# GPU-accelerated simulations of turbulence: Review of current applications and future perspectives

A. Roccon [1,2] G. Amati [3] L. Brandt [4] D. Calhoun [5] P. Costa [6] W. Lu [7] S. Pirozzoli [8] D. Richter [9] M. Umair [10,11] D. You [12] T. Zahtila [13] and C. Marchioli [1,*,†]

[1] *Polytechnic Department of Engineering and Architecture, University of Udine, Udine, Italy*
[2] *Institute of Fluid Mechanics and Heat Transfer, TU-Wien, Vienna, Austria*
[3] *HPC Department, CINECA, Rome, Italy*
[4] *Department of Environment, Land and Infrastructure Engineering (DIATI), Politecnico di Torino, Torino, Italy*
[5] *Department of Mathematics, Boise State University, Boise, Idaho 83725-1555, USA*
[6] *Process & Energy Dept., Delft University, Delft, Netherlands*
[7] *Department Mechanical Engineering, University of Melbourne, Melbourne, Australia*
[8] *Dipartimento Meccanica e Ingegneria Aerospaziale, Universitá La Sapienza, Rome, Italy*
[9] *College of Engineering, Notre Dame University, Notre Dame, Indiana 46556, USA*
[10] *FLOW, Engineering Mechanics, KTH Royal Institute of Technology, Stockholm, Sweden*
[11] *Laboratoire LEGI, UMR 5519, CNRS, BP 53, 38041 Grenoble CEDEX 09, France*
[12] *Flow Physics & Engineering Laboratory, Pohang Univ. Science & Technology, Pohang, S. Korea*
[13] *Center for Turbulence Research, Stanford University, Stanford, California 94305, USA*

(Received 20 July 2025; accepted 20 February 2026; published 23 March 2026)

The growing availability of GPU-accelerated open-source solvers has boosted the capability of tackling complex single-phase and multiphase turbulent flows by means of direct and large-eddy simulations. GPU-accelerated solvers can leverage the heterogeneous computing architectures that are available in leading high-performance computing centers worldwide, taking advantage of the higher throughput and greater energy efficiency offered by GPUs as compared to CPUs. However, porting CPU-based numerical solvers to GPUs entails many outstanding challenges, such as parallelism exposure, inter-GPU communication, memory allocation constraints, and shared memory limitations. To overcome these challenges, GPU-friendly algorithms, performance portability strategies, and careful selection of computational paradigms and programming languages must be developed. Besides, adaptive mesh refinement and data compression may be integrated to mitigate I-O bottlenecks and enable simulations of more complex geometries on top of the existing requirements imposed by incompressible flows. When compressibility effects become significant, further considerations related to the adoption of high-performance preconditioners and multigrid solvers become crucial for tackling large, sparse linear systems and extending simulations to high-Mach flows. Finally, reduced-precision arithmetic can further enhance performance, energy efficiency, and scalability. In this work, we survey current applications of GPU-accelerated solvers in the broad area of fluid mechanics and

*Contact author: marchioli@uniud.it
†Also at Department of Fluid Mechanics, CISM, 33100 Udine, Italy.

turbulence simulations and discuss the main challenges and bottlenecks associated with code porting and optimization. We then conclude our analysis with an outlook on future perspectives for enabling efficient GPU-based exascale computing of turbulence.

DOI: 10.1103/vz9c-bbzm

## I. INTRODUCTION

Computational fluid dynamics (CFD) is the branch of fluid mechanics that uses numerical analysis to predict fluid motion and forces. When dealing with the computation of turbulent flows, the most accurate approach is direct numerical simulation (DNS), which resolves all spatial and temporal flow scales without relying on the use of models. The high accuracy of DNS imposes stringent spatial and temporal grid requirements, resulting in high computational costs. A viable alternative to DNS is large-eddy simulation (LES), in which the effect of eddies smaller than a threshold scale, typically equal to the grid size, are modeled while larger eddies are resolved. LES may be used conveniently instead of DNS when higher Reynolds numbers and/or complex flow geometries are targeted. Nevertheless, the computational burden of LES can still be significant for problems of practical interest in engineering and physics. In this context, high-performance computing (HPC) plays a crucial role, as it provides the processing power and memory capacity necessary to handle the billions of grid points typically required by DNS and LES simulations.

Despite the remarkable increase of available computational power since the early 1970s, the Reynolds numbers attainable by DNS and LES still remain below those encountered in practical applications. To understand how far are we from the targeted values, let us consider a typical airfoil of a commercial airplane. Even with the top state-of-the-art high-performance computing (HPC) systems, DNS can reach up to chord Reynolds numbers $\mathrm{Re}_c \simeq \mathcal{O}(10^6)$, whereas the actual values are at least one order of magnitude higher. The maximum $\mathrm{Re}_c$ that can be handled increases by a factor roughly equal to 2 for wall-resolved LES [1]. Considering that the computational power required by DNS roughly scales with $\mathrm{Re}^{\gamma}$ with $2.5 \leqslant \gamma \leqslant 3$ [2,3], the increase required to close the gap is of the order of $10^3$. In this context, exploiting modern HPC infrastructures is crucial for advancing the fundamental understanding of turbulence through DNS and LES, as well as for improving turbulence models. However, the evolution of HPC architectures is not primarily driven by the requirements of DNS and LES; instead, flow solvers must continuously adapt to the rapid and ongoing changes in computing hardware and software paradigms. The adaptation process not only presents challenges but also offers significant opportunities to exploit emerging HPC capabilities for high-fidelity turbulence modeling. Notable examples of initiatives in this direction include the U.S. Department of Energy Exascale Computing Project (ECP) and NASA Transformational Tools and Technologies ($\mathrm{T}^3$) program, both of which exemplify the drive toward harnessing GPU-based platforms and developing computational frameworks capable of performing high-fidelity turbulence simulations at unprecedented scales.

Current trends in computing architectures are clearly reflected in the TOP500 supercomputers list, which nowadays consists entirely of exascale or preexascale systems [4]. GPU-based architectures dominate the landscape: Nine of the top 10 most powerful machines rely on GPUs, with the sole exception of FUGAKU [5], a system built on ARM CPUs. This stands in sharp contrast to the situation a decade ago: According to the November 2015 TOP500 list [4], only one of the top-10 systems employed GPUs. The shift is not limited to the highest-ranking machines; it is evident across the entire TOP500. A first reason behind the transition from CPUs to GPUs is rooted in practical considerations. To achieve the performance improvements imposed by the market in the post-Moore era [6], CPU-only HPC systems would need to scale up to tens of thousands of servers. However, such an approach would entail prohibitively complex network topologies and unrealistic datacenter sizes, while being unsustainable from both energetic and economic perspectives. This leads to a second, and perhaps more important, reason that explains the transition to GPU-based systems: Energy demand. Indeed, energy consumption and efficiency are crucial factors for the

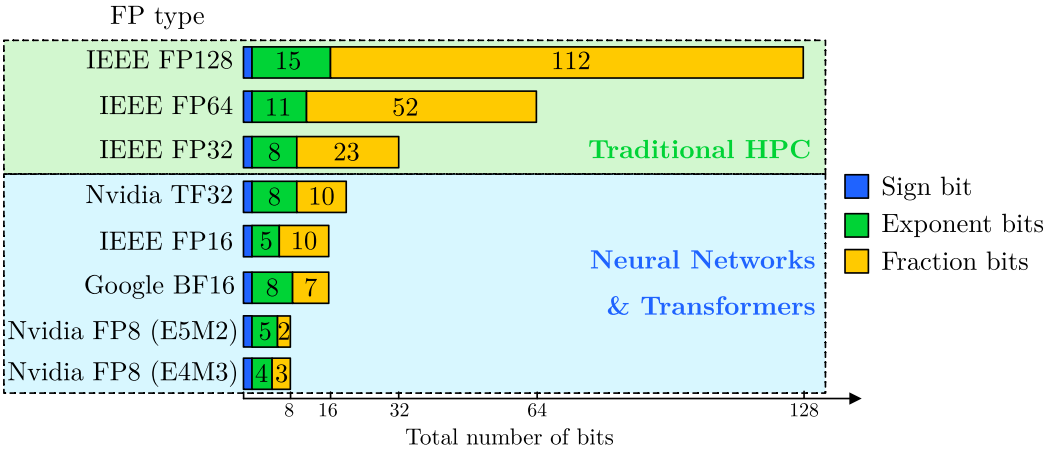


FIG. 1. Overview of the different FP types currently used in HPC. The number of bits used to represent the sign is shown in blue (1 bit for all types), the one for the exponent in green and the one for the fraction in yellow. Floating point types are grouped by their typical field of application (please note that some formats are proprietary).

design of next-generation computing facilities. In this context, GPUs constitute a possible pathway to meet the performance requirements in combination with finite power budgets and approaching the limits of traditional silicon technology [7,8]. Given the existing power constraints, achieving sustainable performance at scale requires coordinated innovation across multiple layers of design, from processors and interconnects to rack-scale integration and facility-level cooling technologies. In particular, the adoption of liquid-cooling systems has proven effective in minimizing energy overheads [8,9]. Finally, the improved energy efficiency of GPU-based architectures also contributes to a reduced carbon footprint, an aspect that is becoming increasingly important in view of the substantial energy demand of CFD simulations [10].

The widespread transition to GPUs for HPC applications has also led to significant advancements in terms of computational performance and capability to tackle increasingly complex physical problems. Still, HPC represents only a minor proportion of the global GPU market, which has always been driven by much bigger sectors, like gaming or, more recently, machine learning (ML) and artificial intelligence (AI). Nowadays, nearly all technical improvements are aimed at satisfying the needs of AI and ML applications. For example, much effort is made to allow low-precision simulations using minifloats, i.e., floating-point (FP) values represented with very few bits like the 8-bit floating point (FP8) and 4-bit floating point (FP4); see Fig. 1. AI and ML algorithms can handle this low precision [11] while standard fluid dynamics simulations typically require single or double precision, i.e., 32-bit floating (FP32) or 64-bit floating point (FP64) [12]. In some cases, even quadruple precision may be required [13].

In this context, the task of the code developer is to adapt, as much as possible, the numerical method and programming paradigms to the new hardware devices, regardless of the complexities and ease of programming. Such a task is not easy because it is time-consuming, and to obtain a future-proof code, the developer needs to identify the emerging trends of future hardware technologies (as well as libraries, programming languages, and compiler support), which are dictated by algorithms that evolve at a very fast pace. As far as CFD is concerned, a further difficulty that is likely to arise in the coming years is related to the aforementioned dominance of AI and ML: New accelerated devices will be targeted to AI and ML algorithms and thus optimized for low precision calculations. For instance, the recent B300 GPU from NVIDIA allows for a 150% improvement of FP4 computations and almost no improvement for FP32 and PF64 computations [14]. The disparity between FP64 and FP4 performance has now reached a factor of 60 000×. A last consideration

should be made about the evolution of HPC centers, which appears to go towards the creation of specialized data centers. For example, in the context of the European Union, the EuroHPC Joint Undertaking recently established the so-called AI factories: Cutting-edge data centers specifically developed for AI applications, while traditional HPC applications rely on more general computing infrastructures.

Considering the general framework just described and the complex evolution of future HPC scenarios, it is clear that developing the next generation of DNS and LES solvers will be a highly demanding task that must take into account many technical aspects and could be daunting for code developers who are just entering the field. This is particularly true for simulations targeting turbulent flows, which add the complexities of the physical problem to the overall picture [15–17]. With this work, we aim to provide a guide to help code developers identify the most suitable numerical and programming approach, thus facilitating the full exploitation of hardware performance and close the intrinsic time gap between hardware and software evolution. To do so, we set the stage by summarizing the current state-of-the-art of DNS and LES of single-phase, two-phase incompressible, or compressible turbulence in GPU-accelerated supercomputing systems. In particular, we detail the main strategies that are currently adopted to port CFD codes for heterogeneous computing infrastructures and list a series of examples of open-source GPU-ready codes. Then, we discuss the main challenges and bottlenecks that developers need to tackle during the code porting and optimization phases. Finally, as present (and future) top level supercomputers will likely be heterogeneous systems where a CPU will drive GPUs for a partial or even complete work offload, we outline future perspectives for the extension/adaption of these codes to preexascale and exascale computing infrastructures.

## II. SETTING THE STAGE: FUNDAMENTALS OF GPU ARCHITECTURES

### A. Historical perspective

In this section, we discuss from an historical perspective some fundamental concepts related to the use of GPUs for general-purpose CFD computing. As mentioned in the Introduction, a growing number of GPU-accelerated solvers are becoming available for a wide range of flow configurations. These solvers represent the answer of the fluid mechanics community to the challenges and bottlenecks encountered in the development of microprocessor architectures since the early 2010s. Indeed, if one considers the evolution of microprocessor characteristics since the early 1970s, shown in Fig. 2, then it can be appreciated that almost all characteristics—with the exception of the number of transistors—plateaued around year 2010. This plateau results from the difficulty in advancing microarchitecture production processes and the technical challenge of removing large heat fluxes from increasingly small devices. Yet the plateau alone does not explain the massive use of GPUs in HPC infrastructures, in view of the fact that the number of transistors (orange triangles in Fig. 2) still show a steady exponential growth over time in accordance with Moore's law.

A more likely reason for the growth in the use of GPUs is that the performance of CFD solvers (as well as that of many other engineering and scientific applications) is not limited by the computational power available but rather by the time required to access the data stored in different memory layers. This feature is typical of algorithms with low arithmetic intensity, in which a relatively small number of arithmetic operations are performed per data element accessed from memory. Many CFD solvers fall into this category, as they typically perform simple arithmetic operations (e.g., additions or multiplications) on large datasets. The time required to access data that are stored in memory, can be one to two orders of magnitude larger than the time required to perform a basic floating-point operation, thus posing a strong limitation on the achievable performance. Naturally, the degree to which a CFD algorithm is memory or compute bound depends on its numerical formulation: Finite-difference and finite-volume methods tend to be memory bound [19], whereas discontinuous Galerkin or spectral-based methods can achieve higher arithmetic intensity and become partially compute bound on contemporary architectures [20]. This bottleneck has become more and more

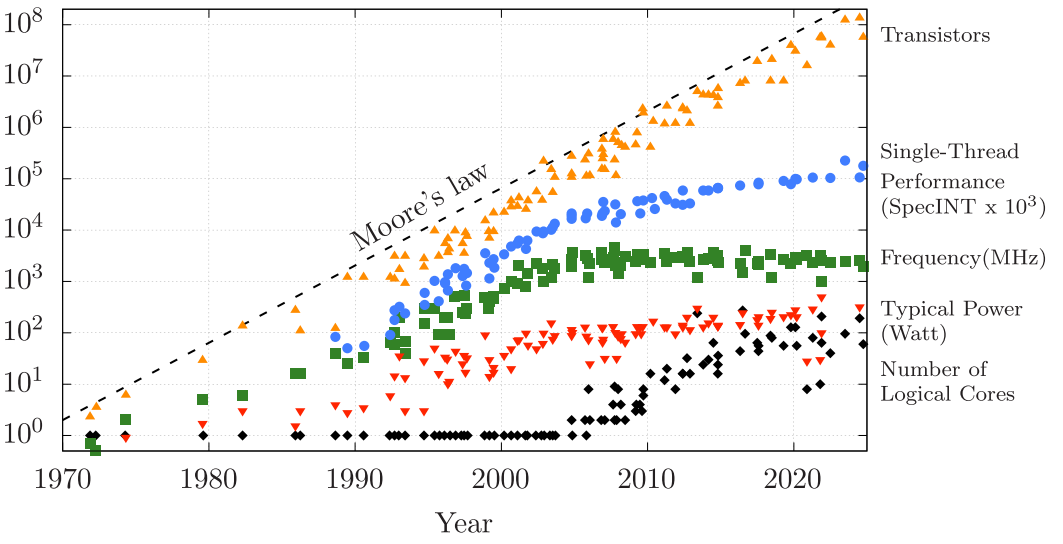


FIG. 2. Microprocessor trend data since the early 1970s. Colors represent the evolution of different microprocessor characteristics: Number of logical cores (black), power required (red), frequency (green), single-thread performance from the specINT benchmark (blue), and number of transistors (orange). For each characteristic, the corresponding values are indicated on the vertical left-end axis. Data were obtained from Ref. [18] and updated up to year 2025. The Moore law, $N/N_0 = 2^{(Y-Y_0)/T}$, is also reported as a reference, assuming that the number of transistors doubles every two years ($T = 2$) with $N_0$ the number of transistor at year $Y_0 = 1970$.

stringent over time due to the slower growth of the memory bandwidth performance as compared to the CPU computing performance, as shown in Fig. 3, where the time evolution of hardware performance as well as memory and interconnections bandwidths over the last 25 years is shown. This figure shows clearly that the performance growth rate for both memory and interconnections bandwidths ($\simeq$1.5 times every 2 years) has been lower than that of hardware ($\simeq$3 times every 2 years). This trend, referred to as the *memory wall* [21,22], has also started to impact AI algorithms [23] and has led to the development and use of low precision floats (see Fig. 1).

### B. CFD solvers and the memory wall

To better understand the memory wall and the performance limitations it imposes to CFD solvers, we can exploit the *roofline model* [24–26]. This model relies on a bound and bottleneck analysis to visualize the performance limitations as a function of the arithmetic intensity, defined as the ratio of total floating-point operations (FLOP) to total data movement (bytes) for a given code section or algorithm. This parameter can be theoretically computed for small kernels or measured via performance analyzer. The roofline model visually relates the performance $P$ (measured in FLOP per second) and the arithmetic intensity $I$ (measured in FLOP per byte) of a given code or kernel compared to the peak performance and memory bandwidth of the system, as shown in Fig. 4. In this figure, the arithmetic intensity is shown along the $x$ axis while the performance is shown along the $y$ axis, such that a given code (or snippet of code) corresponds to a point in the plot. The performance $P$ obeys two upper bounds: the peak performance, $P_{\max}$, of the processor and the bound obtained from the memory bandwidth, $\beta$. Hence, from a mathematical point of view, the roofline equation (black-solid lines in Fig. 4) reads as follows:

$$P \leqslant \min(P_{\max}, \beta I). \tag{1}$$

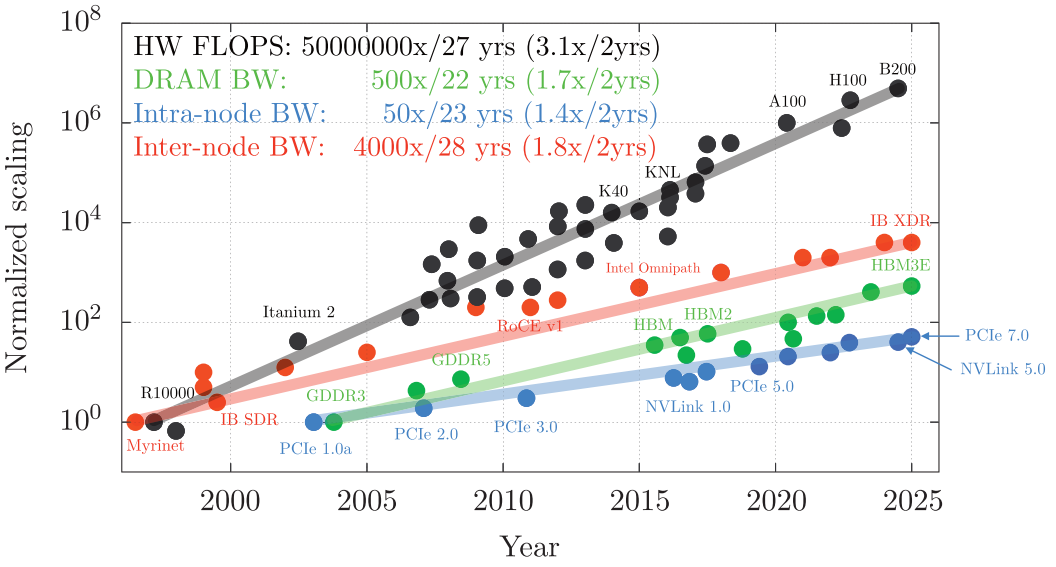


FIG. 3. Time evolution of the FP64 computing power (black), memory bandwidth (green) and intranode and internode connections (blue and red, respectively). Hardware peak floating-point operations (FLOPS) are normalized with the R10000 performance while bandwidths with the GDDR3, PCIe 1.0a, and Myrinet-200 respective values.

The intersection between the memory and computational bounds is located at $I = P_{\max}/\beta$ and identifies a balanced computation. If the code performance $P$ is characterized by an intensity $I \leqslant P_{\max}/\beta$, then it is said to be memory bound (green region in Fig. 4). Otherwise, if $I \geqslant P_{\max}/\beta$, then it is said to be compute bound (blue region in Fig. 4). Most CFD codes are memory bound, i.e., they fall within the green region of Fig. 4, as in the case study reported by Mostafazadeh *et al.* [27].

### C. Simultaneous multithreading and vectorization

To overcome the memory wall limitation, a first possible solution is to use simultaneous multithreading (SMT) architectures [28,29], where multiple logical threads (from two up to four or eight) are virtualized for each physical core. SMT improves hardware utilization by allowing other threads to execute instructions while one thread waits for data from memory, effectively hiding memory latency. While SMT does not increase the arithmetic intensity of an algorithm, it can help saturate available memory bandwidth and improve overall throughput for memory-bound codes. In cases where cache reuse is significant, SMT may also enhance cache efficiency. For a more precise analysis, a hierarchical roofline model may be used to account for multiple memory levels.

Figure 5 illustrates the SMT concept. Each row represents the execution unit for a single cycle: a colored box indicates that the processor is computing an instruction while an empty box denotes an unused (idle) slot. By using multiple threads, the idle time (time required to access data not directly available in the fast cache memory) can be minimized by handling the instructions of another thread. Compared to multiprocess systems, thread-context switching is faster and uses fewer resources because threads share the same memory space and system resources. The shared memory also simplifies communication between threads, eliminating the need for complex interprocess communication mechanisms.

Beyond SMT, data-level parallelism via vectorized single instruction multiple data (SIMD) on CPUs and single instruction multiple threads (SIMT) on GPUs helps in improving performance and mitigating the impact of the memory wall. SIMD allows a single CPU instruction to operate on multiple data elements simultaneously, executing several computations in parallel within a single

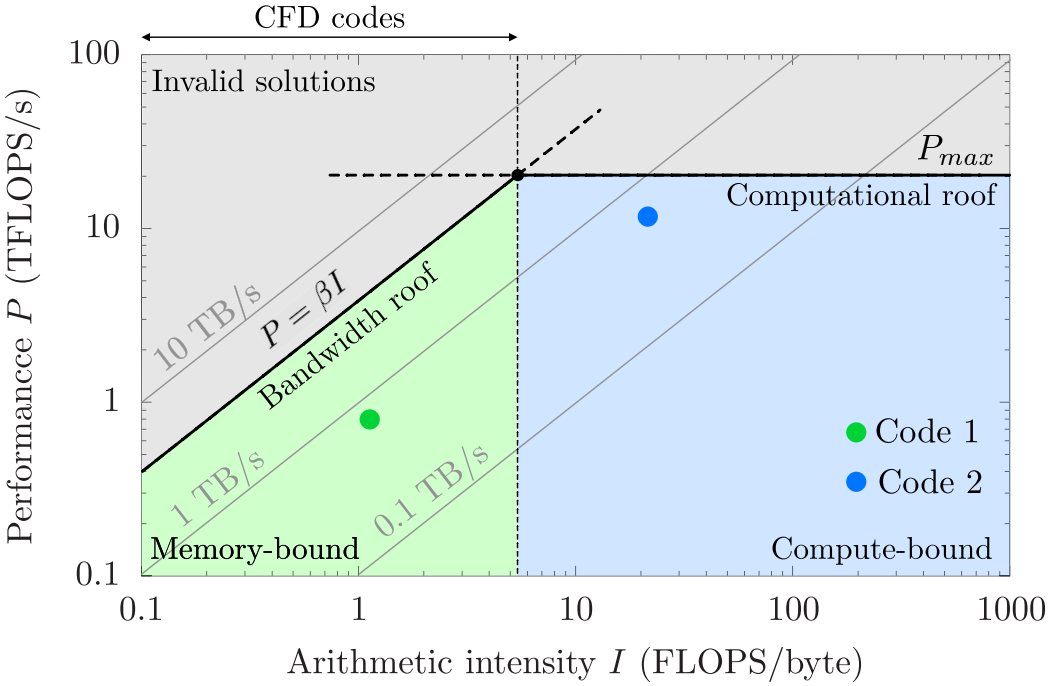


FIG. 4. Example of roofline model in log-log scale. The performance $P$ of a given code or kernel is reported as a function of the arithmetic intensity $I$. The code performance is bounded by memory bandwidth (left, green) or by computational power (right, blue). The roofline line $P = \min(P_{\max}, \beta I)$ identifies the maximum performance achievable by any given code. The case shown here refers to a bandwidth $\beta = 4$ TB/s and maximum performance $P_{\max} = 20$ TFLOP/s (in double precision). CFD solvers are often memory bound, as in the case of Code 1 (green full circles) for example. In general, several rooflines can be identified depending on the type of instructions (FP32 and FP64) and memory pattern (L1, L2, L3, DRAM, etc.) considered. An example is provided by the gray lines in the figure, which are associated to different memory bandwidths.

cycle and increasing arithmetic intensity by performing more FLOPs per memory access. SIMT generalizes this concept for GPUs, where thousands of threads are organized in groups (called warps or wave fronts in the Nvidia and AMD terminology, respectively) that execute the same instruction concurrently. By overlapping computation with memory accesses across threads, SIMT efficiently hides memory latency and maintains high utilization of arithmetic units. This fine-grained vectorization is particularly beneficial for CFD solvers with regular memory access patterns, as it allows multiple data points to be processed simultaneously while minimizing idle execution units and power overhead [30,31].

