 GPU.

5.2. Refactored routines performance comparison

Once a specific technology is fixed, the improvement lies exclusively in the optimization process applied to the solver, leveraging the strategies detailed in the preceding paragraphs. Thus, we thoroughly examined the performance of the five functions subjected to the refactoring process. Fig. 5 analyzes the time per iteration for key subroutines, including WENO-EP-X/Y/Z, viscous fluxes, and the wall model kernel. The data is presented with respect to different solver versions and adopted technologies. Notably, the consistent 2× improvement in overall calculation time remains evident across various platforms, encompassing V100, A100, and H100 architectures. However, as discussed in Section 5.1, the overall speedup per iteration falls slightly short of this factor. This discrepancy is attributed to unoptimized sections within URANOS, which, while marginally contributing to the total computing time, do not significantly diminish the gains achieved by optimizing the five highlighted routines. Turning attention to the A100 architecture, while similar observations apply to all NVIDIA and AMD cards, a notable distinction arises in the WENO-EP X and WENO-EP Y/Z computations. Specifically, WENO-EP Y/Z takes approximately 30% longer than its counterpart in the x direction, a disparity stemming from the distinct data access model along the y and z directions, necessitating access to device memory in a way misaligned with Fortran's data storage. This dissimilarity persists even in the optimized scenario (URANOS-2.0). Notably, WENO-EP X achieves a $\approx 3\times$ speedup, contrasting with the consistently around 2× speedup observed in the other advection flux computations. This emphasizes the tangible impact of the optimization process in addressing challenges associated with data access models. Instead, the performance comparison related to viscous terms follows a different pattern, exhibiting a consistent twofold increase in speed across the three NVIDIA architectures during the transition from URANOS-1.0 to URANOS-2.0. In contrast, their impact appears notably persistent on the AMD architecture, with a time per iteration comparable to URANOS-1.0 on the V100 card. However, the true divergence in performance across architectures becomes evident in the computation of the WMLES kernel. Following optimization efforts, the time per iteration for this kernel significantly decreases, from 45.58 ms/iter in URANOS-1.0 on NVIDIA V100 cards to 5.21 ms/iter in URANOS-2.0 on NVIDIA H100 cards. In contrast, the same kernel requires 19.52 ms/iter on AMD MI250X, a value closely aligned with the performance of URANOS-2.0 on V100 cards. Considering the time needed for calculating viscous flows and the latter performance obtained over the WMLES kernel, it becomes apparent that the overall time per iteration of URANOS-2.0 on an AMD card is on par with the performance on a V100 card, as observed in Section 5.1.

Let us delve into data reading and writing associated with the five routines under investigation. For our subsequent analyses, we will focus specifically on the device's main memory data movement, which is of paramount importance compared to other memory segments. Fig. 6 provides a detailed overview of the data movement (readings and writ-

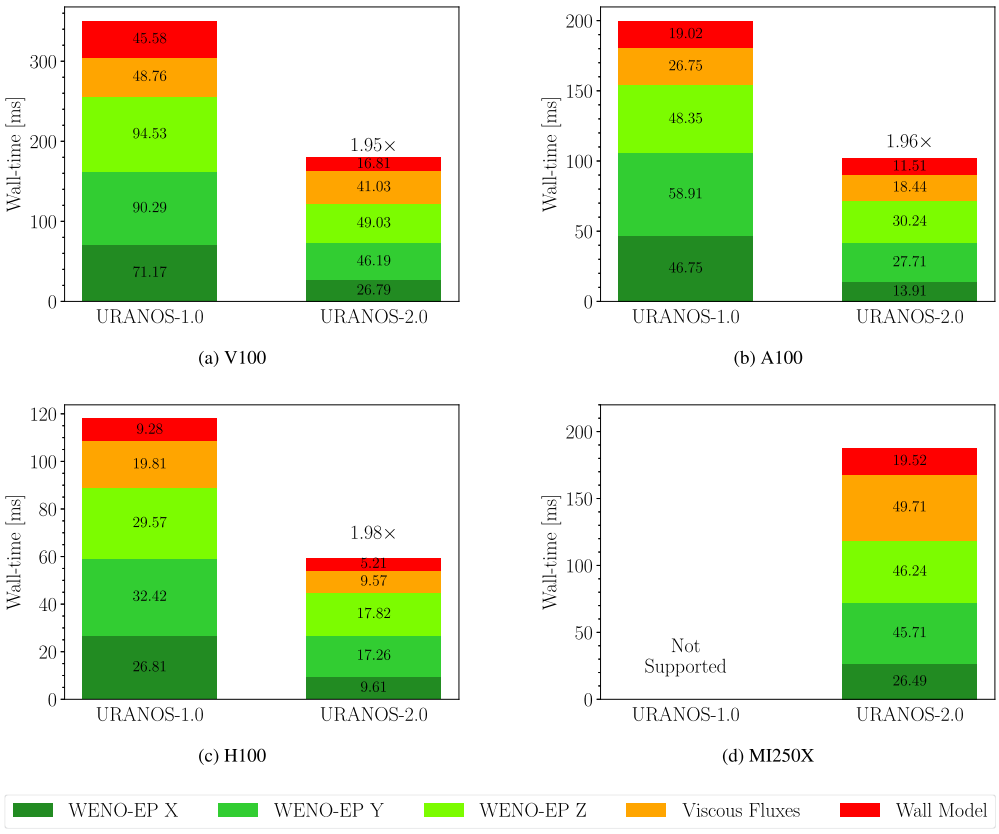


Fig. 5. Time per iteration for URANOS-1.0 and 2.0 using different GPU architectures. For the AMD MI250X, tests are run on a single GCD and reported time are halved to consider the potential of a full GPU. Focus on refactored routines.

