Future Generation Computer Systems 152 (2024) 1–16

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

On floating point precision in computational fluid dynamics using OpenFOAM

F. Brogi [a,*], S. Bnà [b], G. Boga [b,c], G. Amati [b], T. Esposti Ongaro [a], M. Cerminara [a]

[a] Istituto Nazionale di Geofisica e Vulcanologia, Pisa, Italy
[b] CINECA Supercomputing Centre, Via Magnanelli 6/3, 40033 Casalecchio di Reno, Italy
[c] DIEF, University of Modena and Reggio Emilia, 41125 Modena, Italy

ARTICLE INFO

Keywords:
Floating point precision
Mixed precision
Computational fluid dynamics
Computational performance and scalability
OpenFOAM
GPU

ABSTRACT

Thanks to the computational power of modern cluster machines, numerical simulations can provide, with an unprecedented level of details, new insights into fluid mechanics. However, taking full advantage of this hardware remains challenging since data communication remains a significant bottleneck to reaching peak performances. Reducing floating point precision is a simple and effective way to reduce data movement and improve the computational speed of most applications. Nevertheless, special care needs to be taken to ensure the quality and convergence of computed solutions, especially when dealing with complex fluid simulations. In this work, we analyse the impact of reduced (single and mixed compared to double) precision on computational performance and accuracy for computational fluid dynamics. Using the open source library OpenFOAM, we consider incompressible, compressible, and multiphase fluid solvers for testing on relevant benchmarks for flows in the laminar and turbulent regime and in the presence of shock waves. Computational gain and changes in the scalability of applications in reduced precision are also discussed. In particular, an ad hoc theoretical model for the strong scaling allows us to interpret and understand the observed behaviours, as a function of floating point precision and hardware specifics. Finally, we show how reduced precision can significantly speed up a hybrid CPU–GPU implementation, made available to OpenFOAM end-users recently, that simply relies on a GPU linear algebra solver developed by hardware vendors.

1. Introduction

An increasingly wider community is choosing the open source software OpenFOAM [1,2] as a flexible tool to perform numerical simulation in continuum mechanics including fluid dynamics, solid mechanics and electromagnetics. OpenFOAM's modular structure allows end users to easily build new solvers and developers to add new features, constantly enlarging the range of possible applications of interest for both the academy and industry [3,4]. However, made an exception for a recent coordinated effort (www.exafoam.eu), relatively less attention has been paid by OpenFOAM developers to computational performances. In particular, aspects such as its parallel efficiency on massively parallel machines remain challenging [5]. Individual research groups have managed to resolve implementation bottlenecks and improve the performance of OpenFOAM, (see Bnà et al. [6] and references therein) but, often the lack of generality of their implementations has prevented their direct inclusion in the official OpenFOAM distributions.

Tailored optimization strategies are indeed required for any application to exploit the computational power of current and next-generation supercomputing systems (e.g. [7]). Modern HPC machines typically achieve their full computational power using heterogeneous hardware (e.g. CPU–GPU) and a huge number (up to millions) of computing units. Optimizing and reducing data movement, including memory access and I/O, is therefore a basic requirement to reach peak performances. The strong imbalance between the computational power and memory bandwidth still remains a major issue and it has not been attenuated by the latest technological trends. Performing arithmetic operations remains indeed several orders of magnitude faster than accessing data in memory, or performing communications between different computational nodes of a cluster machine. From this perspective, most of the real applications, including OpenFOAM, are and will become more and more memory-bound [8,9].

Changing algorithms can help to significantly improve the communication-to-computation ratio (i.e. the operational intensity) of an application but it may be a challenging and time consuming exercise, especially for complex general-purpose codes such as OpenFOAM. Reducing floating point precision arithmetic is instead often a simple but effective way of cutting down data movement and increasing the computational speed for most applications. On modern CPUs, performing operations in single precision (32-bit) is twice as fast as in double

* Corresponding author.
E-mail address: federico.brogi@ingv.it (F. Brogi).

https://doi.org/10.1016/j.future.2023.10.006
Received 23 October 2022; Received in revised form 3 October 2023; Accepted 6 October 2023
Available online 16 October 2023

precision (64-bit), since the amount of data moved in memory is halved and arithmetic operations are twice as fast as double precision [10]. When considering less conventional hardware such as NVIDIA GPU Tensor Cores the computational gain becomes even more attractive. These architectures in fact support half precision arithmetic with dedicated functional units (accelerators) in the hardware that makes low precision much faster than higher precision [11]. With this hardware, half precision arithmetic has a theoretical speedup of 16× instead of the expected 4×, with respect to double precision [12]. Moreover, the use of low-precision arithmetic may also be an effective way to reduce power and energy consumption [13], one of the main factors to be considered in using and designing current and next-generation high-performance computing machines. In reduced precision, energy saving is mainly due to the overall shortening of the execution time [13], which is in turn related to the combined effect of the lower cost per arithmetic operation and decreased memory communication. However, using low floating point precision is not always possible. Traditionally, CFD codes work with double precision since for complex fluid problems linear algebra solvers may not converge or not provide the solution with the required degree of accuracy. Several studies have proposed using mixed precision algorithms to unleash the power of multi-precision hardware without sacrificing accuracy or numerical stability (Abdelfattah et al. [8] for a review).