### D. From CPU to GPU architectures

GPUs enable the exploitation of data-level parallelism through fine-grained vectorization, leveraging massive thread-level parallelism organized in SIMT groups. Specifically, GPUs schedule thousands of lightweight threads that execute instructions in perfect synchronization (lockstep) within each group, allowing memory latency to be hidden while other threads perform computations. This concept is illustrated in Fig. 6, where the scheduling of four threads on a CPU architecture (top) and GPU architecture (bottom) is shown as an example. Colored boxes are used to identify the different stages of the computation, namely the status of the threads: waiting for data (red), data ready to be processed (orange) and processing of data (green). Although the processing time is higher for GPUs, by continuously switching between the active threads—and thus hiding the

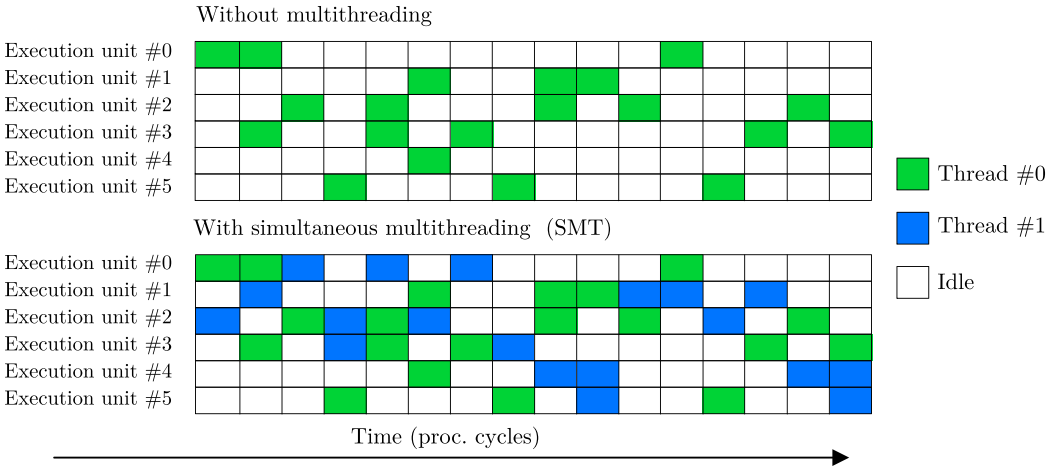


FIG. 5. Sketch showing the SMT concept. We consider the case in which two threads per core are virtualized for each physical core. Each row represents the execution unit for a single cycle: a colored box indicates that the processor has found an instruction to that cycle; an empty box denotes an unused (idle) slot. Using SMT, empty boxes (corresponding, for example, to a situation in which a memory access is required and no instruction can be executed) can be filled executing operations from other threads. This arrangement can operate closer to peak bandwidth, as indicated by the smaller number of idle (white) execution units.

waiting times for data—the overall wall-clock time required for the execution of the four threads is lower than that of CPUs. We remark that the use of thousands of threads per executing unit is not the only factor that can be exploited to enhance the level of performance attainable using GPUs architectures.

Additional factors may help mitigate the memory wall issue. First, the use of high bandwidth memory (HBM): GPUs are often paired with HBM or Graphics Double Data Rate (GDDR)

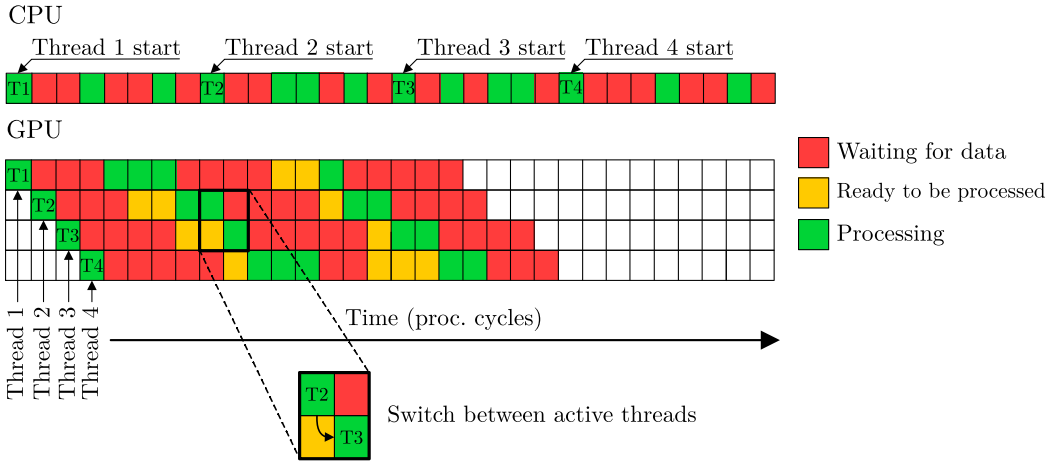


FIG. 6. Sketch showing the general concept behind the use of GPUs for computations. Colors identify the status of the threads: Waiting for data (red), data ready to be processed (orange) and processing of data (green). By continuously switching between the active threads (e.g., $T_2$ and $T_3$ in the close-up), the latency time required to access data can be hidden and the resulting wall-clock time for the computation is shorter for the GPU architecture.

memory. This provides much higher memory bandwidth than standard CPU memory, thus shifting the bandwidth roofline toward higher values; see Fig. 4. Second, GPUs benefit from a carefully designed memory hierarchy that balances throughput, latency, and massive parallelism. Modern GPUs include large, fast caches (L1 and L2) and shared memory, which is often unified with L1 and functions as a programmable, low-latency cache for threads within a block. This allows frequently used data to be accessed quickly and reused efficiently, reducing the use of slower global memory. Differently, CPUs employ complex cache hierarchies optimized for sequential or lightly parallel workloads, with multiple levels of caches designed to minimize latency for individual threads. These differences are also visible in die shots [32]: A GPU dedicates a smaller portion of its die to caches and memory controllers compared to a CPU, reflecting the trade-off between latency optimization and high-throughput parallelism. Third, GPUs rely heavily on shared memory, which is much faster than main memory. Threads within a GPU block can access this shared memory, enabling low-latency data access for algorithms that reuse data (e.g., coalesced memory access), thereby reducing reliance on the slower main memory.

### E. Programming for GPU architectures

From the programming viewpoint, GPUs rely on different sets of instructions compared to standard $\times 86$ architectures. Their use requires specific languages and/or directives implemented in compilers that allow the generation of codes (known as kernels or device codes) that can be executed on GPUs.

The programming framework for GPU computing has advanced significantly since the early 2010s, leading to programming languages or directives characterized by different degrees of portability, adaptability, and performance. The most common are as follows:

(i) Compute Unified Device Architecture (CUDA) [33,34], an extension of Fortran and C++ languages to offload computations on GPUs and to manage memory transfers; (ii) Heterogeneous-compute Interface for Portability [35], a GPU programming framework similar to CUDA but portable across AMD and NVIDIA GPUs; (iii) Open Computing Language (OpenCL) [36], an open, royalty-free standard for cross-platform, parallel programming of different accelerators; (iv) OpenMP [37] and OpenACC [38], directive-based models that use directives to define data movement and computation offloading; (v) Library-based solutions (e.g., Kokkos [39]), which offer high-level abstract programming concepts for performance portability on different types of HPC infrastructures; (vi) oneAPI/SYCL [40,41], a cross-platform programming model that enables accelerator offloading and performance portability using standard C++; (vii) Standard language parallelism, a parallel programming model that allows for accelerating C++ and Fortran codes without using language extensions or additional libraries.

Each of these tools is characterized by its own advantages and disadvantages from the point of view of the performance, portability, maintenance, short- and long-term programming language compatibility [42], compiler requirements, and future developments that one should carefully evaluate before starting the porting of an existing CPU-based code or the development from scratch of a GPU-ready code.

As a final remark, it is important to underline here that additional considerations should be made on the type of profilers and debugging tools available, as they constitute an even more essential instrument (compared to CPU programming) to achieve maximum performance. However, we do not address this specific topics here, as this would be beyond the scope of the present review.

## III. GPU-ACCELERATED DNS AND LES SOLVERS

In this section, we provide a survey of the state-of-the-art numerical methods used for the simulation of turbulence and we discuss their GPU implementation for DNS and LES of incompressible and compressible flows. We consider first applications of DNS (mostly) and LES to both single-phase and multiphase turbulence, concluding with some recent applications of GPU-ready

TABLE I. List of open-source GPU-ready codes for DNS of single-phase incompressible flows.

| Code | Method | Notes |
|---|---|---|
| AFiD [46] | Finite difference | Cylindrical and Cartesian coordinates |
| CaNS [47,48] | Finite difference | Support for scalar transport |
| elbe [49] | Lattice Boltzmann | Support for multiphase and LES |
| Flash-X [50,51] | Finite difference | Level-set and adaptive mesh |
| FLOW36 [52] | Pseudospectral | Scalar transport and multiphase |
| FluidX3D | Lattice Boltzmann | Volume-of-fluid free surface LB |
| Flutas [53] | Finite difference | Scalar transport and multiphase |
| GPUSPH [54] | Smoothed-particle hydrodynamics | Support for multiphase |
| MHIT36 [55,56] | Finite difference | Triply periodic domain |
| Neko [57,58] | Spectral element | Support for scalar transport |
| NekRS [59] | Spectral element | Support for low-Mach and scalar transport |
| nsCouette [60] | Spectral and finite difference | Flow between concentric cylinders |
| Oceananigans.jl [61] | Finite volume | Cartesian and spherical coordinates |
| OpenLB [62] | Lattice Boltzmann | Multiphysics |
| Palabos [63,64] | Lattice Boltzmann | LES support |
| SOD2D [65] | Spectral element | DNS-LES and (in)compressible flow |
| spectralDNS [66] | Pseudospectral | Triply periodic and rectangular domains |
| SU2 [67] | Finite volume or element | Multiphysics and complex geometries |
| waLBerla [68] | Lattice Boltzmann | Support for discrete element method |
| WaterLily.jl [69] | Finite difference | Immersed boundary method and LES |
| Xcompact3d [70] | High-order finite difference | Support for DNS and LES |

LES to atmospheric flows and aerospace problems. For each section, a table that summarizes the available GPU-ready, open-source codes is included. Note that, in the case of incompressible flows, the Navier-Stokes (NS) equations exhibit a highly nonlocal nature associated to the coupling with the mass conservation constraint–zero velocity divergence. Unlike compressible flows, the speed of sound in incompressible NS is considered infinite and so information propagates instantaneously. A common approach for enforcing this constraint is to solve a Poisson equation for the pressure field and then to use the result to project the velocity field into a divergence-free space (fractional method) [43–45]. The global coupling introduced by the Poisson equation represents the main computational challenge in solving the incompressible NS equations, especially in massively parallel frameworks. Any local disturbance in the system, whether due to boundary conditions or changes in the velocity field, needs to be propagated instantaneously across the entire domain. This introduces an unavoidable communication footprint in the algorithm, making the Poisson solver often the most computationally expensive component and challenging part within the core elements of an incompressible DNS solver. The challenge is even greater in multi-GPU configurations, where optimizing collective communication operations is critical. With this in mind, we discuss next applications of GPU-ready DNS to incompressible flows, focusing in particular on open-source solvers.

### A. Direct numerical simulations of incompressible flows

#### *1. Single-phase turbulence*

The most common applications of DNS to single-phase incompressible turbulence refer to canonical flow configurations: Channels, pipes, triperiodic boxes, square ducts, and boundary layers. A list of open-source, GPU-ready DNS codes developed to simulate such flows is provided in Table I.

To integrate the governing equations forward in time, different algorithms are available. One of the most common is the fractional step method (or projection-correction method), which is a popular basis for many open-source and commercial codes [43,71]. The main idea behind this method is to split the advancement from step $n$ to $n+1$ into two substeps. During the first substep (projection), a tentative velocity $\mathbf{u}^*$ is obtained using the momentum equation and neglecting the pressure field contribution (or using the pressure field at time $n$). Then, in the second substep (correction) a Poisson equation for pressure is solved. The resulting pressure field is used to project the tentative velocity into the space of discretely incompressible functions, thereby finding the new velocity. The decoupled equations so obtained can be solved using a number of spatial discretization schemes, including the finite difference method (FDM) [44,72–74], the finite volume method (FVM) [75], the spectral-pseudospectral method (PSM) [76–78], and the finite-spectral element method (FEM-SEM) [79–81]. Two main formulations of the fractional step method can be distinguished: (i) explicit fractional step method, which computes the solution at time $n+1$ using only known values from time $n$ or earlier by means of explicit temporal schemes (e.g., Adams-Bashforth or Runge-Kutta), or (ii) semi-implicit fractional step method, which computes the solution at time $n+1$ by treating some terms (e.g., diffusive or viscous terms) implicitly while the others are treated explicitly. The main advantage of the semi-implicit fractional step method for wall-bounded flows (e.g., channel, pipe flows) is that the implicit treatment of the viscous terms allows one to obtain a stable solution even with large time steps. In fully explicit schemes, the time step is restricted by the small grid sizes near the wall. Owing to its stability and savings in computation time, this method has been widely used for solving the incompressible NS equations [82–85]. When GPUs are used, however, the trade-off between stability and performance is different. Indeed GPUs are optimized for handling independent datasets, which is a characteristic feature of explicit schemes. Therefore, the initial trend in developing GPU-based solvers was to adopt the explicit method for the momentum equation instead of the semi-implicit one [86–93]. Similarly, due to its data independence and ease of parallelization, the Jacobi iterative method [92,93] has been used to solve the Poisson equation on GPUs.

More recently, however, solvers have been developed using a semi-implicit fractional-step method, thus extending to GPUs the use of parallelization techniques initially employed only in CPU-based codes. This is the case, for instance, of the GPU-accelerated FDM codes AFiD [46] and CaNS [48], which report that all-to-all communication accounts for a major portion of the execution time of the flow solver. Communication, particularly inter-GPU communication, has become a major bottleneck in GPU computations [94–96], since GPUs achieve high computational throughput due to the very low overhead associated with thread creation compared to CPU threads [96]. To alleviate this high communication overhead in GPUs, another class of flow solvers [97] adopted the divide-and-conquer algorithm [98–100], an approach originally developed for CPUs to solve tridiagonal systems, and applied it under a one-dimensional (1D) slab-type domain decomposition. However, the restriction imposed by the 1D slab-type decomposition results in low scalability. In the most recent developments, both the all-to-all communication and divide-and-conquer methods have been integrated into a 2D pencil-type domain decomposition to mitigate scalability limitations and communication overhead, demonstrating good scalability up to 1024 GPUs [101]. Further details on all-to-all communications and domain decomposition in the context of the semi-implicit fractional-step method are discussed in Sec. IV.

For PSM-based GPU solvers, fast Fourier transforms (FFT) and all-to-all communications are the dominant operations, and the method provides the highest level of numerical accuracy [102]. Several GPU implementations have been developed [52,103], and as a recent representative example, Yeung *et al.* [104] simulated the PSM at extremely large scales on the *Frontier* exascale supercomputer, employing up to 32 768 GPUs to solve grids of approximately 35 trillion points using 1D and 2D domain decompositions. In this exascale computation, based on a $32\,768^3$ grid, FFT operations accounted for 23.2% and message passing interface (MPI) all-to-all communications for 69.0% of the total runtime on average. In a strong-scaling test with an $8192^3$ grid under a 1D domain decomposition, increasing the number of GPUs from 64 to 512 achieved a parallel efficiency of

84.5% (10.52 s → 1.24 s per time step). In a weak-scaling test with a $512^3$ grid per GPU using a 1D domain decomposition, increasing the number of GPUs from 64 to 32 768 achieved a parallel efficiency of approximately 73.5% (9.62 s → 13.10 s per time step).

While excellent for the simulation of canonical flow geometries, FDM and PSM solvers are not so straightforward to extend to complex geometries. This is where FVM and FEM solvers are most advantageous. Their implementation on unstructured grids is relatively simple and enables efficient discretisation. To take advantage of GPU architectures, significant efforts have been made to extend the use of these methods to GPUs [105–110]. Notably, SEMs have been exploited to extend the range of applicability of FEM solvers and achieve the high-order convergence properties of pseudospectral methods. SEMs originated with the seminal work of Patera [80] and several codes have spawned from this method since then [111–113]. A well known example is Nek5000, a code that has shown excellent scalability up to millions of CPUs. Attempts to port Nek5000, and its electromagnetic counterpart NekCEM, to GPU architectures were initially made starting from the Poisson equation, solved using the conjugate gradient method. Communication among processes, which accounts for most of the computational time, was done using the MPI through an exchange of information at element interfaces, using the gslib library [114]. These attempts were based on the use of OpenACC to accelerate the solver with minimal algorithmic modifications of the code, as documented in several studies [115–120]. While these efforts have led to promising advancements, the achievement of an optimal performance on GPUs still requires, more often than not, substantial refactoring or even partial rewriting of legacy CPU-based codes. This point is also discussed in the ECP report. Moreover, concerns have been raised regarding the performance portability of legacy features in `Fortran 77`, and memory management in OpenACC may also pose issues [57,121].

From these initial efforts, two codes have emerged to target GPUs using the SEM: NekRS [59] and Neko [57,58]. To target portability, NekRS utilizes the Open Concurrent Compute Abstraction (OCCA) library [122], which utilizes just-in-time compilation to translate OKL (OCCA kernel language) code, a modified version of C++, into kernels for a specific architecture. The code has been applied to several canonical flows [123–128], with scaling up to 27 648 Nvidia V100 GPUs for reactor flows with 60 billion grid points [59]. Neko, on the other hand, is written in modern Fortran and allows for efficient implementation of the extensive Nek5000 code base. To interface with GPUs, a device abstraction layer is used in Neko to call platform-specific kernels. Excellent parallel performance has been observed for large-scale problems with billions of grid points [58,129].

Finally, it is worth mentioning alternative methods that do not rely on a direct solution of the NS equations but rather a different set of governing equations that approximate the NS equations, like the lattice-Boltzmann method (LBM) and the smoothed particle hydrodynamics (SPH). LBM and SPH methods have also gained popularity due the absence of a Poisson equation to solve and the relative ease to port these codes to GPUs [130–132]. With advancements in GPU architectures, both methods are increasingly being used for DNS of turbulent flows [133–135]. Specifically, LBM is characterized by inherently local and structured computations, which make it particularly well suited for GPU parallelization, enabling high-performance simulations with fine resolution and large domain sizes. Modern GPU implementations of LBM leverage optimized memory access patterns and massive parallelism to achieve significant speedups compared to CPU-based solvers. Similarly, SPH also allows for substantial improvements on GPUs since neighbor search algorithms and interparticle interactions can be efficiently computed using parallel processing techniques. However, the Lagrangian nature of SPH combined with the irregular data structure and neighbor-list updates require careful optimization to achieve suitable GPU efficiency.

To appreciate the advancements granted by GPU-accelerated DNS solvers of incompressible turbulence, in Fig. 7 we show the time evolution of the highest Reynolds number simulated for different canonical flow configurations since the early 1970s, starting from the seminal work of Orszag and Patterson [136]. This figure shows clearly that the use of GPUs, combined with the development and optimization of tailored flow solvers, has led to a steady increase over the

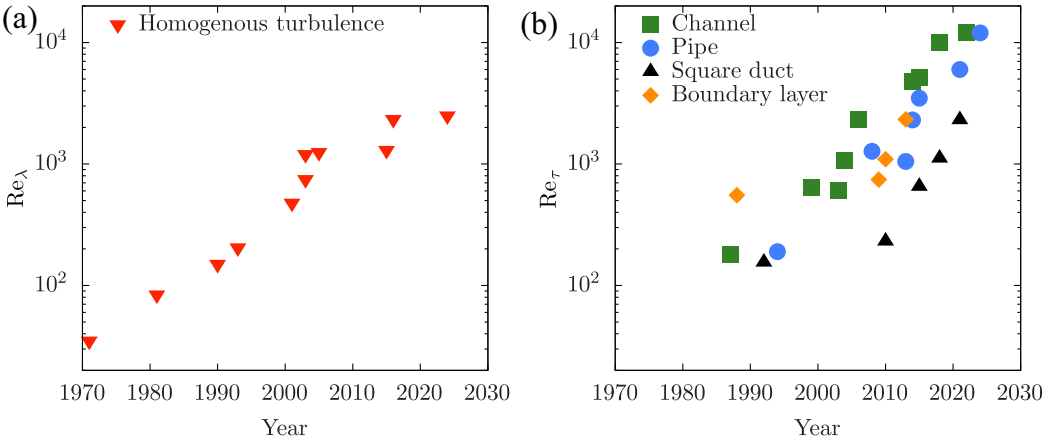


FIG. 7. Evolution of simulated Reynolds numbers using DNS since the early 1970s for homogeneous isotropic turbulence (a) and different instances of wall-bounded flows (b). In panel (a), the reference Reynolds number is computed based on the Taylor microscale ($\mathrm{Re}_\lambda$) in a triply periodic domain. In panel (b), the reference Reynolds number is the friction Reynolds number ($\mathrm{Re}_\tau$) and the trend is shown for different canonical flow configurations: channel flow (green), pipe (blue), square duct (black), and boundary layer (orange). The increase observed since the mid-2010s is largely due to the availability of GPU-ready DNS solvers.

years, stretching the applicability of DNS to higher and higher Reynolds numbers. Examples of applications to canonical flows at such high Reynolds numbers are provided in Fig. 8. The flow fields visualized in this figure highlight the extremely high-scale separation that could be tackled by DNS, as can be appreciated from the zoom-in insets of Fig. 8(a), in particular.

### *2. Multiphase turbulence*

DNS solvers for incompressible multiphase turbulence are an essential tool to investigate a number of complex problems commonly found in environmental, industrial and biomedical applications [141–147]. Many flows of interest are characterized by the interplay of two or more phases, for example a gas or liquid carrier and a dispersed phase constituted by drops, bubbles, particles or fibers. Also common is the presence of complex fluid-structure interactions. Depending on the type of multiphase system considered and the characteristic length scales of the dispersed phase (e.g., the diameter for drops, bubbles, or spherical particles, the length and aspect ratio for elongated particles or fibers), different numerical approaches are available, each characterized by different challenges when porting to GPU architectures.