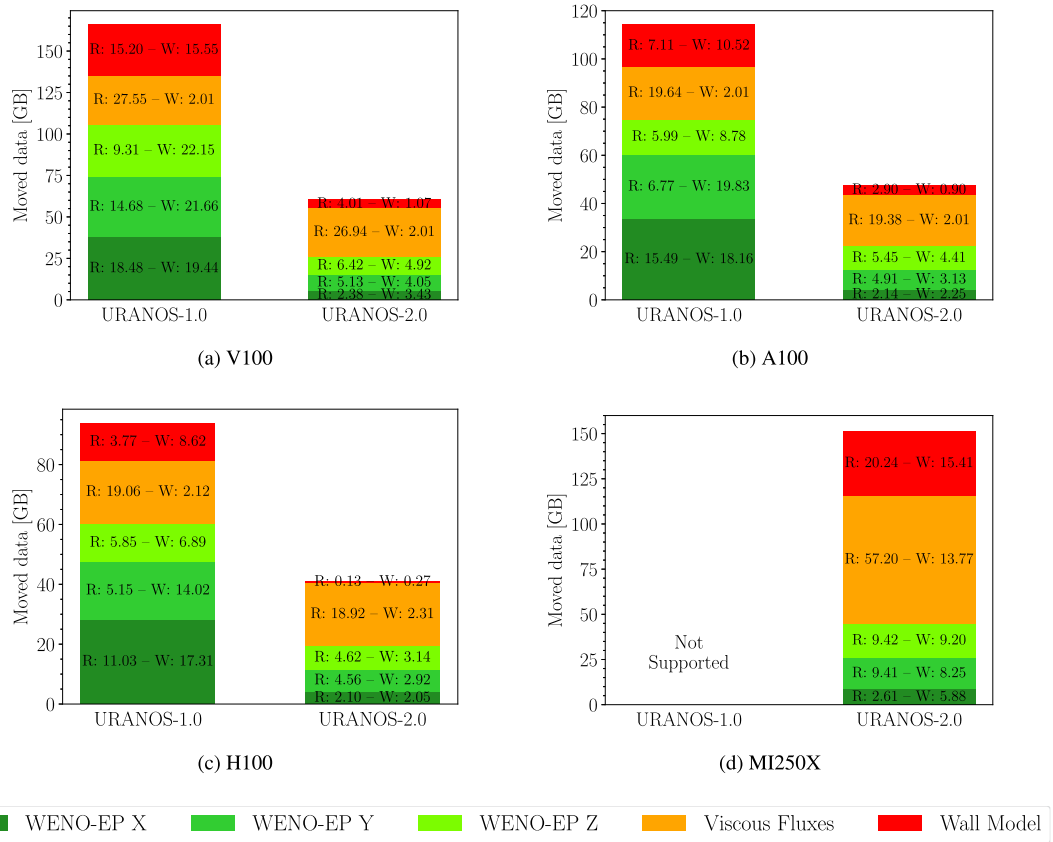


Fig. 6. Moved data from GPU main memory for URANOS-1.0 and 2.0 using different GPU architectures. Focus on refactored routines. R: Read, W: Write.

ings) from/to the device's main memory for the five refactored routines. Results are still reported for the two URANOS versions (left and right columns) and the whole set of tested GPGPUs. The findings offer valuable insights, particularly concerning the computational kernels for convective fluxes. Implementing a chunk strategy has led to a significant reduction in data movement. Importantly, this approach involves minimal read and write operations in the main memory of both NVIDIA and AMD cards, representing a substantial decrease compared to URANOS-1.0. Peak improvements of 2× are observed for WENO-EP Y/Z, while for WENO-EP X, peak reduction of 7× is achieved. These benefits can be attributed to the use of statically declared chunks for one-dimensional arrays, instructing the compiler to store them within shared memory via the `!$acc cache` directive. The approach significantly reduces access to the main memory device, contributing to the observed increase in performance in terms of time per iteration. The wall model kernel exhibits a comparable observation achieved through a distinct approach. Despite undergoing no specific alterations in variable storage and being refactored in more conventional terms, we observe a virtual disappearance of data readings and writings associated with the wall model kernel in the case of H100 URANOS-2.0. In contrast, these data movement components remain relatively stable in all architectures and particularly persistent for the MI250X. The key lies in the increased cache per streaming multiprocessor of H100 compared to previous NVIDIA generations and the AMD card. The augmented cache enables the compiler to store all the critical vector information for wall model calculations within its confines. However, it is crucial to emphasize that the refactoring work in URANOS-2.0 plays a pivotal role in this scenario. In fact, with URANOS-1.0, paired with the H100 architecture, there is no discernible reduction in data movement associated with wall modeling. Moreover, it is noteworthy that the data movement linked with viscous flux computation shows minimal variation across the solver versions and selected architectures. Particularly, the subroutine exhibits significant data movement intensity for the AMD card. This observation highlights the inherent memory-bound nature of the viscous flux computation, where advanced strategies, like variable chunking or alternative memory management approaches, provide limited opportunities for efficiency enhancement. As a result, the observed speedup in this specific solver segment primarily arises from minimizing idle operations and maximizing the reuse of identically named variables to optimize register usage.