In OpenFOAM, a mixed precision feature has been released with version v1906. It is implemented in its complex framework following a rather simple idea: all the code is compiled in single precision except for the linear algebra solvers, which work in double precision. The reduced memory consumption alleviates the bandwidth bottleneck of OpenFOAM, thus increasing the computational speed of almost any application. However, to our knowledge, there are no systematic studies on the use of reduced precision computations in CFD applications in the literature, making an exception for more specific case studies such as the recent work on fluid the lattice Boltzmann method based fluid solvers [14]. Let us also note that in OpenFOAM, the fluid governing equations are discretized using the finite volume method [15] and the fluid solvers have in general low order of accuracy in space and time (below or up to the second order). From this perspective, therefore, it is interesting to understand whether a precision reduction (from single to double) significantly impacts the accuracy and stability of such low-order solvers.

In this work, we analyse and discuss the impact of floating point precision reduction (single and mixed with respect to double) on real CFD applications using OpenFOAM. In particular, we consider two important aspects such as the convergence and accuracy of computed solutions, the computational performance on both CPU and hybrid CPU–GPU hardware as well as the parallel efficiency of CFD applications. With the aid of theoretical and experimental analysis, we describe how precision reduction affects the individual parts of the applications that are commonly present in CFD solvers. To try to keep our results as much as possible of general interest, both incompressible and compressible solvers have been selected for testing, since they may represent the basis for any more complex solver to be built on (e.g. multiphase solvers). The quality of computed solutions (accuracy and convergence) of these solvers are tested considering important flow phenomena in laminar and turbulent flow regimes as well as the presence of shock wave discontinuities. Performance gain and change in the scaling behaviour of applications on parallel machines due to precision reduction are also considered. A theoretical model for the strong scaling of applications is also developed, which allows us to explain and better understand the changes observed with reduced floating point precision and different hardware specifics. Finally, we demonstrate how significant can be the effect of mixed precision on computational performance, with a speedup of 2.4×, when considering the hybrid CPU–GPU implementation of OpenFOAM that has been recently made available to users. Let us also note that, in this work, we do not directly discuss power and energy consumption aspects for which the interested reader is referred to one of the few dedicated works (e.g. [13]). However, any improvement in the computational performance in reduced precision may be related to a shortening of the computational time and hence, decreased energy consumption.

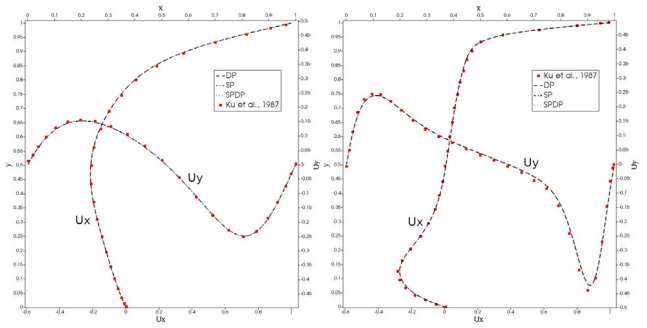
2. Impact of precision reduction on the quality of computed solutions