Assuming that an Eulerian description of the flow field is used, as commonly done in all solvers, it is convenient to classify these approaches based on the mathematical framework that describes the dispersed phase (see Fig. 9): *Lagrangian methods*, in which the behavior of the dispersed phase is described by a system of Lagrangian equations of motion, and *Eulerian methods*, in which the behavior of the dispersed phase is described by solving one or more transport equations on the same (or more refined) grid used to resolve the momentum equation for the fluid. Examples of Lagrangian methods are as follows: Lagrangian particle tracking (LPT), which is typically used to describe the motion of sub-Kolmogorov particles; the immersed boundary method (IBM), in which a set of connected Lagrangian markers is used to model complex boundaries of finite-size objects; and the front-tracking (FT) method, in which a set of linked Lagrangian points is used to describe the motion of deformable interfaces. Examples of Eulerian methods include the following: Eulerian-Eulerian multiphase models, like the lattice-Boltzmann method, in which the dispersed phase is described as a continuous phase, or Eulerian methods for interface-resolved simulations like volume-of-fluid

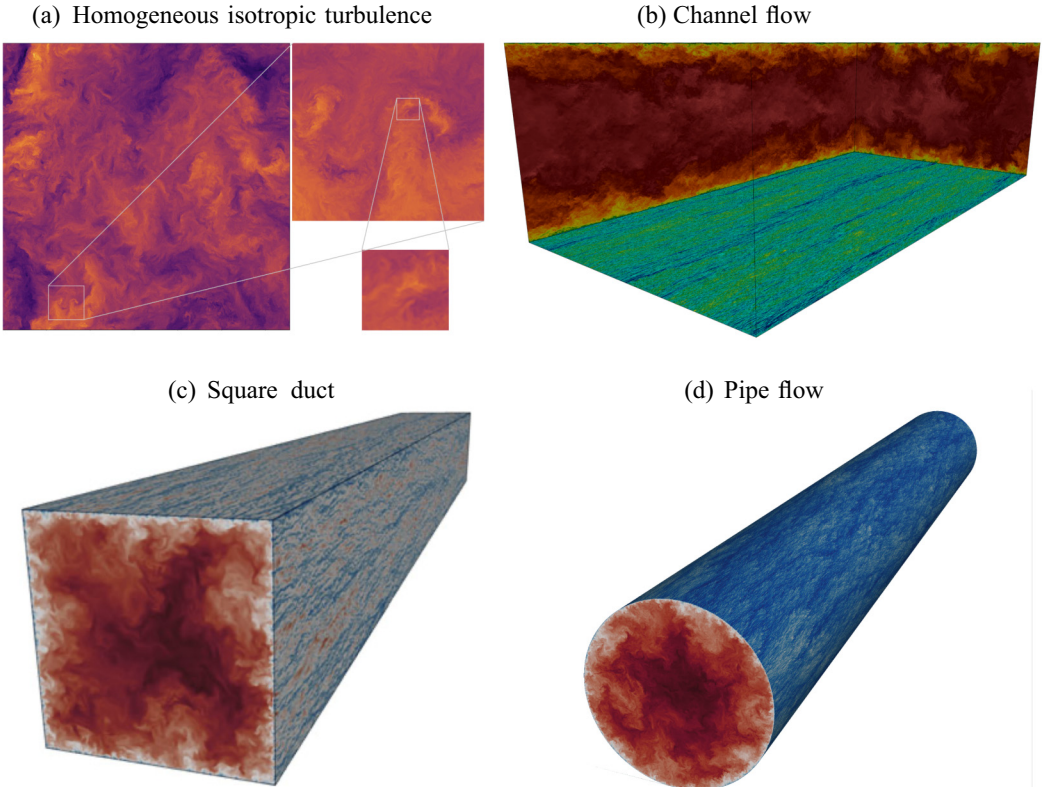


FIG. 8. Visualization of the flow field obtained from GPU-accelerated DNSs of canonical turbulent flows: Homogeneous isotropic turbulence at $\mathrm{Re}_\lambda = 2\,550$ [137,138] (a). Channel flow at $\mathrm{Re}_\tau \simeq 10\,000$ (b). Square duct flow at $\mathrm{Re}_\tau \simeq 2\,000$ [139] (c). Pipe flow at $\mathrm{Re}_\tau \simeq 12\,000$ [140] (d).

(VOF) methods, level-set (LS) methods, and phase-field methods (PFM), in which a scalar indicator (the color function) is used to capture the position of the interface that separates the two phases, as shown in Fig. 10.

Generally speaking, Eulerian methods are somewhat simpler to parallelize and offload to GPU architectures compared to Lagrangian methods. However, it is important to highlight that, when complex interface reconstruction algorithms or nonlinear advection schemes are employed (e.g., using geometrical VoF methods), performance may degrade significantly. The reason behind the simpler porting of Eulerian methods relies on the fact that the parallelization strategy used for the fluid equations can be directly applied to the governing equation of the dispersed phase, which is often represented by a scalar field. In contrast, Lagrangian methods typically require complex communication patterns, leading to significant communication overhead. This overhead hinders code scalability, particularly when using GPUs, and requires the adoption of specialized strategies for load balancing but also for minimizing internode and intranode communication (see Sec. IV C for a more detailed discussion on these issues). The intrinsic challenges posed by an efficient implementation of Lagrangian methods on GPU architectures are reflected in the limited availability of open-source codes for multiphase flow simulations that make use of this approach (see Table II). Indeed, most existing codes rely on interface-capturing methods—such as VoF, LS, PFM, or LBM—to describe the motion of deformable interfaces, e.g., those associated with the presence of drops, bubbles or waves.

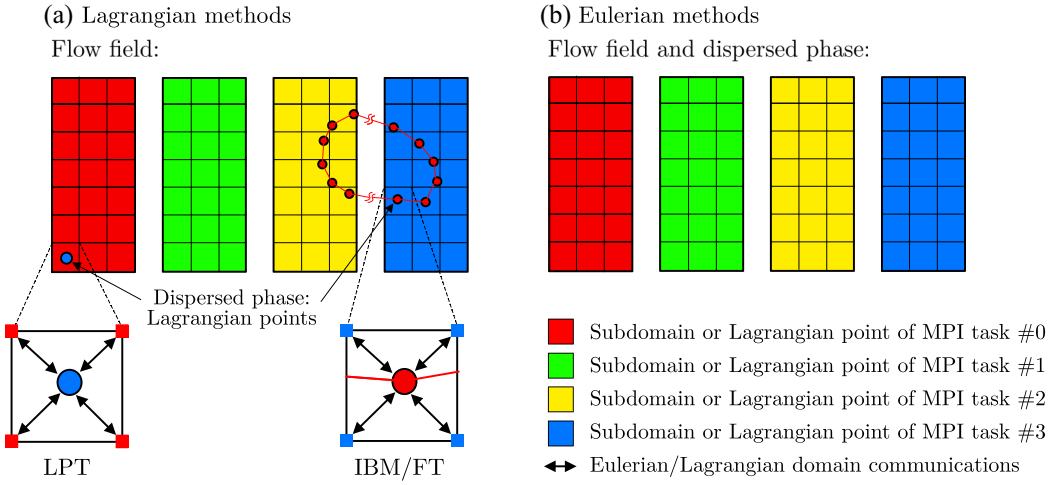


FIG. 9. Conceptual differences between Lagrangian methods (a) and Eulerian methods (b) for multiphase flows. Assuming that the flow field is solved using an Eulerian grid, Lagrangian methods rely on markers (connected or not) for the description of the dispersed phase and communications are needed between the Eulerian grid and the markers to obtain the information required for the interphase coupling. Eulerian methods, on the other hand, rely on a color function (the Eulerian field) to implicitly describe the position and deformation of the interface.

### B. Direct numerical simulations of compressible flows

#### *1. Single-phase turbulence*

Single-phase compressible flows represent another class of problems that is relevant in a number of applications and is very rich in physics, especially in the turbulent regime. Because of this complexity, there are significantly fewer compressible flow codes available compared to the incompressible ones. Computation of single-phase compressible flows avoids the solution of a global pressure Poisson equation and, for large-scale computations on heterogeneous architectures, requires the lowest level of algorithmic redesign effort from existing CPU code bases. The first GPU-accelerated compressible flow simulations, performed by resolving the Euler equations for compressible inviscid flow, could typically achieve an order of magnitude in speedup increase, typically measured by the performance on a single GPU card compared to quad-core CPUs with OpenMP parallelization [160–162]. In these calculations, the task of the CPUs was to preprocess and transfer the data to the GPUs so that the numerical algorithm could be iterated on the GPUs. These early simulations demonstrated the potential for GPU-accelerated computing but were not yet at a readiness level suitable for performing realistic engineering calculations.

Subsequently, the first large-scale DNS of a compressible mixing layer was performed by porting a CPU-based compressible NS solver to GPUs with CUDA [87]. Performance of the single-GPU version was found to be similar to that of the CPU version on 32 cores and statistically steady profiles of the mean velocity and Reynolds stresses showed no visible differences. This simulation represents a milestone for verification and validation, because it demonstrated a way to compare the time-integrated solver output to machine precision. While this can be done routinely in serial-CPU code comparisons (to make sure that the code is both error-free and bug-free), an equivalent tool is not available to the developer in the GPU context, since small perturbations on the order of floating point precision will grow due to the inherently parallel GPU threads.

Following these early efforts, the heterogeneous parallelization landscape has matured and state-of-the-art DNSs are now carried out with GPU-accelerated architectures [91,163,164]. Recent examples of large-scale computations of single-phase compressible flows using GPUs are provided

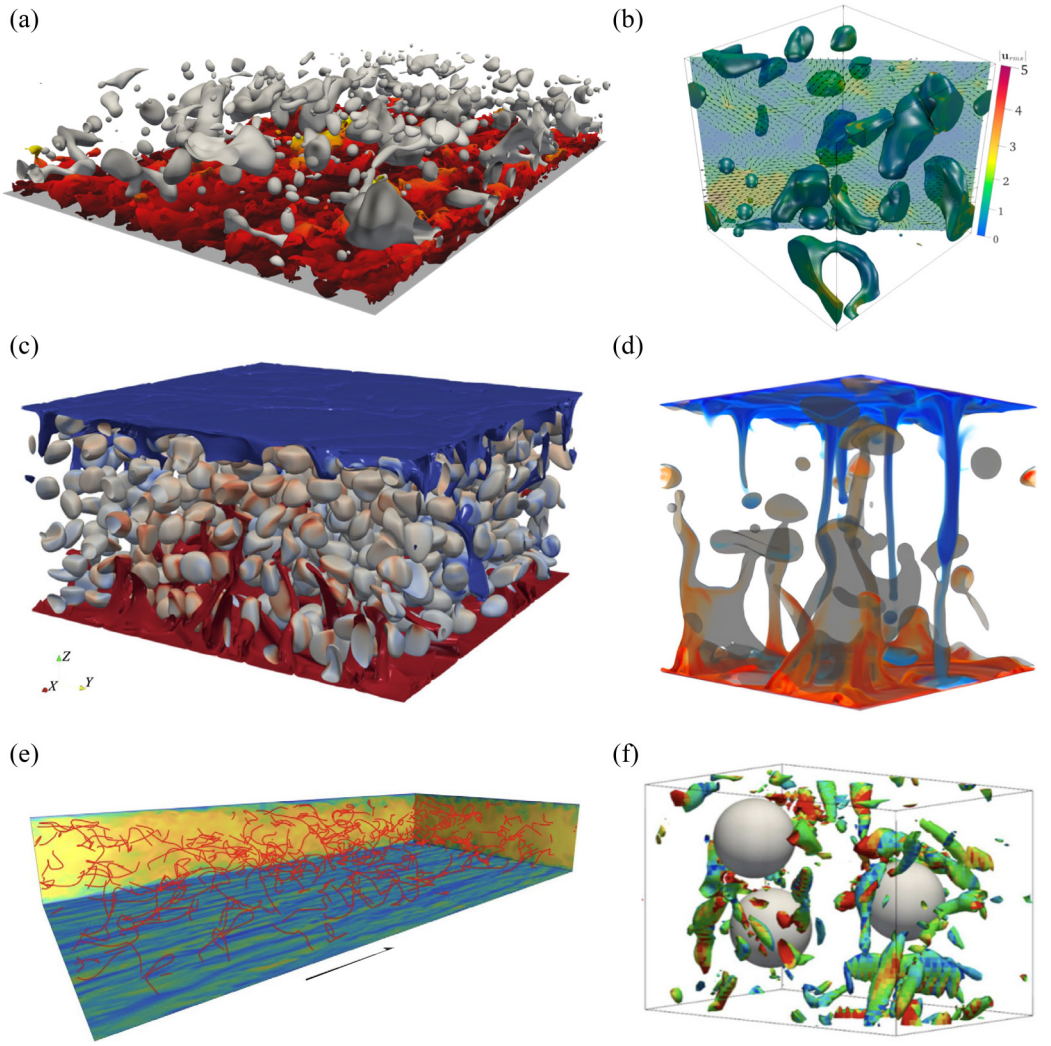


FIG. 10. Visual rendering of multiphase incompressible flows simulated using GPU-accelerated solvers: (a) Surfactant-laden drops in a turbulent channel flow [148]; (b) emulsion in homogeneous isotropic turbulence [149]; heat transfer in Rayleigh-Bénard emulsions at (c) higher volume fraction [150] and (d) lower volume fraction [151]; (e) flexible fibers in turbulent channel flow [152,153]; and (f) fully resolved, finite-size particles in homogeneous isotropic turbulence [154].

in Fig. 11, while a list of open-source GPU-ready DNS solvers is provided in Table III. With the present architectures available, the impact of GPU-accelerated computing in applied calculations is transformative in the size of the computation that can be carried out. The code charLES [165], for example, has performed computation of rotating turbomachinery with a 120 million control volumes with a wall clock time of 346 h on 1024 AMD EPYC 7H12 Rome CPUs and 24 h on 24 Nvidia V100 devices, which represents a dramatic shift in the turnaround time to solution. Clearly, efforts to improve the performance on CPU-GPU heterogeneous architectures are still ongoing. In a recent contribution to HIT, the HyPAR code [166] has been used to target optimization of lexicographic thread configuration, removing data transfer between CPUs and GPUs. The code exploits coalesced memory access using CUDA and has been proven capable of achieving 200× speedup compared to a

TABLE II. List of open-source GPU-ready codes for the direct numerical simulation of multiphase incompressible flows using Eulerian-Lagrangian or Eulerian-Eulerian methodologies.

| Code | Method | Notes |
|---|---|---|
| AFiD-MuRPhFi | Finite difference | Phase-change phenomena |
| Aphros [155] | Finite volume | Multimarker volume-of-fluid |
| Basilisk | Finite volume | Partial support for GPUs |
| CaNS-Fizzy [149] | Finite difference | Volume-of-fluid and phase-field |
| elbe [49] | Lattice Boltzmann | Support for multiphase and LES |
| Flash-X [50,51] | Finite difference | Level-set and adaptive mesh |
| FLOW36 [52] | Pseudospectral | Phase-field method |
| FluidX3D | Lattice Boltzmann | Volume-of-Fluid free surface LB |
| Flutas [53] | Finite difference | Volume-of-fluid method |
| GPUSPH [54] | Smoothed-particle hydrodynamics | Support complex geometries |
| MHIT36 [55,56] | Finite difference | Phase-field method and scalar |
| MFIX-Exa [156] | Finite volume | Discrete element method |
| NaSt3DGPF [157] | Finite difference | Suitable for complicated geometries |
| OpenLB [62] | Lattice Boltzmann | Multiphysics |
| Palabos [63,64] | Lattice Boltzmann | Immersed boundary method |
| PARIS [158] | Finite volume | Only Poisson solver is GPU-ready |
| TLBfind [159] | Lattice Boltzmann | LB for concentrated emulsions |
| waLBerla [68] | Lattice Boltzmann | Support for discrete element method |
| WaterLily.jl [69] | Finite difference | LES support |

CPU code. As far as compressible flows are concerned, it was also found that GPUs are significantly faster than CPUs at performing WENO operations, which represent the most expensive component of the entire calculation.

Another recent application that is worth mentioning involves the Hypersonics Task-based Research (HTR) solver, which has been used to solve supersonic turbulent channel flows, and hypersonic transitional boundary layers of both calorically perfect gases and dissociating air on mesh with up to $2.5 \times 10^9$ grid points on 128 nodes (i.e., 512 GPUs). Note that the HTR solver is built in Legion [169], which automatically manages task scheduling and data movement across heterogeneous architectures. In spite of the progress made in recent years, developing GPU-ready

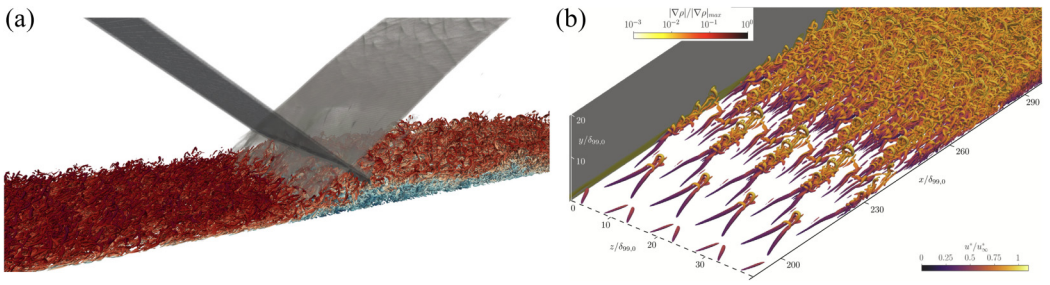


FIG. 11. Panel (a) shows a visualization of supersonic boundary layer interaction topology at free-stream Mach number $M_0 = 2$ and deflection angle $\theta = 10.4°$. The gray isocontour depicts the incident and reflected shock waves, while the coherent vortical structures are rendered using isocontours of the $Q$ criterion, colored by the streamwise velocity, from low values in blue to high values in red. Figure adapted from Ceci *et al.* [167]. Panel (b) shows a flat-plate boundary-layer H-type breakdown. The rendering shows the instantaneous flow structures using isosurfaces of the $Q$ criterion ($Q = 0.015$), colored by the streamwise velocity. The side plane is colored by the normalized density gradient, reproduced from Boldini *et al.* [168].

TABLE III. List of open-source GPU-ready codes for DNS of single-phase compressible flows.

| Code | Method | Notes |
|---|---|---|
| COREFL | Finite difference | Reactive flows and curvilinear coordinates |
| CUBENS [168] | High-order finite-difference | Nonideal fluids |
| Flash-X [50,51] | Finite difference | Level-set and adaptive mesh |
| GALÆXI [173] | Discontinuous Galerkin | Unstructured meshes |
| HTR [90] | Finite difference | Hypersonic aerothermodynamics |
| HyPAR [166] | Finite difference | Unified framework for systems of PDEs |
| JAX-FLUIDS [174] | Finite volume | Diffuse-interface and immersed-boundary |
| NALU | Finite volume | Unstructured grid |
| OpenCFD-SCU [163] | Finite difference | Curvilinear coordinates |
| Parthenon-Hydro [175] | Finite volume | Adaptive mesh refinement |
| PyFR [176] | Flux reconstruction | Curved, mixed and unstructured grids |
| RHEA [177] | Second-order discretization | IBM and LPT support |
| SENSEI [178] | Finite volume | Multiblock structured grid |
| SOD2D [65] | Spectral element | LES support and unstructured grids |
| STREAmS [91,179] | Finite difference | Generalized curvilinear coordinate support |
| URANOS [164,180] | High-order finite difference | Support for LES and wall-modeled LES |
| Xcompact3d [70] | High-order finite difference | Support for DNS and LES |
| ZEFR [181] | Flux reconstruction | Overset grids support |

solvers for compressible flows still poses significant challenges. For instance, efficient implementation of convective flux calculations [170] can become difficult due to memory requirements and efficient MPI communication is crucial for parallel execution [171,172]. These challenges represent clear examples of the level of engagement that GPU programming requires to CFD engineers nowadays.

An intriguing future direction for compressible flow calculations is the use of lower-precision arithmetic to enable faster computations and leverage on consumer-grade GPUs. These formats are significantly cheaper than server-grade components. The development of provable DNS accuracy with reduced floating precision accuracy is a fundamental question that remains wide open for investigation [12,182]. See also Sec. IV for additional discussion on this point. Another area where progress is expected in the near future is compressible flows in complex and varied geometries, the simulation of which requires the adoption of unstructured grids. Many existing GPU-ready solvers are based on structured grids, as these simplify implementation and memory access patterns, but consequently restrict applicability to relatively simple flow domains. Nevertheless, several modern solvers (e.g., NekRS, CHARLESX, and Fun3D) have demonstrated the effectiveness of GPU acceleration on unstructured or spectral element meshes, enabling simulations in more complex geometries. NekRS, for example, achieves significant speedup on GPUs despite the geometrically unstructured mesh (since each element locally retains a dense tensor-product structure) and despite being constrained by unstructured-like memory patterns. Similarly, CharLES scales well and achieves significant increase in throughput [183].

#### *2. Compressible multiphase turbulence*

Multiphase compressible flows are another class of flows that are central in a number of applications. Examples range from cavitation phenomena in hydrofoils [184,185], propellers, and submarines to the breakup of liquid droplets and jets and the erosion of aircraft surfaces during supersonic flight [186]. Generally speaking, performing simulations of compressible multiphase turbulence is a very challenging task because the description of a fixed or moving boundary is also required, in addition to the challenges posed by compressible flows, e.g., those associated to the

TABLE IV. List of open-source GPU-ready codes for direct numerical simulations of compressible two-phase flows using Eulerian-Lagrangian or Eulerian-Eulerian methods.

| Code | Method | Notes |
|---|---|---|
| Flash-X [50,51] | Finite difference | Phase-change phenomena |
| JAX-FLUIDS [174] | Finite volume | Diffuse interface and level set |
| MFC [188,191] | Finite-volume | Multicomponent and multiphase |
| OpenHurricane [192] | Finite volume | Reacting flows |
| Pele [193–195] | Finite-volume | Extensions for different physics are available |
| RHEA [177] | Second-order discretization | IBM and LPT support |
| STREAmS [91] | Finite-difference | Immersed-boundary method |

description of discontinuities or to the appearance of negative volume fractions due to accumulation of numerical errors when the total volume fraction of the different phase is not equal to unity [187]. This task is typically achieved using the same methodologies adopted for incompressible multiphase turbulence, immersed boundary methods or interface tracking or capturing methods being the most popular choice. The extension of these methods to compressible flows is not straightforward since the flow discontinuities can interact with the solid boundary of the dispersed particles in gas-solid flows or with the deformable interface of bubbles and drops in liquid-gas or liquid-liquid flows. To capture these interactions with suitable accuracy, robust numerical methods that maintain discrete conservation, suppress oscillations near discontinuities, and preserve numerical stability are required [188]. Such strict requirements on the numerical method have a negative effect on the porting of compressible multiphase solvers to GPUs. In fact, only a few numerical methods are suitable for this type of flow and implicit schemes are usually employed to deal with the stiffness of the governing equations to be solved and thus stabilize the simulations.

The need to use implicit schemes or multigrid iterative methods may further penalize the efficiency of the adoption to GPUs architectures due the additional communication overhead. Additionally, it is worth mentioning that these codes usually employ adaptive mesh refinement (AMR) techniques. Therefore, if AMR libraries are employed, then these libraries should also be GPU-ready in order to avoid bottlenecks. An example of a GPU-ready AMR library is AMRex [189,190], the development and GPU-porting of which was supported by ECP. Last, a further issue may arise when different numerical schemes are used in the vicinity of discontinuities. The switch from one scheme to the other often requires a change in the discretization to be used. When this transition is handled through an `if-else` branching condition, thread or warp divergence may occur. The divergence stems from the fact that threads within the same block execute different instruction paths, thereby hindering efficient parallel execution on GPU architectures. This issue is directly related to the SIMT execution model used by GPUs.

A list of open-source codes for direct numerical simulations of compressible multiphase flows is reported in Table IV. The most commonly employed are STREAmS [91], MFC (Gordon Bell Prize finalist 2025) [188,191] and PeleC [193–195]. These codes result from a number of joint optimization efforts that have been carried out over the last decade or so. Notably, STREAmS exploits the immersed boundary method to model the presence of solid boundaries [196]. An example of application of STREAmS is provided in Fig. 12(a). The MFC code, instead, is based on an interface capturing approach to describe the deformable interface of drops and bubbles, as shown for example in Fig. 12(b). Finally, Pele [193–195] is a suite of adaptive mesh hydrodynamics simulation codes, the development of which has been supported by ECP for more than seven years, that relies on the AMRex library [189,190]. Pele can be used to simulate turbulent compressible reacting flows using advection-diffusion equations complemented by sink or source terms. Other GPU-ready simulation tools that can be used to investigate compressible reacting flows are OpenHurricane [192], S3D (not open source) [198,199], and COREFL.

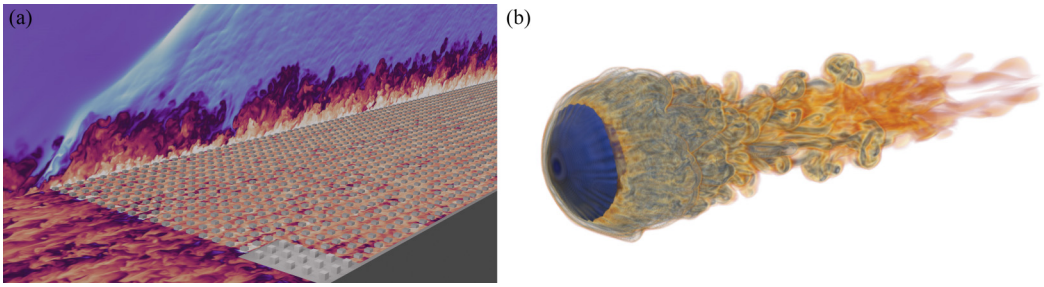


FIG. 12. (a) Instantaneous flow visualization from DNS of a supersonic turbulent boundary layers over rough surfaces [197], simulated using the open-source code STREAmS [91]. (b) Interaction of a water droplet with a shock wave, simulated using the open-source code MFC [188,191].