A roofline analysis is performed to demonstrate the performance enhancements more effectively [62–64]. In particular, the analyses are straightforwardly available on actual versions of the NVIDIA Compute suite, while for AMD cards, manual reconstruction of the performance has to be resorted. Specifically, the Frontier user guide has been adopted to collect the write- and read-requests and floating point operations [65]. This evaluation method is beneficial in assessing a device's performance relative to its peak capabilities and identifying bottlenecks that can aid in further optimization. To create a roofline plot, the performance of a kernel (measured in TFLOP/s) is plotted as a function of Arithmetic Intensity (AI), which is the ratio of floating-point operations to data movement (FLOPs/byte). A roofline plot consists of two ceilings, a roof, and a region below the first ceiling that is limited by the device main memory bandwidth, known as the memory-bound region. Different levels of memory, such as L1 and L2 caches, can be considered, resulting in varying memory ceilings. In this case, we focus on a standard roofline analysis based on the device's main memory. The area below the roof is constrained by the device's peak performance, which is referred to as the compute-bound region. Peak performance is typically assumed to be that of Fused Multiply-Add (FMA) operations, which is not achievable in realistic CFD computations. Fig. 7 shows the results of this analysis. In the context of CFD codes, it is customary for the most computationally intensive operations to reside within the memory-bound region. Analysis of the data revealed that all five functions fell below this region. However, all refactored routines indicate consistent and noteworthy improvements in AI and TFLOP/s performance. An increase in AI signifies improved computational efficiency relative to data movement in a given computing task, which is a desirable outcome for CFD codes and similar applications. Higher AI values represent an increased number of floating-point operations executed per unit of data transferred, optimizing the available main memory bandwidth utilization. In practical terms, a higher AI often implies enhanced data locality and a more efficient use of computational resources, leading to improved overall performance. This balanced approach mitigates potential bottlenecks associated with memory access. Therefore, when discussing AI improvements, we highlight progress toward a more efficient and balanced use of computing resources. All refactored subroutines (a part of viscous flux computation) experienced a noteworthy boost in AI, a key indicator of enhanced efficiency. This increase in AI led to cascading effects on performance, as evidenced by the increase in TFLOP/s, thus reducing the computational time. This positive trend extends to the entire picture, particularly for the wall-modeling kernel in the H100 architecture, where the AI improvement is an outlier. Importantly, this advancement positions the wall modeling in H100 outside the memory-bounded region, indicating substantial progress in meeting the computational demands with greater efficiency. This result is, of course, a combination of kernel refactoring and enhanced technologies. Lastly, it is worth noting that the computation of viscous fluxes stands out as the only function where no improvement in AI is achieved. This lack of benefit is attributable to the inherent structure of these terms, which limits the enhancement of local data exploitation. Despite this specific case, the refactoring operations were successful in pushing this routine to approach the theoretical limit of the memory-bound ceiling for the entire set of NVIDIA hardware. In contrast, a distinct behavior in the computation of viscous terms is observed for AMD architectures. Here, viscous flux computations exhibit low AI values and are significantly distant from the memory-bound edge. This observation may suggest the possibility of developing two different versions of the viscous terms subroutine—one optimized for NVIDIA cards and another for AMD cards—acknowledging the architectural differences. However, for the sake of simplicity, such complexity was not pursued in this work.

5.3. Multi GPU scaling and performance

After examining the performance attainable with a single GPU, we now focus on URANOS-2.0's multi-GPU capabilities. To gauge solver efficiency, we employ the computational power of Leonardo-booster (Cineca) and LUMI-G (CSC) supercomputers partitions. The former, Leonardo, represents a TIER-0 architecture based on the Atos BullSequana XH2000 architecture, comprising 3456 Intel Ice Lake nodes, each equipped with 32 cores and complemented by 4 NVIDIA A100 SXM4 64 GB GPUs. LUMI is an HPE Cray EX system consisting of 2978 nodes, each equipped with a 64-core AMD Trento CPU and four AMD MI250X GPUs. Leonardo and LUMI currently rank sixth and fifth among the world's fastest supercomputers as per the November 2023 Top500 ranking. The NVIDIA HPC SDK 23.1 compiler from NVIDIA, and the AMD ROCm 5.2.3 suite with the Cray Programming Environment (CPE) release 23.09, are utilized for conducting tests in this study. A GPU-aware communication model is implemented to maximize data exchange efficiency, obviating any unnecessary transfers between the CPU and GPU.

Fig. 8 provides a visual representation of the code's parallel performance over Leonardo supercomputing architecture, emphasizing both the strong scaling (Fig. 8a) and weak (Fig. 8b) scaling. The data are reported as a function of the number of computing nodes and of GPUs. For strong scaling, the metrics are determined by maintaining a consistent total number of grid points, specifically with dimensions set to $N_x \times N_y \times N_z = 1024 \times 1024 \times 512 \simeq 536 \cdot 10^6$, while varying the resources involved in the computation. Conversely, for weak scaling, the computational unit size per node remains fixed, still at $536 \cdot 10^6$ grid points, allowing for the evaluation of the network efficiency and effectiveness of the parallelization strategy. Experiments are conducted on a