Moving to multiphase compressible solvers that rely on Lagrangian methods, Liao *et al.* [200] propose a GPU-accelerated implementation for investigating canonical compressible particle-laden wall turbulence based on four-way coupled simulations. To overcome the prohibitively high computational cost and facilitate the four-way coupled PP-DNS, the solver leverages advanced GPU-CPU heterogeneous computing techniques. Specifically, a spatial subdivision algorithm is used to efficiently compute interparticle collisions on multi-GPU platforms. To avoid frequent dynamic memory allocation or compaction manipulation, a specialized data structure for particle variables was designed to mimic the page table in computer operating systems.

### C. Large-eddy simulations

LES has become an important design tool in many engineering processes, where traditional Reynolds-averaged Navier-Stokes models fall short. At the same time, LES has gained popularity as an alternative to DNS by offering a reasonable balance between computational cost and predictive accuracy, particularly for complex aerodynamic and hydrodynamic applications [201]. For these reasons, there is a high demand for GPU-accelerated LES solvers, especially for simulating incompressible flows, advancing ML-based turbulence modeling and performing grid convergence studies [201]. In many ways, the development of such solvers poses a number of advantages and challenges that are similar to those already described for DNS, because the core numerical discretization is often the same between the two approaches. Therefore, strategies for GPU-CPU communication, domain decomposition, load balancing, and memory management are shared between DNS and LES implementations. This is because many of the numerical challenges are shared across hardware platforms. For example, in spectral element methods, the $C^0$ continuity of the solution [202] affects the computation of the strain-rate tensor in LES, but this issue is inherent to the discretization and is largely independent of whether the solver targets CPUs or GPUs. Nevertheless, there are key differences that have important implications for GPU computing. A first difference is represented by the filtering stage, which is intrinsic in LES. A second difference is associated with the additional modeling of the flow physics that LES computations require since they are often more applied than DNS in nature (e.g., in the fields of atmospheric sciences or combustion engineering).

One class of LES solvers that has seen significant progress in porting to GPU architectures is related to the field of atmospheric sciences, where problems of increasing size and complexity require strategies for accelerated computation. Indeed, computing centers dedicated to the study of problems in the broad area of Earth sciences can increasingly rely on clusters equipped with a growing number of GPUs [203]. Applications of GPU-ready LES to atmospheric flows involve a number of numerical methods and programming strategies but generally require the solution of the incompressible NS equations using the Boussinesq approximation or the anelastic approximation (which was introduced to describe the motion of a thermally stratified fluid by filtering sound waves without assuming hydrostatic balance) [204]. Such solvers are meant to simulate turbulence in the

planetary (and possibly also oceanic) boundary layer. Most of them are also capable of performing DNS with minimal changes to the code configuration.

Unlike many of the engineering applications described previously, LES codes often incorporate families of physics models attempting to describe complex phenomena such as land-water surface interaction, cloud physics, and/or atmospheric radiation. Not surprisingly, these models can become quite complex and may require linking to external, public domain packages, which themselves would need to be GPU capable in order to not hinder GPU development and exploit the full potential of the simulations. For example, the GPU-resident atmospheric large-eddy simulation (GALES) model was first written in CUDA for a single GPU device [205] and then expanded to enable high performance across multiple GPUs. This expansion has enabled either running on very large domains (e.g., LES of the atmospheric boundary layer over a computational domain covering the entire country of the Netherlands at a 100 $m$ grid resolution [206]) or running over previously inaccessible simulation time spans [207]. An expanded model (known as GRASP: GPU Resident Atmospheric Simulation Program) has evolved further in recent years and can now be used to study problems such as wind energy and pollutant dispersion. Similarly, other GPU-resident models have been developed to perform LES of the atmospheric boundary layer. One is the microHH model, which is easily configured to perform incompressible DNS. The first version of the code could run on a single GPU device and compute problems typically restricted to MPI-based CPU clusters on a single desktop [208]. Since these early applications, the code has hugely expanded to multi-GPU capabilities. Likewise, the FastEddy model [209] is a GPU-resident model written in CUDA that has been developed to efficiently carry out atmospheric LES.

While the GPU-resident models just described required writing a new code from scratch, others followed the acceleration approach, which uses openACC to send portions of an existing CPU code to the GPU devices. One example is provided by the PALM model [210], which can perform LES and DNS in multiple configurations. The atmospheric CM1 model is in the early stages of GPU acceleration via OpenACC [211] as is the DALES model [212]. While the openACC framework allows one to send specific portions of the code to the GPU devices, it is always possible (with sufficient effort) to make the code GPU-resident as well. Moving to more aerospace-oriented applications, NASA has made significant strides through its $T^3$ initiative. A notable outcome is the Launch Ascent and Vehicle Aerodynamics (LAVA) solver [213,214], an MPI-based code with support for GPU acceleration that will soon be publicly released. LAVA supports a wide range of simulations—from LES and wall-modeled large eddy-eddy simulations (WMLES) to hybrid RANS-LES—across Cartesian, curvilinear, and unstructured grids. Similar GPU porting efforts have been completed or are underway for other NASA codes such as rotLES [215], OVERFLOW [216], and GFR [218]. Some of these codes are open source (e.g., GFR and LAVA) while others are not. Four examples of turbulent flow simulations performed using GPU-ready LES codes are shown in Fig. 13. A list of open-source codes for LESs of single-phase and multiphase flows is reported in Table V.

## IV. MAIN CHALLENGES AND ISSUES

In this section, we discuss the main challenges posed by the porting to GPUs of DNS and LES codes. In particular, we address the main performance bottlenecks associated with the porting, both from a general perspective and from a more application-oriented viewpoint (single-phase vs multiphase flows).

### A. General bottlenecks

A common bottleneck—shared by many applications—is the communication, either from CPU to GPU, or even directly between GPU devices. Solutions are often tailored to the specific numerical technique and domain decomposition employed by the solver, yet one broad strategy is to reduce the communication overhead by overlapping the communication operations with the computation

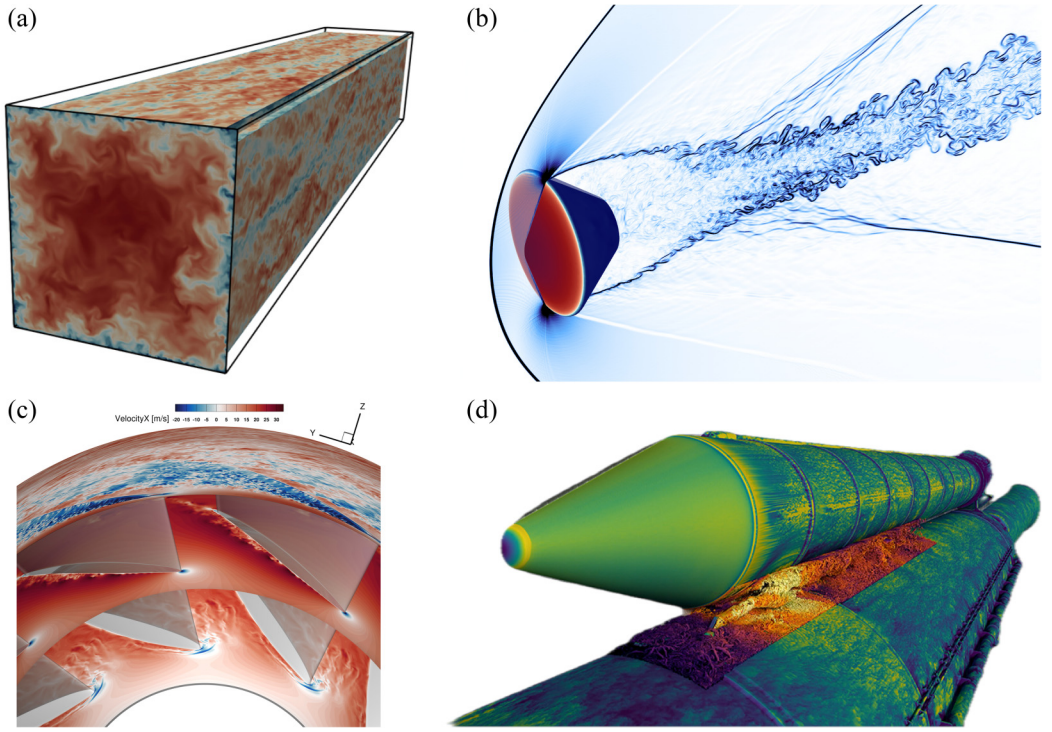


FIG. 13. Examples of GPU-based LES of turbulent flows. (a) Flow in a square duct at bulk Reynolds number $\mathrm{Re}_b = 40\,000$. The simulation employs a WMLES strategy and was performed using CaLES [201]. (b) Unsteady supersonic flow around a capsule during its re-entry in the Mars atmosphere, adapted from Ref. [217]. The simulation was carried out at Mach number $\mathrm{Ma}_\infty = 2$ and Reynolds number $\mathrm{Re}_\infty = 10^6$, based on the free-stream velocity. (c) Clockwise-rotating rotor blades interacting with a freestream flow directed into the page. Instantaneous contours of axial velocity are shown along surfaces at the bottom, middle, and top of the passage height [218]. (d) Space Launch System rocket during ascent, viewed from above. Friction forces on the rocket are shown in green, yellow, and blue. Airflow-induced vibrations around the strakes flanking each booster's forward connection point on the intertank are visualized in purple, yellow, and red [219].

operations. An example of this approach can be found in a GPU-based Poisson solver using the Jacobi iteration method, where communication-computation overlapping is exploited to improve performance [223]. The procedure is illustrated in Fig. 14. First, the boundary data are updated. Subsequently, these data are transferred as halo (ghost) cell data to neighboring processors, concurrently updating the inner cell data. This process is repeated until convergence is achieved. A similar strategy has also been applied to FFT executions in PSM [103], where the computational domain is divided into subdomains and the all-to-all communication associated with one subdomain is overlapped with the FFT executions on another subdomain.

The available GPU memory is also an important constraint in large-scale simulations. To reduce memory usage, a naive solution is to allocate and deallocate variables at each time step; however, this approach results in considerable time overhead during computations. As an alternative, memory reuse through shared workspaces implemented with pointers has been employed to alleviate GPU memory pressure [46,97]. Furthermore, offloading statistical computations to CPUs and performing them in parallel has been shown to further reduce GPU memory requirements [97]. More general methods for optimizing communication and memory usage, particularly at the GPU-architecture level, are nicely summarized in Table I of the review paper by Niemeyer *et al.* [31] and in the CUDA programming guides [34,95]. In the next sections, we examine how the network and

TABLE V. Open-source GPU-ready codes for large-eddy simulations of turbulent flows.

| Code | Method | Notes |
|---|---|---|
| CaLES [201] | Finite difference | Based on CaNS |
| CM1 [211,220] | Finite difference | Only portions GPU capable |
| DALES [212,221] | Finite difference | Atmospheric flow |
| elbe [49] | Lattice Boltzmann | Support for multiphase and LES |
| FastEddy [209] | Finite difference | Compressible |
| GALÆXI [173] | Discontinuous Galerkin | Unstructured meshes |
| GALES [205–207] | Finite difference | Based on DALES |
| GFR [218] | Flux reconstruction | Tailored for complex aeropropulsion flows |
| LAVA [213] | Finite volumes | Support for different mesh types |
| LESGO [222] | Pseudospectral | Atmospheric flows (canopies, wind turbines) |
| MicroHH [208] | Finite difference | Atmospheric boundary layer flows |
| Neko [57] | Spectral element | DNS support |
| NekRS [59] | Spectral element | DNS and RANS support |
| Oceananigans.jl [61] | Finite volume | Cartesian and spherical domains |
| PALM [210] | Finite difference | Atmospheric and oceanic flows |
| SOD2D [65] | Spectral element | Support for DNS |
| URANOS [164,180] | High-order finite difference | Support for DNS and WMLES |
| Xcompact3d [70] | High-order finite difference | GPU-ready version is called x3d2 |

memory constraints just discussed can hamper code performance in the context of single-phase and multiphase flow applications.

### B. Specific bottlenecks for single-phase flows

The main bottlenecks that affect the simulation of single-phase flows are associated with MPI communications and use of 2D and 3D FFTs. Before delving into these bottlenecks, however, it is essential to understand how domain decomposition (or partitioning) is typically implemented for these types of flows and what kind of communication among processor such decomposition requires.

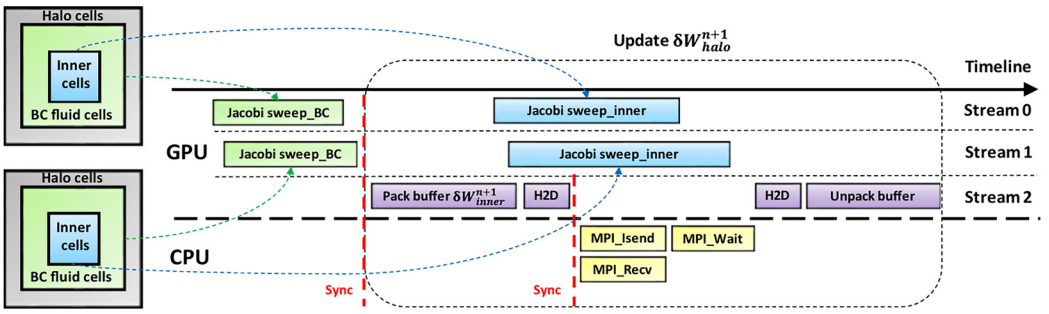


FIG. 14. A computation-communication overlap strategy was applied to the Jacobi iteration method [223] to hide inter-GPU communication latency, which occurs through CPU intermediates (GPU → CPU, CPU → CPU, and CPU → GPU transfers) and constitutes a significant portion of the total runtime. Although a similar approach has been used on CPU architecture, it becomes particularly critical on heterogeneous architecture due to their high communication portion. Specifically, in this strategy, boundary data are updated first and transferred to neighboring processors as halo (ghost) cells, while the Jacobi iterations for the inner cells proceed concurrently during the data transfer. Reproduced from Nguyen *et al.* [223].

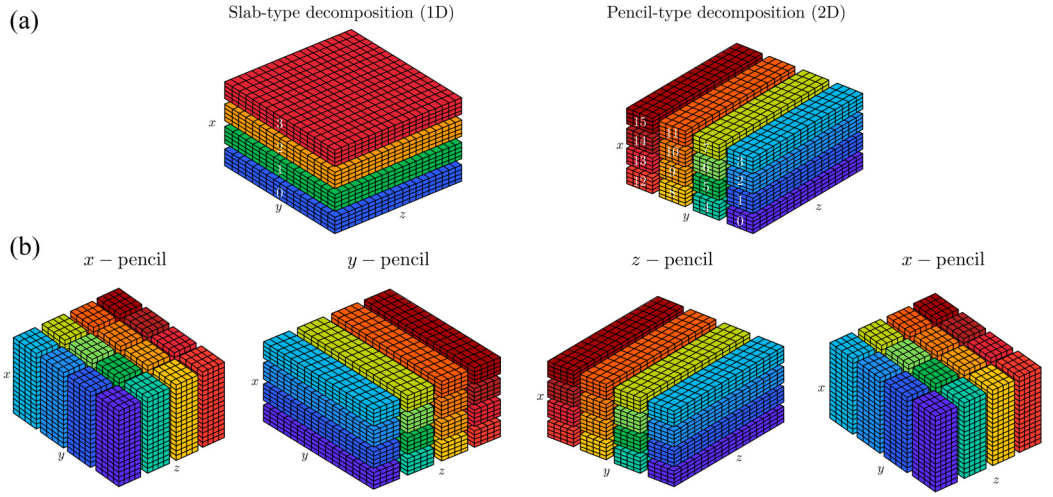


FIG. 15. (a) Example of 1D slab-type decomposition (left) and 2D pencil-type decomposition (right). (b) Global data transposition requiring all-to-all communication. The direction of the pencil-type domain changes three times from left to right: $x \rightarrow y \rightarrow z \rightarrow x$.

Many well-known multiprocessor (GPU- or CPU-based) solvers for incompressible flow are based on Cartesian grids. Since each processor is responsible for a portion of the entire computational domain, partitioning the domain along specific Cartesian directions is a necessary step. Domain partitioning is typically implemented as 1D or 2D segments, also known as slab and pencil decompositions, respectively. An example of these types of decomposition is given in Fig. 15(a). All-to-all communication among processors occurs when the direction of the partitioned domain is changed for global data transposition. For example, all-to-all communication is required to convert $x$-direction pencils into $y$-direction pencils, as shown in Fig. 15(b). This communication step is necessary to align data along specific directions within the same processor, ensuring data locality (i.e., the required calculation data resides on the same processor) for operations such as FFT and tridiagonal matrix algorithm (TDMA). All-to-all communication is a well-known bottleneck in incompressible flow simulations [104], affecting not only recent GPU-based codes (e.g., AFiD [46] and CaNS [48]) but also multi-CPU implementations [224]. This type of communication is unavoidable for implementing the semi-implicit fractional-step method in GPU codes. In particular, all-to-all communication is necessary during the projection step to execute the TDMA along the three directions. Likewise, all-to-all communication is required in the solution of the Poisson equation used in a pressure-correction step. In this context, an FFT is typically used in two directions and a TDMA in the third direction. Since all-to-all communication is a primary computational bottleneck, optimizing this aspect is crucial for obtaining efficient GPU implementations. One of the GPU-ready incompressible flow solvers, AFiD [46], circumvents this issue in the momentum equation by implicitly solving only in the wall-normal direction for turbulent channel flow. However, this approach is feasible at high Reynolds numbers, since explicit solutions in the streamwise and spanwise directions do not restrict the time-step size. Another GPU-ready incompressible flow solver, CaNS [48], provides an option for solving the momentum equation either implicitly or explicitly.

The bottleneck introduced by all-to-all communication—required to solve the Poisson equation using FFT-based methods and/or to perform implicit integration of the viscous term along the wall-normal direction—can be appreciated by examining the strong scaling results shown in Fig. 16. The results refer to different open-source codes: CaNS, CaLES, and MHIT36. Deviations from ideal scaling are more pronounced when fully explicit time-integration schemes are employed

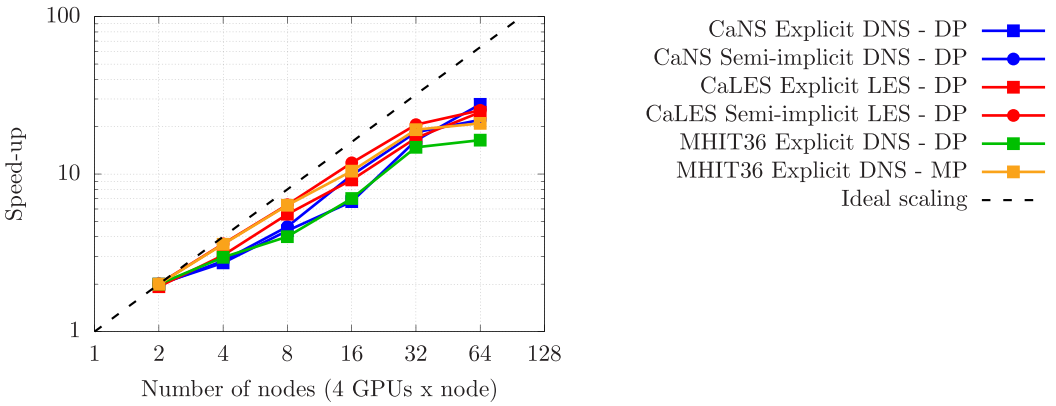


FIG. 16. Strong scaling results obtained from CaNS, CaLES, and MHIT36 for the solution of single-phase flow (DNS and LES). Different treatments of the viscous terms were tested: Implicit integration along the wall-normal direction (semi-implicit) or fully explicit. MHIT36 has been tested in full double-precision (DP) and mixed-precision where the Poisson solver is executed in single-precision mode (FP32). The problem size considered is $1440 \times 512 \times 384$ for CaNS and CaLES, $1024^3$ for MHIT36. All tests were performed on the Leonardo supercomputer ($4 \times$ Nvidia A100 GPU per node) [225], starting from two nodes to remove the influence of fast intranode connection (NVLink) on the results.

(squares). Since the computational cost of evaluating the right-hand side of the NS equations is relatively small, the all-to-all communication overhead in the Poisson solver becomes significant and affects scalability. This bottleneck has a smaller effect when implicit integration schemes are used for the viscous terms along the wall-normal direction, namely when the semi-implicit scheme is used. Although this scheme requires additional all-to-all communication, the computational effort needed to solve extra systems of equations (one for each velocity component) helps offset the communication overhead. Similarly, the additional computations required by the LES approach (CaLES code) result in slightly improved scalability.

Several studies have explored methods for performing TDMA without adhering to strict data locality (i.e., without requiring all data needed for the computation to reside on the same process). This is typically achieved using a divide-and-conquer strategy [98–100], referred to hereafter as multiprocessor TDMA. Within a slab-type domain decomposition, the multiprocessor TDMA proposed by Sun [98] was integrated into an alternating-direction implicit method [97], enabling the implicit solution of the momentum equations while avoiding all-to-all communication. This strategy is also extended to the pressure Poisson equation to eliminate expensive all-to-all communication operations. Their results showed that, compared to the conventional all-to-all communication method, the multiprocessor TDMA achieved a speedup of 52.6% on four V100 GPUs and 65.6% on eight V 100 GPUs. In pencil-type domain decompositions, which are more suitable for extreme-scale simulations, multiprocessor TDMA can be applied along one direction while being combined with all-to-all communication in the other directions [101,226]. When solving the Poisson equation on 32 to 1024 A100 GPUs, this approach achieved approximately a twofold performance improvement compared to conventional all-to-all communication strategies [101].

Outside the context of the semi-implicit method, which is mainly applied in conjunction with FDM and FVM, all-to-all communication represents a major bottleneck also in codes that do not rely on this type of scheme. In particular, DNS codes based on the use of pseudospectral methods are also affected significantly by all-to-all communication, which becomes necessary because the nonlinear terms of the governing equations are evaluated in the physical space to avoid costly convolution operations in the spectral space [227]. This procedure requires forward and backward transforms as well as pencil transpositions, similarly to what happens when the Poisson pressure equation is

solved in the context of the fractional step methods using FFT-based algorithms. As the number of pencil transpositions (and thus all-to-all communication) per time step is larger with respect to the semi-implicit fractional step method (a back-and-forth transform and loop of pencil transposition are required for each nonlinear term that has to be evaluated), this type of operation must be heavily optimized in order not to hinder the strong scaling performance of the code [103,104,228]. To this aim, optimized libraries for multi-GPU FFTs or for pencil transpositions have been developed over the years and are now available [229–232].

#### *1. Runtime performance autotuning*

As already mentioned, efficient inter-GPU communication is a key factor for obtaining efficient methods for large-scale simulations. At present, there are numerous libraries available for data communication among GPUs, targeting different hardware and software ecosystems. In addition to the several MPI implementations commonly deployed on the largest GPU systems, vendors have developed specialized GPU-specific communication libraries to optimize the performance of their hardware. For instance, NVIDIA provides the NVIDIA Collective Communication Library, which implements multi-GPU and multinode communication primitives optimized for NVIDIA GPUs and networking, and NVSHMEM, which is parallel programming interface based on OpenSHMEM shared-memory routines. Similarly, AMD has introduced the ROCm libraries such as ROCm Communication Collectives Library, which is a stand-alone library that provides multi-GPU and multinode collective communication primitives optimized for AMD GPUs. Moreover, the low-level implementation of collective all-to-all communication operations is not unique, each with distinct trade-offs in terms of latency and bandwidth utilization [233,234]. Finally, even for the same GPU vendor, the intra- and internode topology of supercomputers can vary significantly, adding another layer of complexity to achieving optimal communication performance.

With such a diverse set of options, determining the optimal combination of domain partitioning and communication backend is a nontrivial task, as it depends heavily on the specific hardware architecture, software stack, and workload characteristics of each system. One promising strategy to address this task is to resort to an automatic performance tuning (autotuning, in short) framework that dynamically profiles and selects the most suitable communication backend and domain decomposition. This is done using a low-level implementation that maximizes the utilization of resources. Autotuning for GPUs is a very practical solution to easily port existing algorithmic solutions on quickly evolving GPU architectures and to substantially speed up even highly hand-tuned kernels. In particular, autotuning during runtime offers the key advantage of adapting workload performance to the unique characteristics of each system. Implementing from scratch libraries with built-in runtime autotuning from scratch is often impractical for domain scientists, therefore open-source libraries with runtime support functions are recently emerging. One example is the cuDecomp library [233] provides autotuned communication capabilities for pencil-based computational codes.

#### *2. DNS in single and mixed precision*

Another possible route to reduce the communication footprint is to lower the numerical precision of the whole workload or just selected components of it. For example, one may replace double precision with single or mixed precision [12,182,235]. This shift towards lower precision arithmetic is justified not only by the reduction of required memory bandwidth and computational cost but also by energy efficiency arguments [236–238]. The energy used for a FLOP in single-precision (FP32) is about 2.5 to 4 times smaller—with variations depending on the architecture—than the equivalent energy used when working in double precision (FP64) [239,240]. This is a direct consequence of the fact that GPUs are massively optimized for operations on low-precision floats (FP8, FP16, and FP32). It can be anticipated that the capacity of a numerical method or scheme to work using low-precision floats may become a key factor in future systems where the use of energy-efficient simulation tools will become critical [8]. In light of this, many mathematical libraries have been adapted in recent years (e.g., during ECP) to work with mixed-precision arithmetic [240,241].

In the context of incompressible single-phase flow, a first approach for exploring the applicability of the strategy just discussed to NS solvers is rather simple: Exploit mixed precision to solve the Poisson equation. More specifically, one can (i) downcast the right-hand side of the Poisson equation to single precision, (ii) solve the Poisson equation in single precision, and, finally, (iii) upcast the resulting solution back to double precision. Beyond the reduced computational cost of lower-precision floating-point arithmetic, the amount of communicated data among CPUs or GPUs is effectively *halved*, thus mitigating significantly the total communication overhead. When the Poisson equation is solved using a direct linear solver—as done in many finite-difference DNS and LES solvers discussed in this paper—round-off error is the only source of error in the linear system. Thus, lowering the precision in the Poisson solver is analogous to prescribing a looser tolerance in an iterative double-precision solver, where the discrete Poisson equation is not necessarily satisfied to machine precision. Indeed, it is common practice in DNS and LES to terminate the Poisson solve when the residual norm is reduced below $10^{-6}$ (relative or absolute), without degrading accuracy.

To quantify the effects of mixed precision on the performance of a flow solver, we performed turbulent channel flow simulations at $\mathrm{Re}_\tau \approx 1000$ using the CaNS solver in (i) double precision (DP) only; (ii) mixed precision (MP), namely Poisson solver in single precision (SP) and the remainder of the numerical calculation in DP; and (iii) SP only. The domain considered in all cases has size $2\pi h \times \pi h \times 2h$ and is periodic in the streamwise ($x$) and spanwise ($z$) directions, with no-slip and no-penetration conditions imposed at the walls. A constant mass flux drives the flow at bulk Reynolds number $\mathrm{Re}_b = 2hU_b/\nu = 39\,600$, based on the full channel height, as in Bernardini *et al.* [242]. The grid ensures inner-scaled wall-parallel spacings of $\Delta x^+ \approx 9$ and $\Delta z^+ \approx 5$. Along the wall-normal direction $y$, the natural grid-stretching function is employed [3]. In this setup, the mass leakage in a grid cell, relative to the mass flux associated with bulk velocity $U_b$, is given by

$$\frac{|\nabla \cdot \mathbf{u}_{ijk}|\Delta x \Delta y(j) \Delta z}{U_b \Delta y(j) \Delta z} = \frac{\nabla \cdot \mathbf{u}_{ijk} \Delta x}{U_b}, \tag{2}$$

where $\Delta y(j)$ is the wall-normal grid spacing of the $j$th computational cell. Similarly, the total mass-leakage error, $I_{\mathcal{D}}$, is obtained by integrating the divergence over the domain and comparing it to the prescribed flow rate. Exploiting flow symmetry, after integrating and time-averaging, the absolute divergence field yields:

$$I_{\mathcal{D}} = \frac{Q_{\mathrm{leak}}}{Q_b} = \frac{1}{U_b h} \int_0^h |\nabla \cdot \mathbf{u}| \, \mathrm{d}y. \tag{3}$$

The normalized grid-cell leakage profiles obtained with DP and MP, as well as the the mass leakage in a grid cell, are shown in Fig. 17, together with the mean velocity and rms of the fluid velocity vector. The top panels of this figure confirm the minimal impact of lower precision on mass conservation. As expected, both DP and MP yield very small mass conservation errors. The bottom panels show even smaller, hardly noticeable, differences on single-point statistics between DP and MP results. We also remark that, although not shown here, even fully SP simulations accurately reproduced these statistics, with mass-conservation errors comparable to those obtained with MP.

In contrast with the low impact of precision on the quality of the solution, the speedup observed with SP and MP at scale is significant. Tests on the EuroHPC supercomputer MeluXina showed that MP produced a 1.5 up to 1.9 speedup for the full NS solver at scale, while fully SP only yielded an increase of speedup of 10% up to 20% over MP. This is expected, as gains in performance arise primarily from the drastic reduction of the communicated data volume in the Poisson solver rather than from the lower cost per operation granted by reduced precision.

Beyond incompressible single-phase flow, mixed half- and single-precision storage and computation have also been successfully leveraged in the context of compressible flow solvers. One remarkable example is provided by the simulations reported in Wilfong *et al.* [243]. By optimizing the MFC code and exploiting half- and single-precision arithmetic, the authors performed simulations of 33 thrusters in a configuration inspired by the SpaceX Super Heavy booster. The computational grid comprises 200 trillion points, leading to a number of degrees of freedom that

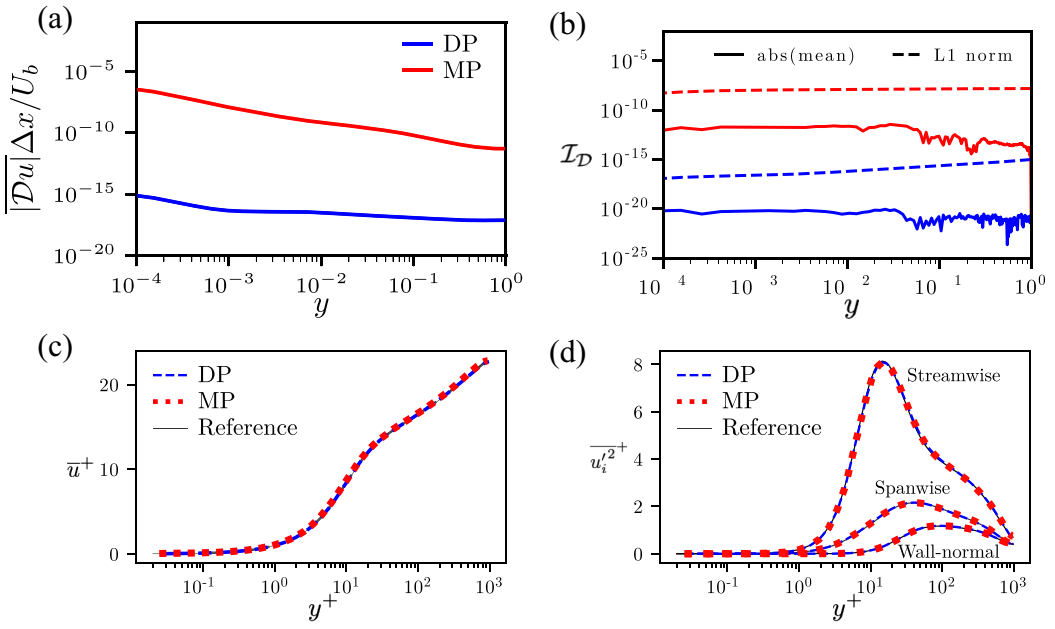


FIG. 17. Comparison of results obtained using CaNS in double precision (DP) and mixed precision (MP). When MP is used, the Poisson solver is executed in SP while all the remaining computations are exectuted in DP. Panel (a) shows the normalized grid-cell leakage profile with $\mathcal{D}u = \nabla \cdot \mathbf{u}_{ijk}$. Panel (b) shows the cumulative leakage function using the net leakage [`abs(mean)`] and the sum of absolute values (`L1 norm`). Panels (c) and (d) show the comparison of different precision modes with respect to a 2-times, over-resolved reference dataset for the mean velocity (left) and root mean square of the velocity vector, also validated against [242] (not shown).

exceeds 1 quadrillion. The authors further report near-ideal weak scaling ($\simeq$100%) in full-system runs on LLNL El Capitan (43K MI300A GPUs), OLCF Frontier (37.6K MI250X GPUs), and CSCS Alps (9.2K GH200 GPUs). The reported strong-scaling performance on full machines runs is also excellent: The code achieves efficiencies of 44% on both El Capitan and Frontier, and 80% on Alps.

### C. Specific bottlenecks for multiphase flows

Beyond the porting and optimization issues that affect standard GPU-based NS solvers, multiphase flow solvers pose additional challenges, for which well-established solutions are not available yet. In the following, these challenges are detailed with specific reference to dispersed flows laden with particles, drops or bubbles, which represent the most common application nowadays. First, we consider the challenges associated with Lagrangian methods (LPT, FT, and IBM) and then those connected to Eulerian methods (VOF, LS, and PFM). For brevity, challenges associated with LBM and SPH approaches are not discussed and we refer the reader to the review papers dealing with this specific topic [244–246].

#### *1. Lagrangian methods*

The main challenges associated with the porting Lagrangian methods (LPT, FT, and IBM) to GPUs involve: (i) handling the communications between the Lagrangian markers and the Eulerian fields, which are required to interpolate the fluid variables at the particle position (e.g., the fluid velocity for computing the hydrodynamic drag force) or to apply particle-to-fluid feedback forces (e.g., using two-way coupling or to enforce boundary conditions at the body surface); (ii)

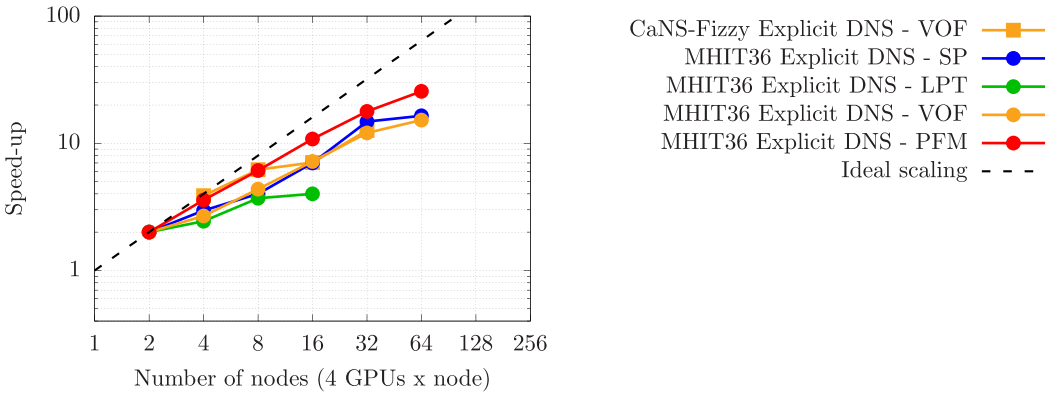


FIG. 18. Strong scaling results obtained for the solution of multiphase flow from CaNS-Fizzy (DNS of multiphase homogeneous isotropic turbulence at $\mathrm{Re}_\lambda = 95$). Different multiphase approaches have been employed: Lagrangian particle tracking (LPT, green), volume-of-fluid (VOF, orange), and phase-field method (PFM, red). Single-phase (SiPh) results are also reported as reference. The size of the computational domain is $1024^3$ grid points and the number of tracked particles is $10^6$. All tests were performed on the Leonardo supercomputer ($4 \times$ Nvidia A100 GPU per node) [225], starting from two nodes to remove the influence of NVLink on results.

maintaining load balancing, which ultimately means ensuring that GPU devices have roughly the same number of Lagrangian equations of motion to solve; (iii) efficient exchange of Lagrangian markers allocated to each GPU device; and (iv) efficient algorithms for handling particle-particle interactions across GPU threads (e.g., elastic collisions between solid particles, coalescence of drops or bubbles). Several of these challenges mirror those that already exist for efficient calculation in MPI frameworks [247,248]. Furthermore, several of these are linked to the choice of where to house the Lagrangian points, i.e., do the particles reside entirely on GPU devices or, rather, are only portions of the Lagrangian solver being accelerated?

A straightforward approach for Lagrangian methods is to assign ownership according to the Eulerian domain decomposition. That is, Lagrangian points are owned by the host node that also owns the portion of the domain in which they reside [211]. An advantage of this setup is that only a single region of the Eulerian grid must be stored in the GPU memory, and this region is exactly the one already stored on that node. If ghost cells are included, then both fluid-to-particle interpolation and particle-to-fluid force calculations can be done entirely on the host node, enabling Lagrangian elements to reside entirely on the GPU device. In this framework, MPI or GPU-to-GPU communication would be required to handle particles moving across the domain. A resulting disadvantage is that the total number of Lagrangian elements on each node is not known *a priori*, and there is a potential for load imbalance. This problem can be mitigated using load balancing techniques [248,249]. Additionally, the list of Lagrangian elements on each node will constantly change, requiring nontrivial bookkeeping strategies on the GPU device [211]. An alternative option that ensures load balancing is to assign to each task a number of Lagrangian points (or a set of connected markers) defined *a priori*. This parallelization, however, requires many internode communications because the task that is solving the equation of motion for the $i$th particle may not have direct access to the portion of the flow domain occupied by the particle, due to the domain decomposition applied to the Eulerian fields. From a GPU-computing perspective, these communications hinder code scalability as they are characterized by irregular patterns and may require information from just a few selected tasks. A striking example is provided by wall-bounded flows, which are characterized by particle accumulation in the near-wall region: In this case, tasks mostly need information on the flow field from this subregion of the domain [250]. The performance of this approach can be evaluated by looking at the strong scaling results reported in Fig. 18, which compare the scaling

performance obtained from the single-phase case (SiPh, blue) and those obtained when LPT is enabled (green) for the MHIT36 code. We observe that, when domain partitioning is moderate (from 8 to 32 GPUs, corresponding to 8 to 32 pencils), the scaling is relatively good and does not deviate much from the solution of the NS equations alone (SiPh results). However, as the number of nodes and GPUs is further increased, the communication overhead of the LPT becomes more and more predominant, thus preventing the code to scaleup. This can be also traced back to the fact that, as the domain is divided into a larger number of pencils, the likelihood that a particle migrates from one pencil to another also increases.

Another approach is to use the MPI-shared memory feature and assign different tasks to the nodes. For example, while $n$ nodes solve for the Eulerian fields, an additional node can be used to solve the Lagrangian particles [52,251]. This is the approach used by Sweet *et al.* [252], who employ a single MPI node with four GPU devices that contain all the Lagrangian particles tracked in the simulation. The clear disadvantage of this approach is that the GPU devices require the entire computational grid (for one-way coupled simulations), and the node taking care of the tracking would need to communicate with all others for a second time if the particle forces must be distributed within the portion of the domain occupied by the particle. This technique would be ideal for small domains with large numbers of Lagrangian elements (or expensive calculations for each Lagrangian point).

Finally, GPU-accelerated DNS of finite-size particles, bubbles, or droplets using direct forcing immersed boundary methods [253], or front-tracking approaches [254], present specific challenges related to the efficient handling of interpolation and spreading operations via a regularized Dirac delta kernel with finite support over a few grid cells (see, e.g., [255]). This step often dominates the computational cost of the particle treatment in IBM-based codes, and its parallelization is nontrivial. One may consider either a Eulerian- or Lagrangian-based approach for the associated communication operations; see the study by de Motta *et al*. [256] for more details. As discussed in this study, the Eulerian-based approach ensures the best performance at scale, at least for nondilute dispersed flows. This Eulerian strategy covers the stencil support of the IBM kernel using ghost layers (e.g., two cells in each direction for three-point kernel of Roma [255]). This enables both interpolation and spreading operations to be performed entirely within the computational subdomain that owns the corresponding Lagrangian grid point. Subsequently, the partial sums in the ghost layers, which must be communicated to neighboring tasks, can be sent via standard ghost-cell halo exchanges. While the total communication volume does not decrease with the number of particles, the key advantage of the Eulerian strategy lies in its predictability and scalability: The communication pattern is known *a priori* and scales favorably as the domain is further decomposed, with a communication overhead equivalent to a halo exchange per interpolated or spread scalar field. Moreover, an additional optimization can be achieved by restricting communication only to those Eulerian grid cells that intersect with the interpolation kernel support. This minimizes the amount of data exchanged, approaching a *no-op* for a vanishing number of Lagrangian markers in the domain. Since the workload associated with these operations is predictable (i.e., not larger than the size of the ghost cell) and local, the strategy is also ideal for the GPU-porting of FT- or IBM-based multiphase codes.

All in all, porting Lagrangian methods to GPU-accelerated architectures is an open challenge. It is not trivial to identify the optimal parallelization strategy and obtain a good compromise between load balancing and communication overhead, especially as far as GPU-to-GPU communication is concerned. The optimal strategy also depends on the nature of the Lagrangian problem: One-way coupled Lagrangian particles, for example, have relatively low arithmetic intensity, whereas physical processes such as collisions and thermodynamic evolution may significantly change the situation if denser suspensions are considered. Furthermore, the typical Lagrangian calculations often result in memory access patterns that are noncoalesced and not known *a priori*. Therefore, there is currently much room for achieving effective Lagrangian memory access and communication.

### *2. Eulerian methods*

In Eulerian methods like VOF, LS, and PFM, all fields (e.g., flow velocity, concentration, and/or color-marker function) are defined on the same grid. Therefore, the same parallelization strategy can be applied. This largely simplifies the coupling between the solution of the flow field and the solution of the color-marker function as well as the coupling terms, e.g., surface tension forces, density, and viscosity maps or the transport of the color-marker function by the velocity field. For instance, using staggered grids (in which velocity is defined at the cell faces and scalars are defined at the cell center), the color-marker function is usually defined at the cell center like pressure (or any other scalar). Despite this advantage, porting Eulerian methods is not always straightforward as some methods require complex geometrical reconstructions of the interface, nonlinear advection schemes, reinitialization algorithms or implicit treatment of the diffusive part. Each of these operations can introduce significant bottlenecks and inefficiencies using GPUs architectures.

Geometric VoF methods rely on geometrical reconstruction schemes to maintain the interface as sharp as possible. Their porting to GPUs is challenging due to the complex algorithms used to perform such reconstructions. Additionally, porting can become complicated in case of irregular memory access (from stencil operations and adaptive time stepping) and thread divergence (e.g., presence of frequent if-else conditions) caused by the adoption of nonlinear advection schemes. The offloading of these operations to GPUs does not always provide a speedup because the memory access pattern becomes irregular. In this regard, a notable example of efficient porting to GPUs of a geometrical VOF method is reported by Karnakov *et al.* [257]. In this work, the CubismAMR library [258] (used in the CUBISM-MPCF code [259], winner of the ACM Gordon Bell Prize 2013) is employed to abstract the parallelization as well as to offload the computations to GPUs of a VOF-based method based on the piecewise linear reconstruction algorithm [260]. Likewise, a GPU implementation of a 2D VOF solver is reported by Reddy *et al.* [261]. In contrast to geometric VOF methods, algebraic VOF methods adopt a different numerical modeling of the color function, avoiding explicit reconstruction of the interface in favor of approximating the color function in a computational cell. This approximation is done using algebraic functions that could involve polynomial approximations or hyperbolic tangent profiles [17]. Algebraic VOF methods are better suited to GPU architectures as they rely less on complex interface reconstruction techniques. See for instance FluTas [53], where an algebraic MTHINC VOF method [262] is implemented and offloaded to GPUs.

In addition to VOF methods, interesting applications to multiphase flows have involved the use of the LS method, which was among the first to be ported to GPUs. The earliest attempt has been reported in the work by Griebel *et al.* [263], where only the reinitialization step—required to restore the signed distance [264] of the level-set function—was offloaded to GPUs. This choice was also dictated by the memory constraints imposed by GPUs, which were much more stringent back in 2010. In subsequent works [265], the entire LS method had been offloaded to GPUs. However, single precision was used. Zaspel *et al.* [157] were the first to implement the LS method using double precision. Recently, Fu *et al.* [266] have developed a solver that combines the LS method with LBM and is accelerated using CUDA. It is worth mentioning that, since the LS method is used in a wide range of applications [267], many GPU-enabled implementations of this method are available [268–270]. A common bottleneck of these implementations, namely of the porting of LS methods to GPU architectures, is linked with the reinitialization step, which is usually performed using fast marching [271] or fast sweeping [272] methods. In fact, the use of fast marching algorithms involves the explicit identification of the interface position; once known, the front propagates outward in an ordered manner using a priority queue. These operations introduce dependencies and branch divergence, preventing full GPU parallelism. Additionally, grid points are updated in an unordered way, leading to noncoalesced memory accesses. Fast sweeping methods, on the other hand, also require the explicit identification of the interface position as a first step; the solution is then updated through a series of directional sweeps across the grid, which can be better parallelized [273,274]. These sweeps imply a structured memory access, making the memory access pattern

more predictable and thus more cache efficient. However, sweeps must be performed several times and along a specific direction: This introduces a non-negligible synchronization overhead, which can reduce the code performance when running on GPUs.

Finally, applications to multiphase flows also involve the use of the PFM, which usually rely on the Cahn-Hilliard equation or Allen-Cahn equation [275]. The main advantage of these methods from a GPU perspective is that they are interface blind and, hence, require no interface reconstruction or reinitialization steps. Efficient GPU implementations of PFM can be obtained, especially with explicit time-stepping and stencil-based computations on structured grids. This is the case, for instance, of PFMs based on the conservative Allen-Cahn equation [276,277], which can be efficiently ported to GPU architectures [55,149], since they are fully explicit and rely on standard PDE solvers. Methods based on the Cahn-Hilliard equation, which involve fourth-order derivatives, often face challenges because they typically require the use of semi-implicit schemes in which the linear part is handled implicitly using FFT-based approaches or linear solvers (e.g., multigrid or conjugate gradient methods). The GPU implementation of semi-implicit methods poses challenges similar to those encountered in solving the Poisson pressure equation. In this context, the use of mixed-precision arithmetic may help reduce communication overhead. However, while this represents an effective way to improve GPU performance, a careful evaluation of the trade-off between accuracy and stability is required. Overall, carefully implemented PFMs can achieve high efficiency on GPUs because they do not rely on interface-based algorithms.

The GPU porting efficiency of different multiphase methods based on Eulerian approaches can be appreciated looking again at Fig. 18. Specifically, we can analyze the strong scaling results obtained by the GPU implementation of VOF methods and phase-field methods, implemented in the codes CaNS-Fizzy and MHIT36. We can observe that for VOF methods, the scaling is comparable to the single-phase case. In spite of the facty that the VOF implementation does not involve specific bottleneck (e.g., all-to-all communication), the presence of many `if-else` conditions, due to the flux or interface reconstruction steps, hamper the efficiency of the GPU implementation. Indeed, these operations can lead to thread divergence (all threads will execute both set of instructions) and additional synchronization during the kernel execution is required. As mentioned above, this is due to the SIMT execution model massively exploited by GPUs. Moving to the PFM results, the resulting scaling shows a better behavior compared to both VOF and single-phase. The good scaling of PFM (ACDI formulation [277]) can be traced back to the fully explicit treatment and to the absence of interface reconstruction and/or all-to-all communication. Notably, results may vary depending on the specific formulation employed for both methods.

## V. CONCLUSIONS

We conclude our review by addressing the most important open questions and challenges posed by GPU-based high-performance computing in the broad area of fluid mechanics, as well as the future perspectives in this field. See also Fig. 19 for a concise summary.

### A. Open questions and challenges

The development and optimization of scientific codes for GPU architectures pose a number of challenges. One of the primary issues is performance portability and the choice of a programming paradigm. Unlike CPU clusters, which predominantly use standardized approaches such as MPI, GPU development lacks a universal standard. Researchers must decide whether to develop for Nvidia, AMD, Intel, or all of them and select between OpenACC, OpenMP, or other available frameworks. In this context, code development will benefit from the availability of performance-portable parallel programming frameworks (e.g., Kokkos [39], Legion [169], and OCCA [122]), which provide a production level solution consisting of multiple libraries that address the primary concerns for developing and maintaining applications in a portable way and for writing applications in a hardware agnostic way. Specifically, a key challenge will be the identification of a unified

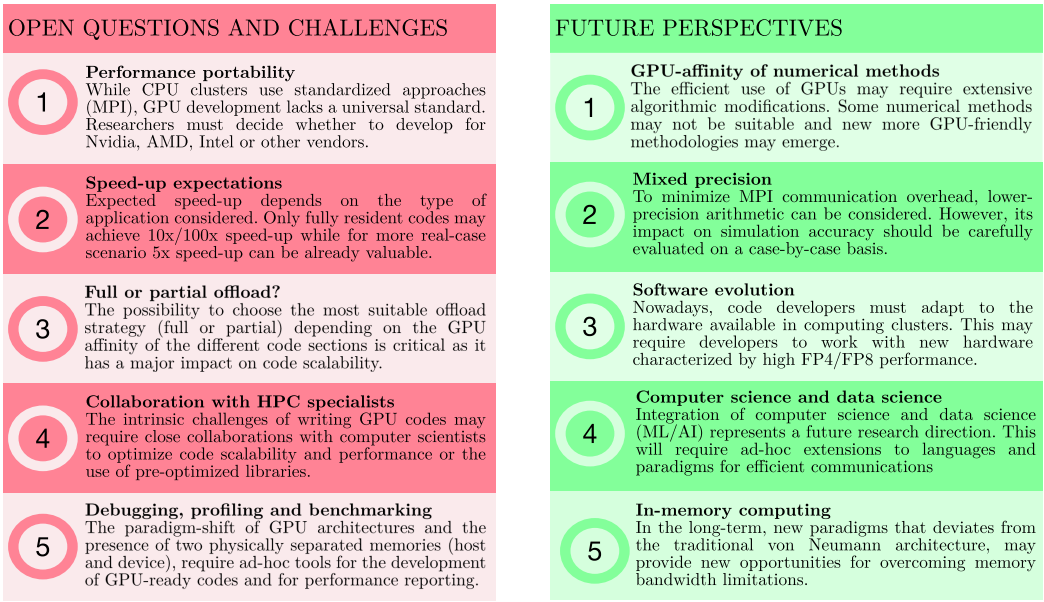


FIG. 19. Open questions and challenges (left) and future perspectives (right) at a glance.

approach that enables portability across GPUs from different vendors (AMD, NVIDIA, Intel) with minimal code modifications, facilitating the access to diverse HPC resources all over the globe. Future developments in this direction are, however, required to extend the number of languages and architectures that are supported.