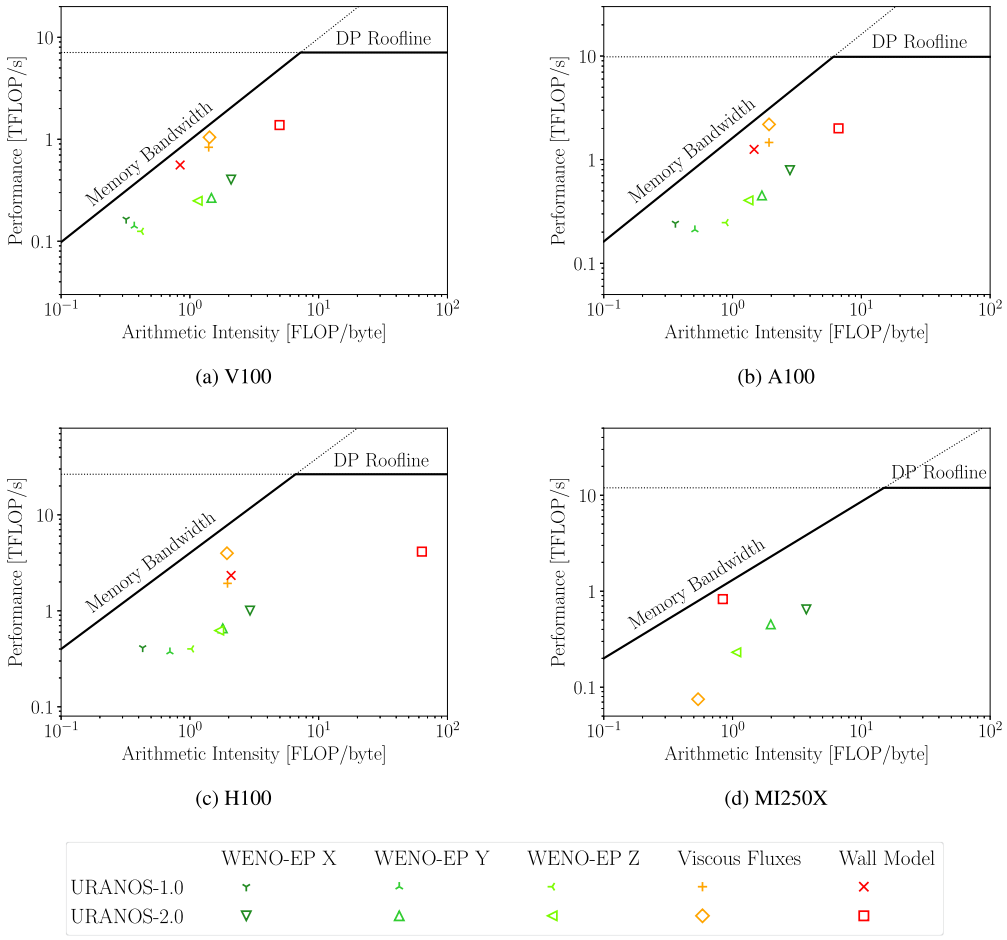


Fig. 7. Roofline analysis for improved subroutines in URANOS-1.0 and 2.0 using different GPU architectures.

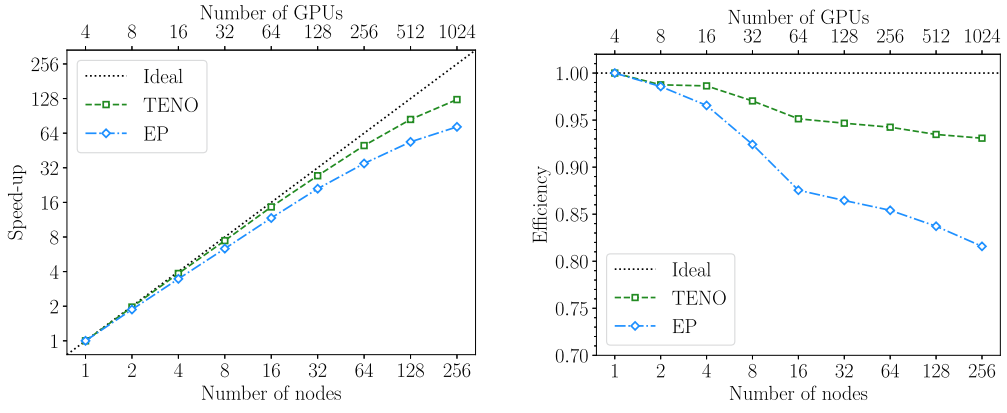
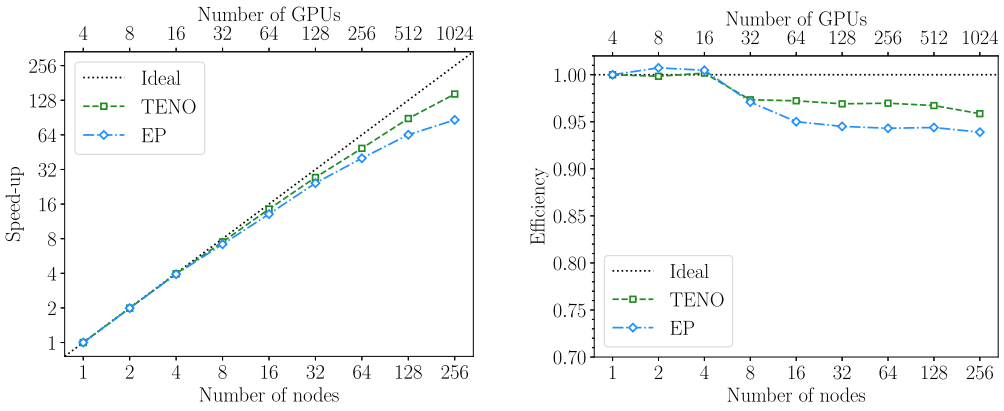


Fig. 8. Leonardo strong (8a) and weak (8b) scaling. Strong scaling is achieved by calculating the ratio of the elapsed time for one node to the elapsed time on n-nodes for a fixed grid of $1024 \times 1024 \times 512$ points. Weak scaling is determined by taking the ratio of the elapsed time for one node to the elapsed time obtained with n-nodes, keeping the points-per-node fixed at $1024 \times 1024 \times 512$.

tri-periodic flow with zero velocity, unit temperature, and unit pressure. Specifically, investigations use the EP 6 and TENO 7 schemes, with the latter representing the least and the most computationally demanding schemes disposed of by URANOS-2.0. Fig. 8a illustrates that the strong scaling pattern closely approximates the ideal behavior with efficiency of nearly 50% for TENO 7 and 30% for EP 6 over 256 nodes. It is worth noting that even with 64 nodes, the efficiency remains steadily high, reaching approximately 80% and 54% for TENO and EP, respectively. The superior efficiency of the TENO scheme is mainly due to its ability to manage an increased computational workload effectively compared to MPI communications. In contrast, the EP scheme demonstrates a less pronounced efficiency in this regard. Regarding weak scaling (Fig. 8b), the solver demonstrates good efficiency, particularly with its ability to maintain good performance up to an impressive scale of $1.4 \cdot 10^{11}$ grid points over 256 nodes. Specifically, TENO 7 sustains an efficiency of



(a) Strong scaling (b) Weak scaling

Fig. 9. LUMI strong (9a) and weak (9b) scaling. Strong scaling is achieved by calculating the ratio of the elapsed time for one node to the elapsed time on n-nodes for a fixed grid of $1024 \times 1024 \times 512$ points. Weak scaling is determined by taking the ratio of the elapsed time for one node to the elapsed time obtained with n-nodes, keeping the points-per-node fixed at $1024 \times 1024 \times 512$. Note that because each MI250X is composed by two GCDs, MPI ranks are doubled for LUMI scalings compared to Leonardo.

Table 3
URANOS-2.0 wall-clock time per time step on Leonardo and LUMI architectures using a multi-GPU framework.

		Wall-clock time (Leonardo)		Wall-clock time (LUMI)	
Nodes	**GPUs**	**EP** [s/iter]	**TENO** [s/iter]	**EP** [s/iter]	**TENO** [s/iter]
1	4	$8.49 \cdot 10^{-1}$	2.30	1.20	4.92
2	8	$4.54 \cdot 10^{-1}$	1.17	$6.02 \cdot 10^{-1}$	2.47
4	16	$2.46 \cdot 10^{-1}$	$5.98 \cdot 10^{-1}$	$3.05 \cdot 10^{-1}$	1.24
8	32	$1.34 \cdot 10^{-1}$	$3.10 \cdot 10^{-1}$	$1.68 \cdot 10^{-1}$	$6.56 \cdot 10^{-1}$
16	64	$7.26 \cdot 10^{-2}$	$1.58 \cdot 10^{-1}$	$9.16 \cdot 10^{-2}$	$3.40 \cdot 10^{-1}$
32	128	$4.04 \cdot 10^{-2}$	$8.44 \cdot 10^{-2}$	$4.92 \cdot 10^{-2}$	$1.81 \cdot 10^{-1}$
64	256	$2.43 \cdot 10^{-2}$	$4.61 \cdot 10^{-2}$	$2.99 \cdot 10^{-2}$	$1.01 \cdot 10^{-1}$
128	512	$1.58 \cdot 10^{-2}$	$2.73 \cdot 10^{-2}$	$1.87 \cdot 10^{-2}$	$5.55 \cdot 10^{-2}$
256	1024	$1.17 \cdot 10^{-2}$	$1.84 \cdot 10^{-2}$	$1.39 \cdot 10^{-2}$	$3.40 \cdot 10^{-2}$

approximately 93% up to this maximum size, whereas EP maintains a commendable efficiency of 80%.

Fig. 9 presents the results for the LUMI architecture, which demonstrate similar outcomes as those observed for the Leonardo cluster. It must be noted that because each MI250X is composed by two GCDs, MPI ranks are doubled for LUMI scalings compared to Leonardo. Specifically, in terms of strong scaling (Fig. 9a), LUMI exhibits performance comparable to Leonardo, with an efficiency of around 57% for TENO on 256 nodes and 34% for EP on the same number of nodes. However, in the case of weak scaling performance (Fig. 9b), the LUMI architecture diverges from Leonardo, as URANOS-2.0 maintains a consistently very good level of efficiency, reaching approximately 94% and 96% for EP and TENO, respectively, at the peak of 256 nodes. The discrepancies between the two architectures can be attributed to differences in network performance and the lower performance of the MI250X cards compared to the A100 cards, exacerbating the computational workload relative to the MPI data movement.

Finally, for greater clarity and to provide more information to the reader, Table 3 reports the wall-clock time per iteration for the two architectures as a function of the number of computational units in use. Notably, even when using TENO 7 on Leonardo, the time per iteration in a realistic scenario (i.e., using 32 nodes, which reflects typical machine traffic) is approximately $8.5 \cdot 10^{-2}$ s/iter. This implies that completing 10 million iterations, which is often required for studying the low-frequency dynamics of a wall flow problem, would take around 10 days.

6. Conclusions

We have presented URANOS-2.0, a renewed and refactored version of our in-house developed, GPU-accelerated Navier-Stokes solver, designed explicitly for compressible wall flows. The solver employs a high-order/high-resolution finite difference framework to address DNS, WRLES, and WMLES of compressible wall flows. In particular, the actual release has been optimized to take advantage of the most recent opportunities offered by the international pre-exascale HPC architectures. To summarize present work outcomes, three major points make the current version outperform the solver precursor:

1. URANOS-2.0 introduces enhancements to the solver options, expanding the array of numerical schemes and flow modeling capabilities. This further extends the already comprehensive modeling opportunities the URANOS 1.0 version provides.
2. In contrast to its initial iteration, which was specifically tailored for NVIDIA GPUs, URANOS-2.0 fully leverages the OpenACC standard, enhancing vendor neutrality and including AMD card to underscore the solver's adaptability to diverse GPU architectures in the ever-evolving HPC market.
3. Due to an intense refactoring of the most computationally demanding kernels, URANOS-2.0 achieves heightened performance with a time-per-iteration reduced by two over equivalent architectures. Moreover, compared to version 1.0, optimized for NVIDIA V100 cards, the current version running on NVIDIA H100 GPUs attains a speedup of over fivefold.

The solver's capabilities are evaluated on two premier multi-GPU architectures within the EuroHPC JU: LUMI and Leonardo, ranked fifth and sixth globally according to the November 2023 Top500 ranking. The solver demonstrates very good performance over both, boasting over 80% strong scaling efficiency across 64 compute nodes (256 GPUs) for both LUMI and Leonardo and reaching approximately 95% and 90% weak scaling efficiency on LUMI and Leonardo, respectively, when scaled up to 256 nodes (1024 GPUs). In essence, the current URANOS iteration establishes itself as an increasingly flexible and multi-platform fluid dynamic solver able to exploit the most performing supercomputing architectures available in Europe and address limitations found in other freely available software. In particular, two key features consistently set URANOS apart from other similar software in the CFD panorama: i) Leveraging OpenACC as a standard directive-based language extension, URANOS achieves vendor-neutral compatibility with contemporary and future GPU-based clusters. This minimizes the tuning process and refactoring efforts amid the HPC market's dynamic changes and continuous evolution. Consequently, URANOS stands out as an easily maintained, open-source, and flexible platform prone for upcoming GPU technologies. ii) Unlike currently available high-resolution

open-source solvers primarily based on DNS, URANOS adopts cutting-edge turbulence modeling strategies, specifically LES and WMLES approaches. This choice positions URANOS as a pivotal starting point for developing more complex physical models tailored to engineering applications. The program's future advancements will incorporate the treatment of complex geometries to address engineering-based applications as those in [66,67]. The open-source version of the code is accessible at the following link: https://github.com/uranos-gpu/uranos-gpu.

Funding

The authors would also like to acknowledge the "Consorzio Interuniversitario per il Calcolo Automatico" (CINECA) for generously providing access to the Marconi100 and Leonardo supercomputing infrastructures within the frameworks of the IsB26_HERMES (HP10BJK91V) and IsB28_ARTEMIDE (HP10BEJ9YD) projects. Gratitude is extended to the EuroHPC JU for facilitating access to the LUMI infrastructure based at CSC, Finland, through the URANOS-AMD project (EHPC-BEN-2023B12-045).