Another key consideration is whether we can meet the high expectations for GPU-accelerated codes, where speedups of $100\times$ or more are often anticipated. While such gains may be feasible for fully GPU resident codes, real applications present a more complex challenge. Even achieving a moderate speedup, $5\times$ for example, without extensive algorithmic modifications can be highly valuable. In this context, experiences from ECP (see [278]) have shown that extending large, production-grade scientific codes to make them run efficiently on GPUs often requires a substantial effort. This includes algorithmic restructuring, memory layout redesign, and performance tuning across multiple architectures to achieve optimal performance on exascale HPC systems.

In this respect, GPU utilization strategies may play an important role. These strategies generally fall into two categories. In a full offload scenario, the entire computation is performed on the GPU, leaving only Input-Output tasks to the CPU. In a partial offload approach, a substantial portion of the computation is assigned to the GPU, but some processing remains on the CPU. In the latter case, data movement between CPU and GPU often becomes the primary bottleneck. The possibility of choosing the most suitable approach (full or partial offload) is becoming increasingly important because it has a major impact on code scalability. Data reduction is also an aspect of crucial importance, since *in situ* and runtime coprocessing of DNS data is becoming increasingly essential for real-time data analysis and visualization.

From a practical point of view, an open question is whether software engineers are required to ensure an efficient porting of scientific codes, which is in turn related to the availability of financial resources to support this kind of consultancy. If GPU development proves to be too time-consuming, then the number of independently developed codes may shrink in the future, leaving only a handful of widely used solutions. Alternatively, future generations of Ph.D. students may become as proficient in GPU programming as they are today in MPI programming. This latter scenario may be supported by the availability of specialized libraries for inter-GPU communications

(e.g., cuDecomp [233]) and paradigm-based approaches. Clearly, it remains to be seen whether such approaches can provide performance comparable to that of more sophisticated ones.

One additional open challenge that must be mentioned is related to the proper debugging, profiling, and benchmarking of GPU-accelerated codes. In terms of debugging, the presence of two physically separate memory spaces (host and device) exposes developers to synchronization issues, especially when memory management is handled manually, i.e., without using managed memory or unified memory features. Comparing the performance between CPU and GPU implementations is further complicated by differences in the MPI task distribution. While using CPUs, the number of MPI tasks per node corresponds to the number of cores or threads available in the computing node. Using GPUs, however, this corresponds to the number of GPUs available in the node. Establishing standardized methods for performance reporting is essential to ensure fair and meaningful comparisons across architectures and to accurately evaluate code performance.

### B. Future perspectives

To conclude our discussion, we discuss next the most significant future perspectives of GPU-based high-performance computing in the broad area of fluid mechanics. From the discussion made in the previous sections, it seems quite obvious that the development of the next-generation GPU-based DNS and LES codes will have to cope with the rapidly evolving hardware and the capability to exploit new architectures. On the one hand, the traditional paradigm of a single researcher working independently may be no longer viable, and close collaboration with HPC specialists will likely be required. On the other hand, new architectures such as the Grace Hopper superchip (featuring a CPU and GPU coherent memory model) will allow performing calculations on both GPU and CPU, rather than so-called GPU-resident calculations, thus paving the way for new computational approaches.

The recent advances in hardware performance and architecture may require a rethinking of the way in which physical problems are tackled. In the past, it was possible to obtain a scalable and efficient parallel implementation for almost all commonly employed numerical methods: As a consequence, scientific research was problem driven. Nowadays, as next-generation GPUs are becoming denser and denser in terms of computational power [14], it is increasingly difficult to obtain good strong scaling performance as the overhead introduced by MPI communications becomes non-negligible [47]. This trend may limit the choice of the numerical method to be used—preferring those characterized by the least amount of MPI communications—and some problems may turn out to be more difficult to tackle in the future. This trend may as well lead to the development of new GPU-tailored methodologies. Furthermore, the energy efficiency of the different numerical methodologies could represent an additional constraint, potentially becoming a key factor in next-generation computing infrastructures [8,238].

A first way to reduce the overhead due to MPI communications is the use of lower-precision arithmetic (FP32 and FP16), which can yield significant gains in power efficiency and computational performance [182,279]. However, following this direction necessitates a thorough assessment of its impact on simulation accuracy [12,20,182,280,281]. For DNS of turbulent single-phase flows, preliminary results indicate that performing computations in mixed precision or even full single precision can be a viable strategy. To enhance the computational efficiency of currently available methods, adaptive mesh refinement and data-compression techniques can be leveraged, thus enabling the simulation of more complex geometries. Nevertheless, the use of libraries introduces dependencies (thus virtually reducing portability) and these libraries must be GPUoptimized to avoid significant bottlenecks. Likewise, the use of preconditioners and solvers, such as Hypre for CPU-based computations, must be adapted to GPU architectures.

In the past, hardware was often designed or optimized to meet the specific needs of software. However, this strategy has become less common nowadays, since general-purpose hardware—such as GPUs and other accelerators—currently dominates the market. Modern developers must adapt their code and algorithms to run efficiently on widely available computing clusters. This often

requires optimizing existing algorithms to take advantage of hardware designed primarily for AI workloads, which deliver high performance when using low-precision floating-point formats like FP4 and FP8, rather than the more traditional FP32 and FP64 formats. Future developments may exploit the large computational power available in the computing units dedicated to FP4 and FP8 formats (i.e., Tensor Cores and Matrix Cores for Nvidia and AMD architectures respectively) instead of the native FP32 and FP64 computing units. In particular, a possible scenario is the emulation of FP32 and FP64 formats on FP4 and FP8 computing units rather than using FP32 and FP64 native computing units [282–284].

An intriguing research direction is the integration of computer science and data science [285,286], which can pave the way for new hybrid approaches. From a programming and implementation point of view, this solution opens the door to new challenges as well, because the languages and paradigms commonly employed in computer and data science are different. It would be desirable to extend current performance-portable parallel programming frameworks (e.g., Kokkos [39], Legion [169], and OCCA [122]) to additional languages as well as to develop and optimize interfaces through which these two research fields can efficiently communicate [287].

Finally, considering a longer-term scenario, new paradigms that deviate from the traditional von Neumann architecture could provide new opportunities for overcoming memory bandwidth limitations and for improving computational throughput. In particular, the use of in-memory computing technologies could be used to overcome the von Neumann bottleneck (e.g., memory wall), thus providing sustainable improvements in computing throughput and energy efficiency [279,288,289]. As of now, however, these technologies are characterized by a low technology readiness level and are still far from being widely adopted in high-performance computing systems.

## ACKNOWLEDGMENTS

This article is the result of a joint effort by the authors and originated from the discussions that took place during the KAUST Turbulence Workshop, titled "Outstanding Challenges in Wall Turbulence and Lessons to Be Learned from Pipes," organized in 2024 by Peter Schmid (KAUST), Sigurdur Thoroddsen (KAUST), Hassan Nagib (Illinois Tech.), K. R. Sreenivasan (New York University), and Ivan Marusic (University of Melbourne). The authors thank the organizers for making the development of this article possible. The authors also thank Josh Romero and Massimiliano Fatica for valuable discussions on single- and mixed-precision computations.

## DATA AVAILABILITY

No data were created or analyzed in this study. A repository with links to all software listed in the tables of this paper is available at [290].

[1] W. Gao, W. Zhang, W. Cheng, and R. Samtaney, Wall-modelled large-eddy simulation of turbulent flow past airfoils, J. Fluid Mech. **873**, 174 (2019).

[2] W. C. Reynolds, The potential and limitations of direct and large eddy simulations, in *Whither Turbulence? Turbulence at the Crossroads*, edited by J. L. Lumley, Lecture Notes in Physics Vol. 357 (Springer, Berlin, Heidelberg, 1990), p. 313.

[3] S. Pirozzoli and P. Orlandi, Natural grid stretching for DNS of wall-bounded flows, J. Comput. Phys. **439**, 110408 (2021).

[4] TOP500 list, https://top500.org/lists/top500/.

[5] S. Matsuoka, Fugaku and A64FX: The first exascale supercomputer and its innovative ARM CPU, in *Proceedings of the 2021 Symposium on VLSI Circuits* (IEEE, Los Alamitos, CA, 2021), pp. 1–3.

[6] J. S. Vetter, E. P. DeBenedictis, and T. M. Conte, Architectures for the post-moore era, IEEE Micro. **37**, 6 (2017).
[7] P. M. Kogge and W. J. Dally, Frontier vs the exascale report: Why so long? and are we really there yet? in *Proceedings of the IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS'22)* (IEEE, Los Alamitos, CA, 2022), pp. 26–35.
[8] W. Shin, J. B. White, W. Elwasif, R. F. Da Silva, C. Zimmer, B. Messer, R. Budiardja, A. Georgiadou, V. M. Vergara, J. Lange, *et al.*, Towards sustainable post-exascale leadership computing, in *Proceedings of the Supercomputing Conference (SC24)* (IEEE, Los Alamitos, CA, 2024), pp. 1790–1794.
[9] W. Shin, V. Oles, A. M. Karimi, J. A. Ellis, and F. Wang, Revealing power, energy and thermal dynamics of a 200PF pre-exascale supercomputer, in *Proceedings of the Supercomputing Conference (SC'21)* (IEEE, Los Alamitos, CA, 2021), pp. 1–14.
[10] X. Yang, W. Zhang, M. Abkar, and W. Anderson, Computational fluid dynamics: Its carbon footprint and role in carbon reduction, J. Renew. Sustain. Energy **16**, 055906 (2024).
[11] L. Bertaccini, G. Paulin, T. Fischer, S. Mach, and L. Benini, MiniFloat-NN and ExSdotp: An ISA extension and a modular open hardware unit for low-precision training on RISC-V cores, in *Proceedings of the IEEE 29th Symposium on Computer Arithmetic (ARITH)* (IEEE, Los Alamitos, CA, 2022), pp. 1–8.
[12] F. Brogi, S. Bnà, G. Boga, G. Amati, T. E. Ongaro, and M. Cerminara, On floating point precision in computational fluid dynamics using OpenFOAM, Future Gener. Comput. Syst. **152**, 1 (2024).
[13] A. Bhatnagar, V. Pandey, P. Perlekar, and D. Mitra, Rate of formation of caustics in heavy particles advected by turbulence, Philos. Trans. R. Soc. A **380**, 20210086 (2022).
[14] Nvidia blackwell specs, https://resources.nvidia.com/en-us-blackwell-architecture.
[15] C. Marchioli, M. Bourgoin, F. Coletti, R. Fox, J. Magnaudet, M. Reeks, O. Simonin, M. Sommerfeld, F. Toschi, L.-P. Wang, and S. Balachandar, Particle-laden flows, Int. J. Multiphase Flow **191**, 105291 (2025).
[16] O. M. H. Rodriguez, P. Angeli, D. Legendre, E. Climent, and A. Soldati, Drop-laden flows, Int. J. Multiphase Flow **191**, 105284, (2025).
[17] M. Garcia-Villalba, T. Colonius, O. Desjardins, D. Lucas, A. Mani, D. Marchisio, O. K. Matar, F. Picano, and S. Zaleski, Numerical methods for multiphase flows, Int. J. Multiphase Flow **191**, 105285 (2025).
[18] K. Rupp, Microprocessor trend data, https://github.com/karlrupp/microprocessor-trend-data.
[19] M. Louboutin, M. Lange, F. J. Herrmann, N. Kukreja, and G. Gorman, Performance prediction of finite-difference solvers for different computer architectures, Comput. Geosci. **105**, 148 (2017).
[20] Y. Chen, P. de Oliveira Castro, P. Bientinesi, N. Jansson, and R. Iakymchuk, Enabling mixed-precision in spectral element codes, Future Gener. Comp. Sy. **174**, 107990 (2025).
[21] W. A. Wulf and S. A. McKee, Hitting the memory wall: Implications of the obvious, ACM SIGARCH Comput. Archit. News **23**, 20 (1995).
[22] S. A. McKee, Reflections on the memory wall, in *Proceedings of the 1st Conference on Computing Frontiers* (ACM Press, New York, 2004), p. 162.
[23] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, AI and memory wall, IEEE Micro. **44**, 33 (2024).
[24] S. Williams, A. Waterman, and D. Patterson, Roofline: An insightful visual performance model for multicore architectures, Commun. ACM **52**, 65 (2009).
[25] G. Ofenbeck, R. Steinmann, V. Caparros, D. G Spampinato, and M. Püschel, Applying the roofline model, in *Proceedings of the IEEE ISPASS 2014* (IEEE, Los Alamitos, CA, 2014), pp. 76–85.
[26] J. Jeffers and J. Reinders, *High Performance Parallelism Pearls Volume Two: Multicore and Many-core Programming Approaches* (Morgan Kaufmann, Burlington, MA, 2015).
[27] B. Mostafazadeh, F. Marti, F. Liu, and A. Chandramowlishwaran, Roofline guided design and analysis of a multi-stencil CFD solver for multicore performance, in *Proceedings of the 2018 IEEE IPDPS* (IEEE, Los Alamitos, CA, 2018), pp. 753–762.

[28] D. M. Tullsen, S. J. Eggers, and H. M. Levy, Simultaneous multithreading: Maximizing on-chip parallelism, in *Proceedings of the 22nd Annual International Symposium on Computer Architecture* (IEEE, Los Alamitos, CA, 1995), pp. 392–403.
[29] S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, R. L. Stamm, and D. M. Tullsen, Simultaneous multithreading: A platform for next-generation processors, IEEE Micro. **17**, 12 (1997).
[30] M. Bernaschi, M. Fatica, S. Melchionna, S. Succi, and E. Kaxiras, A flexible high-performance lattice Boltzmann GPU code for the simulations of fluid flows in complex geometries, Concurr. Comput. Pract. Exper. **22**, 1 (2010).
[31] K. E. Niemeyer and C.-J. Sung, Recent progress and challenges in exploiting graphics processors in computational fluid dynamics, J. Supercomput. **67**, 528 (2014).
[32] The die is the physical piece of silicon that contains the GPU's circuits, cores, caches, memory controllers, and other components. A die shot shows the layout of all these components as seen under a microscope or using specialized imaging equipment.
[33] NVIDIA, P. Vingelmann, and F. H. P. Fitzek, CUDA (2020), https://developer.nvidia.com/cuda/toolkit.
[34] M. Fatica and G. Ruetsch, *CUDA Fortran for Scientists and Engineers, Best Practices for Efficient CUDA Fortran Programming* (Morgan Kaufmann, Waltham, MA, 2014).
[35] AMD. HIP documentation, https://rocm.docs.amd.com/projects/HIP/en/latest/.
[36] J. E. Stone, D. Gohara, and G. Shi, OpenCL: A parallel programming standard for heterogeneous computing systems, J. Parallel Distrib. Comput. **12**, 66, (2010).
[37] L. Dagum and R. Menon, OpenMP: an industry standard API for shared-memory programming, Comput. Sci. Eng. **5**, 46 (1998).
[38] OpenACC working group *et al.*, The OpenACC application programming interface (2011), https://www.openacc.org.
[39] H. C. Edwards, C. R. Trott, and J. Sunderland, Kokkos: Enabling manycore performance portability through polymorphic memory access patterns, J. Parallel Distrib. Comput. **74**, 3202 (2014).
[40] Intel, The oneapi implementation of sycl, https://www.intel.com/content/www/us/en/developer/tools/oneapi/data-parallel-c-plus-plus.html.
[41] Khronos, C++ programming for heterogeneous parallel computing, https://www.khronos.org/sycl/.
[42] GPU vendor/programming model compatibility overview, https://github.com/AndiH/gpu-lang-compat.
[43] A. J. Chorin, Numerical solution of the Navier-Stokes equations, Math. Comp. **22**, 745 (1968).
[44] J. Kim and P. Moin, Application of a fractional-step method to incompressible Navier-Stokes equations, J. Comput. Phys. **59**, 308 (1985).
[45] P. J. Blair, An analysis of the fractional step method, J. Comput. Phys. **108**, 51 (1993).
[46] X. Zhu, E. Phillips, V. Spandan, J. Donners, G. Ruetsch, J. Romero, R. Ostilla-Mónico, Y. Yang, D. Lohse, and R. Verzicco, AFiD-GPU: A versatile Navier–Stokes solver for wall-bounded turbulent flows on GPU clusters, Comput. Phys. Commun. **229**, 199 (2018).
[47] P. Costa, A FFT-based finite-difference solver for massively-parallel direct numerical simulations of turbulent flows, Comput. Math. Appl. **76**, 1853 (2018).
[48] P. Costa, E. Phillips, L. Brandt, and M. Fatica, GPU acceleration of CaNS for massively-parallel direct numerical simulations of canonical fluid flows, Comput. Math. Appl. **81**, 502 (2021).
[49] D. Mierke, C. F. Janßen, and T. Rung, An efficient algorithm for the calculation of sub-grid distances for higher-order LBM boundary conditions in a GPU simulation environment, Comput. Math. Appl. **79**, 66 (2020).
[50] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo, FLASH: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes, Astrophys. J. Suppl. Ser. **131**, 273 (2000).
[51] A. Dubey, K. Weide, J. O Neal, A. Dhruv, S. Couch, J. A. Harris, T. Klosterman, R. Jain, J. Rudi, B. Messer, *et al.*, Flash-X: A multiphysics simulation software instrument, SoftwareX **19**, 101168 (2022).
[52] A. Roccon, G. Soligo, and A. Soldati, FLOW36: A spectral solver for phase-field based multiphase turbulence simulations on heterogeneous computing architectures, Comput. Phys. Commun. **313**, 109640 (2025).

[53] M. Crialesi-Esposito, N. Scapin, A. D. Demou, M. E. Rosti, P. Costa, F. Spiga, and L. Brandt, FluTAS: A GPU-accelerated finite difference code for multiphase flows, Comput. Phys. Commun. **284**, 108602 (2023).
[54] G. Bilotta, From GPU to CPU (and beyond): Extending hardware support in GPUSPH through a SYCL-inspired interface, Concur. Comp. Pract. E **37**, e8313 (2025).
[55] A. Roccon, L. Enzenberger, D. Zaza, and A. Soldati, MHIT36: A phase-field code for GPU simulations of multiphase homogeneous isotropic turbulence, Comput. Phys. Commun. **316**, 109804, (2025).
[56] A. Roccon, MHIT36: Extension to wall-bounded turbulence and scalar transport equation, Comput. Phys. Commun. **320**, 109956 (2026).
[57] N. Jansson, M. Karp, A. Podobas, S. Markidis, and P. Schlatter, Neko: A modern, portable, and scalable framework for high-fidelity computational fluid dynamics, Comput. Fluids **275**, 106243 (2024).
[58] N. Jansson, M. Karp, J. Wahlgren, S. Markidis, and P. Schlatter, Design of Neko: A scalable high-fidelity simulation framework with extensive accelerator support, Concurr. Comput. Pract. Exp. **37**, e8340 (2025).
[59] P. Fischer, S. Kerkemeier, M. Min, Y.-H. Lan, M. Phillips, T. Rathnayake, E. Merzari, A. Tomboulides, A. Karakus, and N. Chalmers, NekRS, a GPU-accelerated spectral element Navier–Stokes solver, Parallel Comput. **114**, 102982 (2022).
[60] J. M. López, D. Feldmann, M. Rampp, A. Vela-Martín, L. Shi, and M. Avila, nsCouette–a high-performance code for direct numerical simulations of turbulent Taylor–Couette flow, SoftwareX **11**, 100395 (2020).
[61] A. Ramadhan, G. Wagner, C. Hill, J.-M. Campin, V. Churavy, T. Besard, A. Souza, A. Edelman, R. Ferrari, and J. Marshall, Oceananigans.jl: Fast and friendly geophysical fluid dynamics on GPUs, J. Open Source Softw. **5**, 2018 (2020).
[62] V. Heuveline and J. Latt, The OpenLB project: An open source and object oriented implementation of lattice Boltzmann methods, Int. J. Mod. Phys. C **18**, 627 (2007).
[63] J. Latt, O. Malaspinas, D. Kontaxakis, A. Parmigiani, D. Lagrava, F. Brogi, M. B. Belgacem, Y. Thorimbert, S. Leclaire, S. Li, *et al.*, Palabos: Parallel lattice Boltzmann solver, Comput. Math. Appl. **81**, 334 (2021).
[64] J. Latt and C. Coreixas, Multi-GPU acceleration of PALABOS fluid solver using C standard parallelism, Comput. Phys. Commun. **323**, 110094 (2026).
[65] L. Gasparino, F. Spiga, and O. Lehmkuhl, SOD2D: A GPU-enabled spectral finite elements method for compressible scale-resolving simulations, Comput. Phys. Commun. **297**, 109067 (2024).
[66] M. Mortensen and H. P. Langtangen, High performance Python for direct numerical simulations of turbulent flows, Comput. Phys. Commun. **203**, 53 (2016).
[67] T. D. Economon, F. Palacios, S. R. Copeland, T. Lukaczyk, and J. J. Alonso, SU2: An open-source suite for multiphysics simulation and design, AIAA J. **54**, 828 (2016).
[68] M. Bauer, S. Eibl, C. Godenschwager, N. Kohl, M. Kuron, C. Rettinger, F. Schornbaum, C. Schwarzmeier, D. Thönnes, H. Köstler, *et al.*, waLBerla: A block-structured high-performance framework for multiphysics simulations, Comput. Math. Appl. **81**, 478 (2021).
[69] WaterLily.jl: A differentiable fluid simulator in Julia with fast heterogeneous execution, Comput. Phys. Commun. **315**, 109748 (2025).
[70] P. Bartholomew, G. Deskos, R. A. S. Frantz, F. N. Schuch, E. Lamballais, and S. Laizet, Xcompact3D: An open-source framework for solving turbulence problems on a Cartesian mesh, SoftwareX **12**, 100550 (2020).
[71] P. M. Gresho and S. T. Chan, On the theory of semi-implicit projection methods for viscous incompressible flow and its implementation via a finite element method that also introduces a nearly consistent mass matrix. Part 2: Implementation, Int. J. Numer. Methods Fluids **11**, 621 (1990).
[72] H. Le and P. Moin, An improvement of fractional step methods for the incompressible Navier-Stokes equations, J. Comput. Phys. **92**, 369 (1991).
[73] S. Hahn, J. Je, and H. Choi, Direct numerical simulation of turbulent channel flow with permeable walls, J. Fluid Mech. **450**, 259 (2002).