CRediT authorship contribution statement

Francesco De Vanna: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Giacomo Baldan:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that support the findings of this study are available on request from the corresponding author, Francesco De Vanna.

Acknowledgements

The authors have several persons to acknowledge who have played significant roles in developing and supporting this project. Firstly, special thanks to Professor Ernesto Benini for spearheading the URANOS project over the years and turning it into a reality. Secondly, heartfelt appreciation goes to Dr. Matt Bettencourt, Ph.D., for guiding us through this venture and emphasizing the importance of having an optimized and well-supported code. Finally, thanks go to NVIDIA Coorporation for granting access to the upcoming H100 GPU architectures, enabling us to test the solver on these platforms.

References

[1] Francesco De Vanna, Filippo Avanzi, Michele Cogo, Simone Sandrin, Matt Bettencourt, Francesco Picano, Ernesto Benini, URANOS: a GPU accelerated Navier-Stokes solver for compressible wall-bounded flows, Comput. Phys. Commun. 287 (2023) 108717, https://doi.org/10.1016/j.cpc.2023.108717.

[2] Xiaojue Zhu, Everett Phillips, Vamsi Spandan, John Donners, Gregory Ruetsch, Joshua Romero, Rodolfo Ostilla-Mónico, Yantao Yang, Detlef Lohse, Roberto Verzicco, et al., AFiD-GPU: a versatile Navier–Stokes solver for wall-bounded turbulent flows on GPU clusters, Comput. Phys. Commun. 229 (2018) 199–210, https://doi.org/10.1016/j.cpc.2018.03.026.

[3] Pedro Costa, Everett Phillips, Luca Brandt, Massimiliano Fatica, GPU acceleration of CaNS for massively-parallel direct numerical simulations of canonical fluid flows, Comput. Math. Appl. 81 (2021) 502–511, https://doi.org/10.1016/j.camwa.2020.01.002.

[4] OpenACC, https://www.openacc.org/, 2022.

[5] David J. Lusher, Satya P. Jammy, Neil D. Sandham, OpenSBLI: automated code-generation for heterogeneous computing architectures applied to compressible fluid dynamics on structured grids, Comput. Phys. Commun. 267 (2021) 108063, https://doi.org/10.1016/j.cpc.2021.108063.

[6] Freddie D. Witherden, Antony M. Farrington, Peter E. Vincent, PyFR: an open source framework for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach, Comput. Phys. Commun. 185 (11) (2014) 3028–3040, https://doi.org/10.1016/j.cpc.2014.07.011.

[7] Freddie D. Witherden, Antony M. Farrington, Peter E. Vincent, Pyfr: an open source framework for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach, Comput. Phys. Commun. 185 (11) (2014) 3028–3040, https://doi.org/10.1016/j.cpc.2014.07.011.

[8] Joshua Romero, Jacob Crabill, Jerry E. Watkins, Freddie D. Witherden, Antony Jameson, ZEFR: a GPU-accelerated high-order solver for compressible viscous flows using the flux reconstruction method, Comput. Phys. Commun. 250 (2020) 107169, https://doi.org/10.1016/j.cpc.2020.107169.

[9] Matteo Bernardini, Davide Modesti, Francesco Salvadore, Sergio Pirozzoli, STREAmS: a high-fidelity accelerated solver for direct numerical simulation of compressible turbulent flows, Comput. Phys. Commun. 263 (2021) 107906, https://doi.org/10.1016/j.cpc.2021.107906.

[10] Guillaume A. Bres, Sanjeeb T. Bose, Christopher B. Ivey, Michael Emory, Frank Ham, GPU-accelerated large-eddy simulations of supersonic jets from twin rectangular nozzle, in: 28th AIAA/CEAS Aeroacoustics 2022 Conference, 2022, p. 3001.

[11] Matteo Bernardini, Davide Modesti, Francesco Salvadore, Srikanth Sathyanarayana, Giacomo Della Posta, Sergio Pirozzoli, STREAmS-2.0: supersonic turbulent accelerated Navier-Stokes solver version 2.0, Comput. Phys. Commun. 285 (2023) 108644, https://doi.org/10.1016/j.cpc.2022.108644.

[12] Esteban Ferrer, Gonzalo Rubio, Gerasimos Ntoukas, Wojciech Laskowski, O.A. Mariño, S. Colombo, Andrés Mateo-Gabín, H. Marbona, F. Manrique de Lara, David Huergo, et al., HORSES3D: a high-order discontinuous Galerkin solver for flow simulations and multi-physics applications, Comput. Phys. Commun. 287 (2023) 108700, https://doi.org/10.1016/j.cpc.2023.108700.

[13] Xiao Liu, Yan-Qing Ma, AMFlow: a mathematica package for Feynman integrals computation via auxiliary mass flow, Comput. Phys. Commun. 283 (2023) 108565, https://doi.org/10.1016/j.cpc.2022.108565.

[14] Johan Larsson, Soshi Kawai, Julien Bodart, Ivan Bermejo-Moreno, Large eddy simulation with modeled wall-stress: recent progress and future directions, Mech. Eng. Rev. 3 (1) (2016) 15–00418, https://doi.org/10.1299/mer.15-00418.

[15] Francesco De Vanna, Michele Cogo, Matteo Bernardini, Francesco Picano, Ernesto Benini, Unified wall-resolved and wall-modeled method for large-eddy simulations of compressible wall-bounded flows, Phys. Rev. Fluids 6 (3) (2021) 034614, https://doi.org/10.1103/PhysRevFluids.6.034614.

[16] Francesco De Vanna, Giacomo Baldan, Francesco Picano, Ernesto Benini, Effect of convective schemes in wall-resolved and wall-modeled LES of compressible wall turbulence, Comput. Fluids 250 (2023) 105710, https://doi.org/10.1016/j.compfluid.2022.105710.

[17] Sergio Pirozzoli, Numerical methods for high-speed flows, Annu. Rev. Fluid Mech. 43 (2011) 163–194, https://doi.org/10.1146/annurev-fluid-122109-160718.

[18] Sergio Pirozzoli, Matteo Bernardini, Turbulence in supersonic boundary layers at moderate Reynolds number, J. Fluid Mech. 688 (2011) 120–168, https://doi.org/10.1017/jfm.2011.368.

[19] Sergio Pirozzoli, Stabilized non-dissipative approximations of Euler equations in generalized curvilinear coordinates, J. Comput. Phys. 230 (8) (2011) 2997–3014, https://doi.org/10.1016/j.jcp.2011.01.001.

[20] Guoyan Zhao, Mingbo Sun, Antonio Memmolo, Sergio Pirozzoli, A general framework for the evaluation of shock-capturing schemes, J. Comput. Phys. 376 (2019) 924–936, https://doi.org/10.1016/j.jcp.2018.10.013.

[21] Guang-Shan Jiang, Chi-Wang Shu, Efficient implementation of weighted ENO schemes, J. Comput. Phys. 126 (1) (1996) 202–228, https://doi.org/10.1006/jcph.1996.0130.

[22] Bernardo Cockburn, Chi-Wang Shu, Claes Johnson, Eitan Tadmor, Chi-Wang Shu, Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws, in: Advanced Numerical Approximation of Nonlinear Hyperbolic Equations, 1998.

[23] Chi-Wang Shu, High order ENO and WENO schemes for computational fluid dynamics, in: High-Order Methods for Computational Physics, 1999, pp. 439–582.

[24] Sergio Pirozzoli, Conservative hybrid compact-WENO schemes for shock-turbulence interaction, J. Comput. Phys. 178 (1) (2002) 81–117, https://doi.org/10.1006/jcph.2002.7021.

[25] Bao-Shan Wang, Peng Li, Zhen Gao, Wai Sun Don, An improved fifth order alternative WENO-Z finite difference scheme for hyperbolic conservation laws, J. Comput. Phys. 374 (2018) 469–477, https://doi.org/10.1016/j.jcp.2018.07.052.

[26] Marcos Castro, Bruno Costa, Wai Sun Don, High order weighted essentially non-oscillatory WENO-Z schemes for hyperbolic conservation laws, J. Comput. Phys. 230 (5) (2011) 1766–1792, https://doi.org/10.1016/j.jcp.2010.11.028.

[27] Lin Fu, Xiangyu Y. Hu, Nikolaus A. Adams, A family of high-order targeted ENO schemes for compressible-fluid simulations, J. Comput. Phys. 305 (2016) 333–359, https://doi.org/10.1016/j.jcp.2015.10.037.