[74] H. Abe, H. Kawamura, and Y. Matsuo, Direct numerical simulation of a fully developed turbulent channel flow with respect to the Reynolds number dependence, J. Fluids Eng. **123**, 382 (2001).
[75] S. V. Patankar and D. B. Spalding, A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows, in *Numerical Prediction of Flow, Heat Transfer, Turbulence and Combustion* (Elsevier, Amsterdam, 1983), pp. 54–73.
[76] J. Kim, P. Moin, and R. Moser, Turbulence statistics in fully developed channel flow at low Reynolds number, J. Fluid Mech. **177**, 133 (1987).
[77] P. R. Spalart, Direct simulation of a turbulent boundary layer up to $\mathrm{Re}_\theta = 1410$, J. Fluid Mech. **187**, 61 (1988).
[78] P. F. Fischer, L. W. Ho, G. E. Karniadakis, E. M. Conquest, and A. T. Patera, Recent advances in parallel spectral element simulation of unsteady incompressible flows, in *Computational Structural Mechanics & Fluid Dynamics* (Pergamon Press, Oxford, 1988), pp. 217–231.
[79] S. Mittal and V. Kumar, Finite element study of vortex-induced cross-flow and in-line oscillations of a circular cylinder at low Reynolds numbers, Int. J. Numer. Methods Fluids **31**, 1087 (1999).
[80] A. T. Patera, A spectral element method for fluid dynamics: Laminar flow in a channel expansion, J. Comput. Phys. **54**, 468 (1984).
[81] Y. Maday and A. T. Patera, Spectral element methods for the incompressible Navier-Stokes equations, in *State-of-the-art Surveys on Computational Mechanics (A90-47176 21-64)* (ASME, New York, 1989), pp. 71–143.
[82] R. G. Jacobs and P. A. Durbin, Simulations of bypass transition, J. Fluid Mech. **428**, 185 (2001).
[83] J. Kim, D. Kim, and H. Choi, An immersed-boundary finite-volume method for simulations of flow in complex geometries, J. Comput. Phys. **171**, 132 (2001).
[84] T. S. Lund, X. Wu, and K. D. Squires, Generation of turbulent inflow data for spatially-developing boundary layer simulations, J. Comput. Phys. **140**, 233 (1998).
[85] H. Sanghyun, P. Junshin, and Y. Donghyun, A GPU-accelerated semi-implicit fractional-step method for numerical solutions of incompressible Navier–Stokes equations, J. Comput. Phys. **352**, 246 (2018).
[86] G. Alfonsi, S. A. Ciliberti, M. Mancini, and L. Primavera, GPGPU implementation of mixed spectral-finite difference computational code for the numerical integration of the three-dimensional time-dependent incompressible Navier Stokes equations, Comput. Fluids **102**, 237 (2014).
[87] F. Salvadore, M. Bernardini, and M. Botti, GPU accelerated flow solver for direct numerical simulation of turbulent flows, J. Comput. Phys. **235**, 129 (2013).
[88] S. P. Vanka, A. F. Shinn, and K. C. Sahu, Computational fluid dynamics using graphics processing units: Challenges and opportunities, in *Proceedings of the ASME International Mechanical Engineering Congress and Exposition* (ASME, New York, 2011), Vol. 54921, p. 429.
[89] H. Zolfaghari, B. Becsek, M. G. Nestola, W. B. Sawyer, R. Krause, and D. Obrist, High-order accurate simulation of incompressible turbulent flows on many parallel GPUs of a hybrid-node supercomputer, Comput. Phys. Commun. **244**, 132 (2019).
[90] M. Di Renzo, L. Fu, and J. Urzay, HTR solver: An open-source exascale-oriented task-based multi-GPU high-order code for hypersonic aerothermodynamics, Comput. Phys. Commun. **255**, 107262 (2020).
[91] M. Bernardini, D. Modesti, F. Salvadore, and S. Pirozzoli, STREAmS: A high-fidelity accelerated solver for direct numerical simulation of compressible turbulent flows, Comput. Phys. Commun. **263**, 107906 (2021).
[92] J. Thibault and I. Senocak, CUDA implementation of a Navier-Stokes solver on multi-GPU desktop platforms for incompressible flows, in *Proceedings of the 47th AIAA Aerospace Sciences Meeting* (AIAA, Reston, VA, 2009), p. 758.
[93] D. Jacobsen, J. Thibault, and I. Senocak, An MPI-CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters, in *Proceedings of the 48th AIAA Aerospace Sciences Meeting* (AIAA, Reston, VA, 2010), p. 522.
[94] H. Cheng, M. Grossman, and T. McKercher, *Professional CUDA C Programming* (John Wiley & Sons, New York, 2014).
[95] D. B. Kirk and W. H. Wen-Mei, *Programming Massively Parallel Processors: A Hands-on Approach* (Morgan Kaufmann, Waltham, MA, 2016).

[96] G. Ruetsch and M. Fatica, *CUDA Fortran for Scientists and Engineers: Best Practices for Efficient CUDA Fortran Programming* (Elsevier, Amsterdam, 2024).
[97] S. Ha, J. Park, and D. You, A multi-GPU method for ADI-based fractional-step integration of incompressible Navier-Stokes equations, Comput. Phys. Commun. **265**, 107999 (2021).
[98] X. H. Sun, H. Z. Sun, and L. M. Ni, Parallel algorithms for solution of tridiagonal systems on multicomputers, in *Proceedings of the International Conference on Supercomputing* (ACM Press, New York, 1989), pp. 303–312.
[99] E. Laszlo, M. Giles, and J. Appleyard, Manycore algorithms for batch scalar and block tridiagonal solvers, ACM Trans. Math. Softw. **42**, 1 (2016).
[100] K. H. Kim, J. H. Kang, X. Pan, and J. I. Choi, PaScaL_TDMA: A library of parallel and scalable solvers for massive tridiagonal systems, Comput. Phys. Commun. **260**, 107722 (2021).
[101] R. D. Sanhueza, J. Peeters, and P. Costa, A pencil-distributed finite-difference solver for extreme-scale calculations of turbulent wall flows at high Reynolds number, Comput. Phys. Comm. **316**, 109811 (2025).
[102] L. N. Trefethen, *Spectral Methods in MATLAB* (SIAM, Philadelphia, PA, 2000).
[103] K. Ravikumar, D. Appelhans, and P. K. Yeung, GPU acceleration of extreme scale pseudo-spectral simulations of turbulence using asynchronism, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (ACM Press, New York, 2019), pp. 1–22.
[104] P. K. Yeung, K. Ravikumar, S. Nichols, and R. Uma-Vaideswaran, GPU-enabled extreme-scale turbulence simulations: Fourier pseudo-spectral algorithms at the exascale using OpenMP offloading, Comput. Phys. Commun. **306**, 109364 (2025).
[105] C. Cecka, A. J. Lew, and E. Darve, Assembly of finite element methods on graphics processors, Int. J. Numer. Methods Eng. **85**, 640 (2011).
[106] N. S.-C. Kao and T. W.-H. Sheu, Development of a finite element flow solver for solving three-dimensional incompressible Navier–Stokes solutions on multiple GPU cards, Comput. Fluids **167**, 285 (2018).
[107] M. U. Zapata, F. J. Hernandez-Lopez, and R. I. Balam, A parallel unstructured multi-color SOR solver for 3D Navier–Stokes equations on graphics processing units, Comput. Fluids **260**, 105909 (2023).
[108] R. Borrell, D. Dosimont, M. Garcia-Gasulla, G. Houzeaux, O. Lehmkuhl, V. Mehta, H. Owen, M. Vázquez, and G. Oyarzun, Heterogeneous CPU/GPU co-execution of CFD simulations on the POWER-9 architecture: Application to airplane aerodynamics, Future Gener. Comput. Syst. **107**, 31 (2020).
[109] H. Owen, D. Ernst, T. Gruber, O. Lemkuhl, G. Houzeaux, L. Gasparino, and G. Wellein, Alya towards exascale: Optimal OpenACC performance of the Navier-Stokes finite element assembly on GPUs, in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS'24)* (IEEE, Los Alamitos, CA, 2024), pp. 408–416.
[110] H. Owen, O. Lehmkuhl, P. D'Ambra, F. Durastante, and S. Filippone, Alya toward exascale: Algorithmic scalability using psctoolkit, J. Supercomput. **80**, 13533 (2024).
[111] C. D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, and D. Ekelschot, Nektar: An open-source spectral/hp element framework, Comput. Phys. Commun. **192**, 205 (2015).
[112] D. Moxey, C. D. Cantwell, Y. Bao, A. Cassinelli, G. Castiglioni, S. Chun, E. Juda, E. Kazemi, K. Lackhove, and J. Marcon, Nektar: Enhancing the capability and application of high-fidelity spectral/hp element methods, Comput. Phys. Commun. **249**, 107110 (2020).
[113] H. M. Blackburn, D. Lee, T. Albrecht, and J. Singh, Semtex: A spectral element–Fourier solver for the incompressible Navier–Stokes equations in cylindrical or cartesian coordinates, Comput. Phys. Commun. **245**, 106804 (2019).
[114] A. Abdelfattah, V. Barra, N. Beams, R. Bleile, J. Brown, J.-S. Camier, R. Carson, N. Chalmers, V. Dobrev, and Y. Dudouit, GPU algorithms for efficient exascale discretizations, Parallel Comput. **108**, 102841 (2021).
[115] S. Markidis, J. Gong, M. Schliephake, E. Laure, A. Hart, D. Henty, K. Heisey, and P. Fischer, OpenACC acceleration of the Nek5000 spectral element code, Int. J. High Perform. Comput. Appl. **29**, 311 (2015).

[116] M. Otten, J. Gong, A. Mametjanov, A. Vose, J. Levesque, P. Fischer, and M. Min, An MPI/OpenACC implementation of a high-order electromagnetics solver with GPUDirect communication, Int. J. High Perform. Comput. Appl. **30**, 320 (2016).

[117] J. Gong, S. Markidis, E. Laure, M. Otten, P. Fischer, and M. Min, Nekbone performance on GPUs with OpenACC and CUDA fortran implementations, J. Supercomput. **72**, 4160 (2016).

[118] E. Otero, J. Gong, M. Min, P. Fischer, P. Schlatter, and E. Laure, OpenACC acceleration for the PN–PN-2 algorithm in Nek5000, J. Parallel Distrib. Comput. **132**, 69 (2019).

[119] M. Karp, N. Jansson, A. Podobas, P. Schlatter, and S. Markidis, Optimization of tensor-product operations in nekbone on GPUs, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC20)* (ACM Press, New York, 2020).

[120] J. Vincent, J. Gong, M. Karp, A. Peplinski, N. Jansson, A. Podobas, A. Jocksch, J. Yao, F. Hussain, and S. Markidis, Strong scaling of OpenACC enabled Nek5000 on several GPU based HPC systems, in *Proceedings of the Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region Workshops* (ACM Press, New York, 2022), pp. 94–102.

[121] M. Karp, Direct numerical simulation of turbulence on heterogenous computer systems, Ph.D. thesis, KTH Royal Institute of Technology, 2024.

[122] D. S. Medina, A. St-Cyr, and T. Warburton, OCCA: A unified approach to multi-threading languages, arXiv:1403.0968.

[123] W. Lu, D. Aljubaili, T. Zahtila, L. Chan, and A. Ooi, Asymmetric wakes in flows past circular cylinders confined in channels, J. Fluid Mech. **958**, A8 (2023).

[124] T. Zahtila, L. Chan, A. Ooi, K. Liu, M. Benjamin, and G. Iaccarino, Influence of miura-origami shapes on drag in turbulent flows, in *Proceedings of the Summer Program 2022 for the Center for Turbulence Research* (Stanford University, Center for Turbulence Research, Stanford, CA, 2022).

[125] D. Reger, E. Merzari, T. Nguyen, Y.-H. Lan, P. Fischer, and Y. Hassan, A study of the transition to turbulence in a bed of 67 spherical pebbles, Flow Turb. Comb. **114**, 765 (2024).

[126] W. Lu, L. Chan, and A. Ooi, Spectral analysis of confined cylinder wakes, Fluids **10**, 84 (2025).

[127] P.-H. Tsai, P. Fischer, and T. Iliescu, A time-relaxation reduced order model for the turbulent channel flow, J. Comput. Phys. **521**, 113563 (2025).

[128] W. Lu, T. Zahtila, L. Chan, Q. D. Nguyen, C. Lei, G. Iaccarino, and A. Ooi, Modeling of uncertainties from spanwise asymmetries in upstream conditions and measurement plane location for flow past a circular cylinder confined within a duct, Phys. Rev. Fluids **10**, 064601 (2025).

[129] D. Massaro, M. Karp, N. Jansson, S. Markidis, and P. Schlatter, Direct numerical simulation of the turbulent flow around a flettner rotor, Sci. Rep. **14**, 3004 (2024).

[130] M. Gomez-Gesteira, A. J. C. Crespo, B. D. Rogers, R. A. Dalrymple, J. M. Dominguez, and A. Barreiro, SPHysics—Development of a free-surface fluid solver-Part 2: Efficiency and test cases, Comput. Geosci. **48**, 300 (2012).

[131] A. Montessori, M. Lauricella, A. Tiribocchi, M. Durve, M. La Rocca, G. Amati, F. Bonaccorso, and S. Succi, Thread-safe lattice Boltzmann for high-performance computing on GPUs, J. Comput. Sci. **74**, 102165 (2023).

[132] M. Lauricella, A. Mukherjee, L. Brandt, S. Succi, D. Izbassarov, and A. Montessori, accLB: A high-performance lattice-Boltzmann code for multiphase turbulence on multi-GPU architectures, Proc. Comput. Sci. **267**, 40 (2025).

[133] G. Amati, S. Succi, and R. Piva, Massively parallel lattice-Boltzmann simulation of turbulent channel flow, Int. J. Mod. Phys. C **08**, 869 (1997).

[134] H. Chen, S. Kandasamy, S. Orszag, R. Shock, S. Succi, and V. Yakhot, Extended Boltzmann kinetic equation for turbulent flows, Science **301**, 633 (2003).

[135] C. Peng, N. Geneva, Z. Guo, and L.-P. Wang, Direct numerical simulation of turbulent pipe flow using the lattice Boltzmann method, J. Comput. Phys. **357**, 16 (2018).

[136] S. A Orszag and G. S. Patterson Jr, Numerical simulation of three-dimensional homogeneous isotropic turbulence, Phys. Rev. Lett. **28**, 76 (1972).

[137] Y. Li, E. Perlman, M. Wan, Y. Yang, C. Meneveau, R. Burns, S. Chen, A. Szalay, and G. Eyink, A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence, J. Turbul. **9**, N31 (2008).
[138] P. K. Yeung, K. Ravikumar, R. Uma-Vaideswaran, D. L. Dotson, K. R. Sreenivasan, S. B. Pope, C. Meneveau, and S. Nichols, Small-scale properties from exascale computations of turbulence on a periodic cube, J. Fluid Mech. **1019**, R2 (2025).
[139] D. Modesti and S. Pirozzoli, Direct numerical simulation of forced thermal convection in square ducts up to $\mathrm{Re}_\tau \approx 2000$, J. Fluid Mech. **941**, A16 (2022).
[140] S. Pirozzoli, On the streamwise velocity variance in the near-wall region of turbulent flows, J. Fluid Mech. **989**, A5 (2024).
[141] G. A. Voth and A. Soldati, Anisotropic particles in turbulence, Annu. Rev. Fluid Mech. **49**, 249 (2017).
[142] F. Risso, Agitation, mixing, and transfers induced by bubbles, Annu. Rev. Fluid Mech. **50**, 25 (2018).
[143] J. Wang, M. Alipour, G. Soligo, A. Soldati, M. De Paoli, F. Picano, and A. Soldati, Short-range exposure to airborne virus transmission and current guidelines, Proc. Natl. Acad. Sci. USA **118**, e2105279118 (2021).
[144] L. Bourouiba, The fluid dynamics of disease transmission, Annu. Rev. Fluid Mech. **53**, 473 (2021).
[145] L. Deike, Mass transfer at the ocean–atmosphere interface: The role of wave breaking, droplets, and bubbles, Annu. Rev. Fluid Mech. **54**, 191 (2022).
[146] F. Viola, G. Del Corso, R. De Paulis, and R. Verzicco, GPU accelerated digital twins of the human heart open new routes for cardiovascular research, Sci. Rep. **13**, 8230 (2023).
[147] C. Marchioli, M. E. Rosti, and G. Verhille, Flexible fibers in turbulence, Annu. Rev. Fluid Mech. **58**, 167 (2026).
[148] G. Soligo, A. Roccon, and A. Soldati, Effect of surfactant-laden droplets on turbulent flow topology, Phys. Rev. Fluids **5**, 073606 (2020).
[149] G. Lupo, P. Wellens, and P. Costa, CaNS-Fizzy: A GPU-accelerated finite difference solver for turbulent two-phase flows, J. Open Source Soft. **10**, 8076 (2025).
[150] A. M. Bilondi, N. Scapin, L. Brandt, and P. Mirbod, Turbulent convection in emulsions: The Rayleigh–Bénard configuration, J. Fluid Mech. **999**, A4 (2024).
[151] H.-R. Liu, K. L. Chong, C. S. Ng, R. Verzicco, and D. Lohse, Enhancing heat transport in multiphase Rayleigh–Bénard turbulence by changing the plate–liquid contact angles, J. Fluid Mech. **933**, R1 (2022).
[152] D. J. Dhas and C. Marchioli, Orientational dynamics of long flexible fibers in wall-bounded turbulence, J. Fluids Eng. **147**, 071102 (2025).
[153] D. Di Giusto and C. Marchioli, Turbulence modulation by slender fibers, Fluids **7**, 255 (2022).
[154] W. Su, H. Zhang, S. Fu, X. Xiang, and L. Wang, Particle-resolved direct numerical simulation of particle-laden turbulence modulation with high stokes number monodisperse spheres, Phys. Fluids **35**, 105120 (2023).
[155] P. Karnakov, F. Wermelinger, S. Litvinov, and P. Koumoutsakos, Aphros: High performance software for multiphase flows with large scale bubble and drop clusters, in *Proceedings of the Platform for Advanced Scientific Computing Conference (PASC20)* (ACM Press, New York, 2020), pp. 1–10.
[156] R. Porcu, J. Musser, A. S. Almgren, J. B. Bell, W. D. Fullmer, and D. Rangarajan, MFIX-Exa: CFD-DEM simulations of thermodynamics and chemical reactions in multiphase flows, Chem. Eng. Sci. **273**, 118614 (2023).
[157] P. Zaspel and M. Griebel, Solving incompressible two-phase flows on multi-GPU clusters, Comput. Fluids **80**, 356 (2013).
[158] W. Aniszewski, T. Arrufat, M. Crialesi-Esposito, S. Dabiri, D. Fuster, Y. Ling, J. Lu, L. Malan, S. Pal, and R. Scardovelli, Parallel, robust, interface simulator (PARIS), Comput. Phys. Commun. **263**, 107849 (2021).
[159] F. Pelusi, M. Lulli, M. Sbragaglia, and M. Bernaschi, TLBfind: A thermal lattice Boltzmann code for concentrated emulsions with FINite-size droplets, Comput. Phys. Commun. **273**, 108259 (2022).
[160] T. Brandvik and G. Pullan, Acceleration of a 3D euler solver using commodity graphics hardware, in *Proceedings of the 46th AIAA Aerospace Sciences Meeting* (AIAA, Reston, VA, 2008), p. 607.

[161] E. Elsen, P. LeGresley, and E. Darve, Large calculation of the flow over a hypersonic vehicle using a GPU, J. Comput. Phys. **227**, 10148 (2008).
[162] A. Corrigan, F. F. Camelli, R. Löhner, and J. Wallin, Running unstructured grid-based CFD solvers on modern graphics hardware, Int. J. Numer. Methods Fluids **66**, 221 (2011).
[163] G. Dang, S. Liu, T. Guo, J. Duan, and X. Li, Direct numerical simulation of compressible turbulence accelerated by graphics processing unit: An open-source high accuracy accelerated computational fluid dynamic software, Phys. Fluids **34**, 126106 (2022).
[164] F. De Vanna, F. Avanzi, M. Cogo, S. Sandrin, M. Bettencourt, F. Picano, and E. Benini, URANOS: A GPU accelerated Navier-Stokes solver for compressible wall-bounded flows, Comput. Phys. Commun. **287**, 108717 (2023).
[165] K. Wang, S. Bose, and C. Ivey, GPU-accelrated full-wheel large-eddy simulations of a transonic fan stage, in *Turbo Expo: Power for Land, Sea, and Air* (ASME, New York, 2024), Vol. 88070, p. V12CT32A042.
[166] Y. Kim, D. Ghosh, E. M. Constantinescu, and R. Balakrishnan, GPU-accelerated DNS of compressible turbulent flows, Comput. Fluids **251**, 105744 (2023).
[167] A. Ceci, A. Palumbo, J. Larsson, and S. Pirozzoli, On low-frequency unsteadiness in swept shock wave-boundary layer interactions, J. Fluid Mech. **956**, R1 (2023).
[168] P. C. Boldini, R. Hirai, P. Costa, J. W. R. Peeters, and R. Pecnik, CUBENS: A GPU-accelerated high-order solver for wall-bounded flows with non-ideal fluids, Comput. Phys. Commun. **309**, 109507 (2025).
[169] E. Slaughter, W. Lee, S. Treichler, M. Bauer, and A. Aiken, Regent: A high-productivity programming language for HPC with logical regions, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'15)* (IEEE, Los Alamitos, CA, 2015), pp. 1–12.
[170] S. Pirozzoli, Generalized conservative approximations of split convective derivative operators, J. Comput. Phys. **229**, 7180 (2010).
[171] J. Choi, Y. Kim, and H.-Y. Yeom, Overcoming GPU memory capacity limitations in hybrid MPI implementations of CFD, in *Internet and Distributed Computing Systems,* edited by R. Montella, A. Ciaramella, G. Fortino, A. Guerrieri, and A. Liotta (Springer International, Cham, 2019), pp. 100–111.
[172] J. Lai, H. Yu, Z. Tian, and H. Li, Hybrid MPI and CUDA parallelization for CFD applications on multi-GPU HPC clusters, Sci. Program. **2020**, 8862123 (2020).
[173] M. Kurz, D. Kempf, M. P Blind, P. Kopper, P. Offenhäuser, A. Schwarz, S. Starr, J. Keim, and A. Beck, Galæxi: Solving complex compressible flows with high-order discontinuous Galerkin methods on accelerator-based systems, Comput. Phys. Commun. **306**, 109388 (2025).
[174] D. A Bezgin, A. B. Buhendwa, and N. A. Adams, Jax-fluids 2.0: Towards HPC for differentiable CFD of compressible two-phase flows, Comput. Phys. Commun. **308**, 109433 (2025).
[175] P. Grete, J. C. Dolence, J. M. Miller, J. Brown, B. Ryan, A. Gaspar, F. Glines, S. Swaminarayan, J. Lippuner, and C. Solomon, Parthenon—A performance portable block-structured adaptive mesh refinement framework, Int. J. High Perform. Comput. Appl. **37**, 465 (2023).
[176] F. D. Witherden, A. M. Farrington, and P. 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**, 3028 (2014).
[177] L. Jofre, A. M. A. Abdellatif, and G. Oyarzun Altamirano, RHEA: An open-source reproducible hybrid-architecture flow solver engineered for academia, J. Open Source Softw. **8**, 4637 (2023).
[178] W. Xue, H. Wang, and C. J. Roy, CPU–GPU heterogeneous code acceleration of a finite volume computational fluid dynamics solver, Future Gener. Comput. Syst. **158**, 367 (2024).
[179] F. Salvadore, G. Soldati, A. Ceci, G. Rossi, A. Memmolo, G. D. Posta, D. Modesti, S. Sathyanarayana, M. Bernardini, and S. Pirozzoli, STREAmS-2.1: Supersonic turbulent accelerated Navier-Stokes solver version 2.1, Comput. Phys. Commun. **314**, 109652 (2025).
[180] F. De Vanna and G. Baldan, URANOS-2.0: Improved performance, enhanced portability, and model extension towards exascale computing of high-speed engineering flows, Comput. Phys. Commun. **303**, 109285 (2024).