[28] Lin Fu, Xiangyu Y. Hu, Nikolaus A. Adams, Targeted ENO schemes with tailored resolution property for hyperbolic conservation laws, J. Comput. Phys. 349 (2017) 97–121, https://doi.org/10.1016/j.jcp.2017.07.054.
[29] Lin Fu, A very-high-order TENO scheme for all-speed gas dynamics and turbulence, Comput. Phys. Commun. 244 (2019) 117–131, https://doi.org/10.1016/j.cpc.2019.06.013.
[30] Tian Liang, Feng Xiao, Wei Shyy, Lin Fu, A fifth-order low-dissipation discontinuity-resolving TENO scheme for compressible flow simulation, J. Comput. Phys. 467 (2022) 111465, https://doi.org/10.1016/j.jcp.2022.111465.
[31] Lin Fu, Review of the high-order TENO schemes for compressible gas dynamics and turbulence, Arch. Comput. Methods Eng. 30 (4) (2023) 2493–2526, https://doi.org/10.1007/s11831-022-09877-7.
[32] Lin Fu, Xiangyu Y. Hu, Nikolaus A. Adams, A new class of adaptive high-order targeted ENO schemes for hyperbolic conservation laws, J. Comput. Phys. 374 (2018) 724–751, https://doi.org/10.1016/j.jcp.2018.07.043.
[33] Lin Fu, Very-high-order TENO schemes with adaptive accuracy order and adaptive dissipation control, Comput. Methods Appl. Mech. Eng. 387 (2021) 114193, https://doi.org/10.1016/j.cma.2021.114193.
[34] Lin Fu, Tian Liang, A new adaptation strategy for multi-resolution method, J. Sci. Comput. 93 (2) (2022) 43, https://doi.org/10.1007/s10915-022-02012-5.
[35] Francesco De Vanna, Francesco Picano, Ernesto Benini, A sharp-interface immersed boundary method for moving objects in compressible viscous flows, Comput. Fluids 201 (2020) 104415, https://doi.org/10.1016/j.compfluid.2019.104415.
[36] Francesco De Vanna, Francesco Picano, Ernesto Benini, Mark Kenneth Quinn, Large-eddy simulations of the unsteady behavior of a hypersonic intake at Mach 5, AIAA J. 59 (10) (2021) 3859–3872, https://doi.org/10.2514/1.J060160.
[37] Francesco De Vanna, Alberto Benato, Francesco Picano, Ernesto Benini, High-order conservative formulation of viscous terms for variable viscosity flows, Acta Mech. 232 (6) (2021) 2115–2133, https://doi.org/10.1007/s00707-021-02937-2.
[38] Davide Modesti, Sergio Pirozzoli, Reynolds and Mach number effects in compressible turbulent channel flow, Int. J. Heat Fluid Flow 59 (2016) 33–49, https://doi.org/10.1016/j.ijheatfluidflow.2016.01.007.
[39] Sigal Gottlieb, Chi-Wang Shu, Total variation diminishing Runge-Kutta schemes, Math. Comput. 67 (221) (1998) 73–85, https://doi.org/10.1090/S0025-5718-98-00913-2.
[40] F. Ducros, V. Ferrand, Franck Nicoud, C. Weber, D. Darracq, C. Gacherieu, Thierry Poinsot, Large-eddy simulation of the shock/turbulence interaction, J. Comput. Phys. 152 (2) (1999) 517–549, https://doi.org/10.1006/jcph.1999.6238.
[41] Sergio Pirozzoli, Matteo Bernardini, Probing high-Reynolds-number effects in numerical boundary layers, Phys. Fluids 25 (2) (2013), https://doi.org/10.1063/1.4792164.
[42] Amareshwara Sainadh Chamarthi, Natan Hoffmann, Steven Frankel, A wave appropriate discontinuity sensor approach for compressible flows, Phys. Fluids 35 (6) (2023), https://doi.org/10.1063/5.0149314.
[43] William Sutherland, LII. The viscosity of gases and molecular force, Lond. Edinb. Dublin Philos. Mag. J. Sci. 36 (223) (1893) 507–531, https://doi.org/10.1080/14786449308620508.
[44] Frank Archer Williams, The effect of temperature on the viscosity of air, Proc. R. Soc. Lond. Ser. A, Contain. Pap. Math. Phys. Character 110 (753) (1926) 141–167, https://doi.org/10.1098/rspa.1926.0008.
[45] Alberto Scotti, Charles Meneveau, Douglas K. Lilly, Generalized Smagorinsky model for anisotropic grids, Phys. Fluids A, Fluid Dyn. 5 (9) (1993) 2306–2308, https://doi.org/10.1063/1.858537.
[46] Johan Meyers, Pierre Sagaut, On the model coefficients for the standard and the variational multi-scale Smagorinsky model, J. Fluid Mech. 569 (2006) 287–319, https://doi.org/10.1017/S0022112006002850.
[47] Bogey Christophe, Christophe Bailly, Large eddy simulations of round free jets using explicit filtering with/without dynamic Smagorinsky model, Int. J. Heat Fluid Flow 27 (4) (2006) 603–610, https://doi.org/10.1016/j.ijheatfluidflow.2006.02.008.
[48] Nicoud Franck, Frédéric Ducros, Subgrid-scale stress modelling based on the square of the velocity gradient tensor, Flow Turbul. Combust. 62 (3) (1999) 183–200, https://doi.org/10.1023/A:1009995426001.
[49] H. Baya Toda, O. Cabrit, G. Balarac, S. Bose, J. Lee, H. Choi, F. Nicoud, A subgrid-scale model based on singular values for LES in complex geometries, pp. 193–202, https://web.stanford.edu/group/ctr/Summer/SP10/3_03_bayatoda.pdf, 2010.
[50] Masahide Inagaki, Tsuguo Kondoh, Yasutaka Nagano, A mixed-time-scale SGS model with fixed model-parameters for practical LES, J. Fluids Eng. 127 (1) (2005) 1–13, https://doi.org/10.1115/1.1852479.
[51] Soshi Kawai, Johan Larsson, Wall-modeling in large eddy simulation: length scales, grid resolution, and accuracy, Phys. Fluids 24 (1) (2012), https://doi.org/10.1063/1.3678331.
[52] Julien Bodart, Johan Larsson, Parviz Moin, Large eddy simulation of high-lift devices, in: 21st AIAA Computational Fluid Dynamics Conference, 2013, p. 2724.
[53] J. Bodart, J. Larsson, Wall-modeled large eddy simulation in complex geometries with application to high-lift devices, in: Annual Research Briefs, Center for Turbulence Research, Stanford University, 2011, pp. 37–48.
[54] Dan S. Henningson, John Kim, On turbulent spots in plane Poiseuille flow, J. Fluid Mech. 228 (1991) 183–205, https://doi.org/10.1017/S0022112091002677.
[55] NVIDIA, Datasheet: NVIDIA V100 tensor core GPU, https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf.
[56] NVIDIA, Datasheet: NVIDIA A100 tensor core GPU, https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf.
[57] NVIDIA, Datasheet: NVIDIA H100 tensor core GPU, https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet.
[58] AMD, AMD instinct MI200 series accelerator, https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/instinct-mi200-datasheet.pdf.
[59] NVIDIA, NVIDIA HPC SDK version 23.1 documentation, https://docs.nvidia.com/hpc-sdk/archive/23.1/index.html, 2023.
[60] AMD, ROCm 5.2.3, https://rocm.docs.amd.com/en/docs-5.2.3/release.html, 2023.
[61] Brent P. Pickering, Charles W. Jackson, Thomas R.W. Scogland, Wu-Chun Feng, Christopher J. Roy, Directive-based GPU programming for computational fluid dynamics, Comput. Fluids 114 (2015) 242–253, https://doi.org/10.1016/j.compfluid.2015.03.008.
[62] Diogo Marques, Helder Duarte, Aleksandar Ilic, Leonel Sousa, Roman Belenov, Philippe Thierry, Zakhar A. Matveev, Performance analysis with cache-aware roofline model in intel advisor, in: 2017 International Conference on High Performance Computing & Simulation (HPCS), IEEE, 2017, pp. 898–907.
[63] Khaled Z. Ibrahim, Samuel Williams, Leonid Oliker, Performance analysis of gpu programming models using the roofline scaling trajectories, in: International Symposium on Benchmarking, Measuring and Optimization, Springer, 2019, pp. 3–19.
[64] Charlene Yang, Thorsten Kurth, Samuel Williams, Hierarchical roofline analysis for GPUs: accelerating performance optimization for the NERSC-9 perlmutter system, Concurr. Comput., Pract. Exp. 32 (20) (2020) e5547, https://doi.org/10.1002/cpe.5547.
[65] OAK Ridge National Laboratory, Frontier user guide, https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html#profiling-applications.
[66] Francesco De Vanna, Giacomo Baldan, Francesco Picano, Ernesto Benini, On the coupling between wall-modeled LES and immersed boundary method towards applicative compressible flow simulations, Comput. Fluids 266 (2023) 106058, https://doi.org/10.1016/j.compfluid.2023.106058.
[67] F. De Vanna, G. Baldan, F. Picano, E. Benini, High-Reynolds compressible flows simulation with wall-modeled LES and immersed boundary method, in: ERCOFTAC Workshop Direct and Large Eddy Simulation, Springer, 2023, pp. 203–208.