[181] J. Romero, J. Crabill, J. E. Watkins, F. D. Witherden, and A. Jameson, ZEFR: A GPU-accelerated high-order solver for compressible viscous flows using the flux reconstruction method, Comput. Phys. Commun. **250**, 107169 (2020).
[182] M. Karp, R. Stanly, T. Mukha, L. Galimberti, S. Toosi, H. Song, L. Dalcin, S. Rezaeiravesh, N. Jansson, S. Markidis, M. Parsani, S. Bose, L. Sanjiva, and P. Schlatter, Effects of lower floating-point precision on scale-resolving numerical simulations of turbulence, J. Comput. Phys. **549**, 114600 (2026).
[183] G. A. Bres, S. T. Bose, C. B. Ivey, M. Emory, and F. Ham, GPU-accelerated large-eddy simulations of supersonic jets from twin rectangular nozzle, in *28th AIAA/CEAS Aeroacoustics 2022 Conference* (AIAA, Reston, VA, 2022), p. 3001.
[184] R. Saurel, F. Petitpas, and R. Abgrall, Modelling phase transition in metastable liquids: Application to cavitating and flashing flows, J. Fluid Mech. **607**, 313 (2008).
[185] F. Petitpas, J. Massoni, R. Saurel, E. Lapebie, and L. Munier, Diffuse interface model for high speed cavitating underwater systems, Int. J. Multiph. Flow **35**, 747 (2009).
[186] D. D. Joseph, J. Belanger, and G. S. Beavers, Breakup of a liquid drop suddenly exposed to a high-speed airstream, Int. J. Multiph. Flow **25**, 1263 (1999).
[187] Q. Li, Y. Lv, and L. Fu, A high-order diffuse-interface method with TENO-THINC scheme for compressible multiphase flows, Int. J. Multiph. Flow **173**, 104732 (2024).
[188] S. H. Bryngelson, K. Schmidmayer, V. Coralic, J. C. Meng, K. Maeda, and T. Colonius, MFC: An open-source high-order multi-component, multi-phase, and multi-scale compressible flow solver, Comput. Phys. Commun. **266**, 107396 (2021).
[189] W. Zhang, A. Almgren, V. Beckner, J. Bell, J. Blaschke, Cy Chan, M. Day, B. Friesen, K. Gott, and G. Daniel, AMReX: A framework for block-structured adaptive mesh refinement, J. Open Source Softw. **4**, 1370 (2019).
[190] W. Zhang, A. Myers, K. Gott, A. Almgren, and J. Bell, Amrex: Block-structured adaptive mesh refinement for multiphysics applications, Int. J. High Perform. Comput. Appl. **35**, 508 (2021).
[191] A. Radhakrishnan, H. Le Berre, B. Wilfong, J.-S. Spratt, M. Rodriguez Jr., T. Colonius, and S. H. Bryngelson, Method for portable, scalable, and performant GPU-accelerated simulation of multiphase compressible flow, Comput. Phys. Commun. **302**, 109238 (2024).
[192] S. Rao, B. Chen, and X. Xu, Heterogeneous CPU-GPU parallelization for modeling supersonic reacting flows with detailed chemical kinetics, Comput. Phys. Commun. **300**, 109188 (2024).
[193] M. T. H. de Frahan, J. S. Rood, M. S. Day, H. Sitaraman, S. Yellapantula, B. A. Perry, R. W. Grout, A. Almgren, W. Zhang, J. B. Bell, and J. H. Chen, PeleC: An adaptive mesh refinement solver for compressible reacting flows, Int. J. High Perform. Comput. Appl. **37**, 115 (2023).
[194] M. T. Henry de Frahan, L. Esclapez, J. Rood, N. T. Wimer, P. Mullowney, B. A. Perry, L. Owen, H. Sitaraman, S. Yellapantula, M. Hassanaly, M. J. Rahimi, M. J. Martin, O. A. Doronina, N. A. Sreejith, M. Rieth, W. Ge, R. Sankaran, A. S. Almgren, W. Zhang, J. B. Bell, *et al.*, The Pele simulation suite for reacting flows at exascale, in *Proceedings of the 2024 SIAM Conference on Parallel Processing for Scientific Computing (PP)* (SIAM, Philedelphia, PA, 2024), pp. 13–25.
[195] H. Sitaraman, S. Yellapantula, M. T. H. de Frahan, B. Perry, J. Rood, R. Grout, and M. Day, Adaptive mesh based combustion simulations of direct fuel injection effects in a supersonic cavity flame-holder, Combust. Flame. **232**, 111531 (2021).
[196] L. Placco, N. Cogo, M. Bernardini, A. Aboudan, F. Ferri, and F. Picano, Large-eddy simulation of the unsteady supersonic flow around a mars entry capsule at different angles of attack, Aerosp. Sci. Technol. **143**, 108709 (2023).
[197] M. Cogo, D. Modesti, M. Bernardini, and F. Picano, DNS of supersonic turbulent boundary layers over rough surfaces, in *APS DFD Meeting Abstracts* (APS, New York, 2024), pp. X31–011.
[198] J. H. Chen, Petascale direct numerical simulation of turbulent combustion-fundamental insights towards predictive models, Proc. Combust. Inst. **33**, 99 (2011).
[199] J. M. Levesque, R. Sankaran, and R. Grout, Hybridizing S3D into an exascale application using OpenACC: An approach for moving to multi-petaflops and beyond, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC12)* (IEEE, Los Alamitos, CA, 2012), pp. 1–11.

[200] Z.-M. Liao, L.-B. Chen, Z.-H. Wan, N.-S. Liu, and X.-Y. Lu, GPU acceleration of four-way coupled PP-DNS for compressible particle-laden wall turbulence, Int. J. Multiph. Flow **176**, 104840 (2024).
[201] M. Xiao, A. Ceci, P. Costa, J. Larsson, and S. Pirozzoli, CaLES: A GPU-accelerated solver for large-eddy simulation of wall-bounded flows, Comput. Phys. Commun. **310**, 109546 (2025).
[202] T. Zahtila, W. Lu, L. Chan, and A. Ooi, A systematic study of the grid requirements for a spectral element method solver, Comput. Fluids **251**, 105745 (2023).
[203] Computational and Information Systems Laboratory, *Derecho: HPE Cray E. X. System (University Community Computing)*, Technical Report, NSF National Center for Atmospheric Research, Boulder, CO (2023).
[204] Y. Ogura and N. A. Phillips, Scale analysis of deep and shallow convection in the atmosphere, J. Atmos. Sci. **19**, 173 (1962).
[205] J. Schalkwijk, E. J. Griffith, F. H. Post, and H. J. J. Jonker, High-performance simulations of turbulent clouds on a desktop PC: Exploiting the GPU, Bull. Am Meteorol. Soc. **93**, 307 (2012).
[206] J. Schalkwijk, H. J. J. Jonker, A. P. Siebesma, and E. Van Meijgaard, Weather forecasting using GPU-based large-dddy simulations, Bull. Am Meteorol. Soc. **96**, 715 (2015).
[207] J. Schalkwijk, H. J. J. Jonker, A. P. Siebesma, and F. C. Bosveld, A year-long large-eddy simulatoin of the weather over Cabau: An overview, Mon. Weather Rev. **143**, 828 (2015).
[208] C. C. van Heerwaarden, B. J. H. van Stratum, T. Heus, J. A. Gibbs, E. Fedorovich, and J. P. Mellado, MicroHH 1.0: A computational fluid dynamics code for direct numerical simulation and large-eddy simulation of atmospheric boundary layer flows, Geosci. Model Dev. **10**, 3145 (2017).
[209] J. A. Sauer and E. Muñoz Esparza, The FastEddy resident-GPU accelerated large-eddy simulation framework: Model formulation, dynamical-core validation and performance benchmarks, J. Adv. Model. Earth Syst. **12**, e2020MS002100 (2020).
[210] B. Maronga, M. Gryschka, R. Heinze, F. Hoffmann, F. Kanani-Sühring, M. Keck, K. Ketelsen, M. O. Letzel, M. Sühring, and S. Raasch, The parallelized large-eddy simulation model (PALM) version 4.0 for atmospheric and oceanic flows: Model formulation, recent developments, and future perspectives, Geosci. Model Dev. **8**, 2515 (2015).
[211] J. Dennis, J. Sun, S. Voelz, G. Bryan, and D. Richter, A portable and efficient Lagrangian particle capability for idealized atmospheric phenomena, in *Proceedings of the Platform for Advanced Scientific Computing Conference (PASC24)* (ACM Press, New York, 2024), pp. 1–11.
[212] L. Esclapez, L. Soucasse, C. Jungbacker, F. Jansson, S. de Roode, P. Costa, G. van den Oord, and A. Sclocco, Accelerating the dutch atmospheric large-eddy simulation (DALES) model with OpenACC, in *Proceedings of the 39th IEEE International Parallel &Distributed Processing Symposium (IPDPS'25)* (IEEE, Los Alamitos, CA, 2025).
[213] C. C. Kiris, M. F. Barad, J. A. Housman, E. Sozer, C. Brehm, and S. Moini-Yekta, The LAVA computational fluid dynamics solver, in *Proceedings of the 52nd AIAA Aerospace Sciences Meeting* (AIAA, Reston, VA, 2014), p. 0070.
[214] C. C. Kiris, J. A. Housman, M. F. Barad, C. Brehm, E. Sozer, and S. Moini-Yekta, Computational framework for launch, ascent, and vehicle aerodynamics LAVA, Aerosp. Sci. Technol. **55**, 189 (2016).
[215] N. Peters, RotLES: A Rotorcraft-based Lattice Boltzmann LES Code for Rapid Analysis and Design Space Exploration, Technical Report, National Aeronautics and Space Administration (2025).
[216] C. W. Jackson, D. Appelhans, J. M. Derlaga, and P. G. Buning, GPU implementation of the OVERFLOW CFD code, in *AIAA SCITECH 2024 Forum* (AIAA, Reston, VA, 2024), p. 0042.
[217] L. Placco, G. Soldati, M. Bernardini, and F. Picano, On flight instabilities of capsule-rigid parachute system during supersonic planetary descent, Aerosp. Sci. Technol. **160**, 110026 (2025).
[218] S. C. Spiegel, D. A. Yoder, J. R. DeBonis, H. T. Huynh, G. Heinlein, M. R. Borghi, and N. J. Georgiadis, New capabilities and improvements to the high-order Glenn flux reconstruction code, in *AIAA 2025-0061, Session: NASA's Revolutionary Computational Aerosciences, AIAA SCITECH 2025 Forum, Orlando, FL* (AIAA, 2025).
[219] From supercomputers to wind tunnels: NASA road to Artemis II (2025), https://www.nasa.gov/directorates/esdmd/common-exploration-systems-development-division/space-launch-system/from-supercomputers-to-wind-tunnels-nasas-road-to-artemis-ii/.

[220] G. H. Bryan and J. M. Fritsch, A benchmark simulation for moist nonhydrostatic numerical models, Mon. Weather Rev. **130**, 2917 (2002).

[221] T. Heus, C. C. van Heerwaarden, H. J. J. Jonker, A. P. Siebesma, S. Axelsen, K. Van Den Dries, O. Geoffroy, A. F. Moene, D. Pino, and S. R. De Roode, Formulation of the dutch atmospheric large-eddy simulation (DALES) and overview of its applications, Geosci. Model Dev. **3**, 415 (2010).

[222] J. D. Albertson, Large eddy simulation of land-atmosphere interaction, Ph.D. thesis, University of California, Davis, 1996.

[223] M. T. Nguyen, P. Castonguay, and E. Laurendeau, GPU parallelization of multigrid RANS solver for three-dimensional aerodynamic simulations on multiblock grids, J. Supercomput. **75**, 2562 (2019).

[224] M. Lee, N. Malaya, and R. D. Moser, Petascale direct numerical simulation of turbulent channel flow on up to 786k cores, in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC13)* (IEEE, Los Alamitos, CA, 2013), pp. 1–11.

[225] M. Turisini, G. Amati, and M. Cestari, LEONARDO: A pan-European pre-exascale supercomputer for HPC and AI applications, arXiv:2307.16885.

[226] Z. Gong, G. Deng, C. An, Z. Wu, and X. Fu, A high order finite difference solver for simulations of turbidity currents with high parallel efficiency, Comput. Math. Appl. **128**, 21 (2022).

[227] C. Canuto, M. Y. Hussaini, A. M. Quarteroni, and T. A. Zang, *Spectral Methods in Fluid Dynamics* (Springer-Verlag, Berlin, 1988).

[228] A. Roccon, A GPU-ready pseudo-spectral method for direct numerical simulations of multiphase turbulence, Proc. Comput. Sci. **240**, 17 (2024).

[229] A. G. Chatterjee, M. K. Verma, A. Kumar, R. Samtaney, B. Hadri, and R. Khurram, Scaling of a fast fourier transform and a pseudo-spectral fluid solver up to 196608 cores, J. Parallel. Distr. Com. **113**, 77 (2018).

[230] K. Czechowski, C. Battaglino, C. McClanahan, K. Iyer, P.-K. Yeung, and R. Vuduc, On the communication complexity of 3D FFTs and its implications for exascale, in *Proceedings of the ACM International Conference on Supercomputing (ICS)* (ACM Press, New York, 2012), pp. 205–214.

[231] L. Dalcin, M. Mortensen, and D. E. Keyes, Fast parallel multidimensional FFT using advanced MPI, J. Parallel. Distr. Com. **128**, 137 (2019).

[232] D. Pekurovsky, P3DFFT: A framework for parallel computations of Fourier transforms in three dimensions, SIAM J. Comput. **34**, C192 (2012).

[233] J. Romero, P. Costa, and M. Fatica, Distributed-memory simulations of turbulent flows on modern GPU systems using an adaptive pencil decomposition library, in *Proceedings of the Platform for Advanced Scientific Computing Conference (PASC22)* (ACM Press, New York, 2022), pp. 1–11.

[234] R. Uma-Vaideswaran, J. Romero, D. L. Dotson, D. Appelhans, and P. K. Yeung, A peak performance model for all-to-all on hierarchical systems and its applications, in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC25)* (IEEE, Los Alamitos, CA, 2025), pp. 1442–1451.

[235] N. Okamoto, T. Ishihara, M. Yokokawa, and Y. Kaneda, Effects of finite arithmetic precision on large-scale direct numerical simulation of box turbulence by spectral method, Phys. Rev. Fluids **10**, 064603 (2025).

[236] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, Efficient processing of deep neural networks: A tutorial and survey, Proc. IEEE **105**, 2295 (2017).

[237] R. Sakamoto, M. Kondo, K. Fujita, T. Ichimura, and K. Nakajima, The effectiveness of low-precision floating arithmetic on numerical codes: A case study on power consumption, in *Proceedings of the Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region Workshops* (ACM Press, New York, 2020), pp. 199–206.

[238] K. Kulkarni, S. Kemmler, A. Schwarz, G. Gedik, Y. Chen, D. Papageorgiou, I. Kavroulakis, and R. Iakymchuk, Harvesting energy consumption on european HPC systems: Sharing experience from the CEEC project (unpublished).

[239] Y. Arafa, A. ElWazir, A. ElKanishy, Y. Aly, A. Elsayed, A.-H. Badawy, G. Chennupati, S. Eidenbenz, and N. Santhi, Verified instruction-level energy consumption measurement for NVIDIA GPUs, in

*Proceedings of the 17th ACM International Conference on Computing Frontiers* (ACM Press, New York, 2020), pp. 60–70.

[240] A. Kashi, H. Lu, W. Brewer, D. Rogers, M. Matheson, M. Shankar, and F. Wang, Mixed-precision numerics in scientific applications: Survey and perspectives, arXiv:2412.19322.

[241] A. Abdelfattah, H. Anzt, E. G. Boman, E. Carson, T. Cojean, J. Dongarra, A. Fox, M. Gates, N. J Higham, X. Li, *et al.*, A survey of numerical linear algebra methods utilizing mixed-precision arithmetic, Int. J. High Perform. Comput. Appl. **35**, 344 (2021).

[242] M. Bernardini, S. Pirozzoli, and P. Orlandi, Velocity statistics in turbulent channel flow up to $\mathrm{Re}_\tau = 4000$, J. Fluid Mech. **742**, 171 (2014).

[243] B. Wilfong, A. Radhakrishnan, H. Le Berre, D. J. Vickers, T. Prathi, N. Tselepidis, B. Dorschner, R. Budiardja, B. Cornille, A. Abbott, F. Schäfer, and S. H. Bryngelson, Simulating many-engine spacecraft: Exceeding 1 quadrillion degrees of freedom via information geometric regularization, in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC25)* (IEEE, Los Alamitos, CA, 2025), pp. 14–24.

[244] R. Vacondio, C. Altomare, M. De Leffe, X. Hu, D. Le Touzé, S. Lind, J.-C. Marongiu, S. Marrone, B. D. Rogers, and A. Souto-Iglesias, Grand challenges for smoothed particle hydrodynamics numerical schemes, Comput. Part. Mech. **8**, 575 (2021).

[245] J. Pozorski and M. Olejnik, Smoothed particle hydrodynamics of multiphase flows: An overview, Acta Mech. **235**, 1685 (2024).

[246] A. Tiribocchi, M. Durve, M. Lauricella, A. Montessori, J.-M. Tucny, and S. Succi, Lattice Boltzmann simulations for soft flowing matter, Phys. Rep. **1105**, 1 (2025).

[247] J. Capecelatro and O. Desjardins, An Euler-Lagrange strategy for simulating particle-laden flows, J. Comput. Phys. **238**, 1 (2013).

[248] D. Zwick and S. Balachandar, A scalable Euler–Lagrange approach for multiphase flow simulation on spectral elements, Int. J. High Perform. Comput. Appl. **34**, 316 (2020).

[249] K. Zhai, T. Banerjee, D. Zwick, J. Hackl, and S. Ranka, Dynamic load balancing for compressible multiphase turbulence, in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC18)* (IEEE, Los Alamitos, CA, 2018), pp. 318–327.

[250] P. Valero-Lara, F. D. Igual, M. Prieto-Matías, A. Pinelli, and J. Favier, Accelerating fluid–solid simulations (lattice-Boltzmann and immersed-boundary) on heterogeneous architectures, J. Comput. Sci. **10**, 249 (2015).

[251] X. Zhang, F. Zonta, Z. F Tian, G. J. Nathan, R. C. Chin, and A. Soldati, Dynamics of semi-and neutrally-buoyant particles in thermally stratified turbulent channel flow, Int. J. Multiph. Flow **139**, 103595 (2021).

[252] J. Sweet, D. H. Richter, and D. Thain, GPU acceleration of Eulerian–Lagrangian particle-laden turbulent flow simulations, Int. J. Multiph. Flow **99**, 437 (2018).

[253] M. Uhlmann, An immersed boundary method with direct forcing for the simulation of particulate flows, J. Comput. Phys. **209**, 448 (2005).

[254] S. O. Unverdi and G. Tryggvason, A front-tracking method for viscous, incompressible, multi-fluid flows, J. Comput. Phys. **100**, 25 (1992).

[255] A. M. Roma, C. S. Peskin, and M. J. Berger, An adaptive version of the immersed boundary method, J. Comput. Phys. **153**, 509 (1999).

[256] J. C. B. de Motta, P. Costa, J. J. Derksen, C. Peng, L.-P. Wang, W.-P. Breugem, J.-L. Estivalèzes, S. Vincent, E. Climent, and P. Fede, Assessment of numerical methods for fully resolved simulations of particle-laden turbulent flows, Comput. Fluids **179**, 1 (2019).

[257] P. Karnakov, F. Wermelinger, M. Chatzimanolakis, S. Litvinov, and P. Koumoutsakos, A high performance computing framework for multiphase, turbulent flows on structured grids, in *Proceedings of the Platform for Advanced Scientific Computing Conference* (ACM Press, New York, 2019).

[258] M. Chatzimanolakis, P. Weber, F. Wermelinger, and P. Koumoutsakos, CubismAMR—AC library for distributed block-structured adaptive mesh refinement, arXiv:2206.07345.

[259] P. Hadjidoukas, D. Rossinelli, F. Wermelinger, J. Sukys, U. Rasthofer, C. Conti, B. Hejazialhosseini, and P. Koumoutsakos, High throughput simulations of two-phase flows on Blue Gene/Q, in *Parallel Computing: On the Road to Exascale* (IOS Press, Bristol, UK, 2016), pp. 767–776.

[260] E. Aulisa, S. Manservisi, R. Scardovelli, and S. Zaleski, Interface reconstruction with least-squares fit and split advection in three-dimensional Cartesian geometry, J. Comput. Phys. **225**, 2301 (2007).
[261] R. Reddy and R. Banerjee, GPU accelerated VOF based multiphase flow solver and its application to sprays, Comput. Fluids **117**, 287 (2015).
[262] S. Ii, K. Sugiyama, S. Takeuchi, S. Takagi, Y. Matsumoto, and F. Xiao, An interface capturing method with a continuous function: The thinc method with multi-dimensional reconstruction, J. Comput. Phys. **231**, 2328 (2012).
[263] M. Griebel and P. Zaspel, A multi-gpu accelerated solver for the three-dimensional two-phase incompressible Navier-Stokes equations, Comput. Sci. Res. Dev. **25**, 65 (2010).
[264] In standard level-set methods, the level-set function is defined to be a signed distance function calculated from the distance at the interface that separates the two fluids.
[265] J. Kelly, GPU-accelerated simulation of two-phase incompressible fluid flow using a level-set method for interface capturing, in *ASME International Mechanical Engineering Congress & Exposition (IMECE09)* (ASME, New York, 2009), Vol. 9, p. 2221.
[266] S. Fu, Z. Hao, W. Su, H. Zhang, and L. Wang, Level-set lattice Boltzmann method for interface-resolved simulations of immiscible two-phase flow, Phys. Rev. E **110**, 045309 (2024).
[267] F. Gibou, R. Fedkiw, and S. Osher, A review of level-set methods and some recent applications, J. Comput. Phys. **353**, 82 (2018).
[268] Y. Li, B. Zhou, and X. Hu, A two-grid method for level-set based topology optimization with GPU-acceleration, J. Comput. Appl. Math. **389**, 113336 (2021).
[269] D. Reska and M. Kretowski, GPU-accelerated image segmentation based on level sets and multiple texture features, Multimedia Tools Appl. **80**, 5087 (2021).
[270] D. Reska and M. Kretowski, GPU-accelerated lung CT segmentation based on level sets and texture analysis, Sci. Rep. **14**, 1444 (2024).
[271] J. A. Sethian, A fast marching level set method for monotonically advancing fronts, Proc. Natl. Acad. Sci. USA **93**, 1591 (1996).
[272] Y.-H. R. Tsai, L.-T. Cheng, S. Osher, and H.-K. Zhao, Fast sweeping algorithms for a class of Hamilton–Jacobi equations, SIAM J. Numer. Anal. **41**, 673 (2003).
[273] H. Zhao, Parallel implementations of the fast sweeping method, J. Comput. Math. **25**, 421, (2007).
[274] M. Detrixhe, F. Gibou, and C. Min, A parallel fast sweeping method for the Eikonal equation, J. Comput. Phys. **237**, 46 (2013).
[275] A. Roccon, F. Zonta, and A. Soldati, Phase-field modeling of complex interface dynamics in drop-laden turbulence, Phys. Rev. Fluids **8**, 090501 (2023).
[276] S. Mirjalili, C. B. Ivey, and A. Mani, A conservative diffuse interface method for two-phase flows with provable boundedness properties, J. Comput. Phys. **401**, 109006 (2020).
[277] S. S. Jain, Accurate conservative phase-field method for simulation of two-phase flows, J. Comput. Phys. **469**, 111529 (2022).
[278] J. Kwack, J. Tramm, C. Bertoni, Y. Ghadar, B. Homerding, E. Rangel, C. Knight, and S. Parker, Evaluation of performance portability of applications and mini-apps across AMD, Intel and nVidia GPUs, in *Proceedings of the 2021 International Workshop on Performance, Portability & Productivity in HPC (P3HPC)* (IEEE, Los Alamitos, CA, 2021), pp. 45–56.
[279] J. Dongarra, M. Gates, P. Luszczek, and S. Tomov, Translational process: Mathematical software perspective, J. Comput. Sci. **52**, 101216 (2021).
[280] M. Lehmann, M. J. Krause, G. Amati, M. Sega, H. Harting, and S. Gekle, Accuracy and performance of the lattice-Boltzmann method with 64-bit, 32-bit, and customized 16-bit number formats, Phys. Rev. E **106**, 015308 (2022).
[281] M. Karp, R. Stanly, H. Song, T. Mukha, L. Galimberti, S. Toosi, L. Dalcin, S. Rezaeirsavesh, M. Munsch, N. Jansson, S. Markidis, M. Parsani, S. T. Bose, S. K. Lele, and P. Schlatter, Sensitivity of numerical simulations of turbulence to lower floating-point precision, in *Proceedings of the Summer Program 2022 of the Center Turbulence Research* (Stanford University, Stanford, CA, 2024).
[282] H. Ootomo, K. Ozaki, and R. Yokota, DGEMM on integer matrix multiplication unit, Int. J. High Perform. Comput. Appl. **38**, 297 (2024).

[283] H. Liu, J. Li, and Y. Wang, A pilot study on tunable precision emulation via automatic BLAS offloading, arXiv:2503.22875.

[284] A. Schwarz, A. Anders, C. Brower, H. Bayraktar, J. Gunnels, K. Clark, R. G. Xu, S. Rodriguez, S. Cayrols, P. Tabaszewski, *et al.*, Guaranteed DGEMM accuracy while using reduced precision tensor cores through extensions of the Ozaki scheme, in *Proceedings of the Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region Workshops* (ACM Press, New York, 2026), pp. 91–101.

[285] P. Koumoutsakos, On roads less between AI and computational science, Nat. Rev. Phys. **6**, 342 (2024).

[286] R. Vinuesa and S. L. Brunton, Enhancing computational fluid dynamics with machine learning, Nat. Comput. Sci. **2**, 358 (2022).

[287] M. Curcic, A parallel Fortran framework for neural networks and deep learning, in *Proceedings of the ACM SIGPLAN Fortran Forum* (ACM Press, New York, 2019), Vol. 38, pp. 4–21.

[288] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, Memory devices and applications for in-memory computing, Nat. Nanotechnol. **15**, 529 (2020).

[289] Z. Sun, S. Kvatinsky, X. Si, A. Mehonic, Y. Cai, and R. Huang, A full spectrum of computing-in-memory technologies, Nat. Electron. **6**, 823 (2023).

[290] https://github.com/aroccon/GPU_DNS_LES_codes.git.