Most CFD solver algorithms, such as those implemented in OpenFOAM, can be divided into two main steps: the assembly of a matrix that results from the discretization (Finite Volumes, in OpenFOAM) of the governing partial differential equations, and the solution of the linear algebra system that brings to the numerical solution of the equations. When OpenFOAM is compiled in single/double precision means that all floating point numbers, hence for both the matrix assembly and the linear algebra solver, are in single/double precision. When using a mixed precision built of OpenFOAM, all floating point numbers are in single precision for the matrix assembly but are converted in double precision before solving the linear algebra. Here we analyse the impact of reduced floating-point precision (single and mixed) on two important aspects of CFD applications: the accuracy and convergence of computed solutions. Given the large number of CFD applications available in OpenFOAM, we limit our study to solvers for single-phase incompressible and compressible flows. It is well known in fact that these solvers face different computational challenges and represent the ground for almost any more complex solver (e.g. multiphase). Here, and in the rest of the paper, we use OpenFOAM v1912 for our tests, if not otherwise stated. The classical `icoFOAM` (incompressible) and `rhoPimpleFOAM` (compressible) solvers have been selected to solve radically different benchmarks: the laminar 3D lid-driven cavity, the decay of isotropic turbulence, the shock tube and the starting compressible jet. The lid driven cavity is a well known benchmark for incompressible solvers [16], especially when considering the laminar steady state regime. The isotropic turbulence benchmark [17] is also a standard test and it is well suited to evaluate the effect of reduced precision on numerical simulations with turbulence, here in the weakly compressible regime. The Sod shock tube test [18] is instead used to understand whether discontinuous solutions are affected by floating point precision representation. The fully compressible dynamic of a starting jet (high Mach and high Reynolds number) is used to test the behaviour of the numerical solver with reduced precision for transient compressible problems. Finally, as an example of a complex real-case application, we consider the simulation of a compressible, multiphase turbulent volcanic plume with the OpenFOAM-based solver `ASHEE` [19].

2.1. Lid-driven cavity

In this benchmark, an incompressible fluid is initially at rest in a cubic cavity. A tangential velocity is imposed on the top boundary and no-slip conditions (wall) are enforced on all the other boundaries. The flow regime in the cavity depends only on the dimensionless Reynolds number ($\mathrm{Re} = UL/\nu$, with U, the velocity of the moving lid, L the length of the cavity and ν the kinematic viscosity). The interested reader may refer to Shankar and Deshpande [16] for a complete review. Here we consider the laminar steady state regime for two different Reynolds numbers ($\mathrm{Re} = 100, 1000$). For this benchmark, we used the incompressible `icoFOAM` (v1912) solver compiled in single, mixed and double precision. Details of the simulation setup (boundary conditions and numerical schemes) can be found in Appendix E (Table 5).

All the simulations converge to physical solutions (Fig. 1) and regardless of the precision, they are consistent with a well-known reference study [20]. In particular, our results demonstrate that the single precision is sufficient for the fluid solver to provide accurate velocity profiles along the cavity centerlines (Fig. 1), with a mesh size of 100^3. We then also tested the order of accuracy, that is how the



(a) Re = 100. (b) Re = 1000.

Fig. 1. Lid-driven cavity benchmark solved with `icoFOAM` solver, using a mesh size of 100^3 cells, in single (SP), double (DP) and mixed precision (SPDP). Velocity profiles of U_x along y-axis centerline (left vertical axis) and U_y along x-axis centerline (top horizontal axis) are compared with those extracted from the figures of Ku et al. [20].

accuracy of the numerical solution varies with increasing mesh size, in both single and double precision for Re = 100. Our test shows that the order of accuracy is rather low, less than 1 for all variables (Appendix C, Fig. C.1, Table 4). However, the convergence is almost the same in double and single precision. In other words, it seems that precision reduction has no effect on grid convergence, at least for such a low-order fluid solver.

2.2. Sod shock tube

This test considers the dynamic that develops inside a 1D tube when the diaphragm, separating high-pressure and low-pressure regions of a gas, is removed. Typically, a rarefaction wave propagates toward the high-pressure region and a shock wave (a discontinuity on all fluid variables) forms and moves through the low-pressure region. Numerical schemes, that require a smooth solution, typically face accuracy and stability issues when dealing with these discontinuities. The `rhoPimpleFOAM` solver (v1912) compiled in single, double and mixed precision is here used to produce the numerical solution. Details of the simulation setup (boundary conditions and numerical schemes) can be found in Appendix E (Table 6). The initial conditions are reported in the caption of Fig. 2. The tube length is meshed with 1000 cells and the Courant number has to be set to Co = 0.01 using PISO (1 iteration outer loop to couple with energy equation) or to Co = 0.2 using PIMPLE (with 3 iterations in the outer loop) for the double precision built to be accurate. Therefore, PIMPLE is computationally more convenient, since allows us to perform simulations with larger time steps. As in the previous benchmark, despite the precision used, the solver converges to a physical solution that is in excellent agreement with the theoretical one [18]. The presence of contact and shock discontinuities are well resolved and important numerical artifacts (e.g. spurious oscillations) are not evident in the solutions with reduced precision. Moreover, as for the lid-driven cavity, the convergence of the numerical error with the mesh size is practically the same in single and double precision (Appendix C, Table 4, Fig. C.2). Setting larger values of the Courant number causes the numerical solution to become less accurate. Bigger values for the Courant number may also increase spurious numerical oscillations that affect not only the accuracy but also the stability of the fluid solver, especially in the presence of non-linear waves. Therefore, we performed dedicated tests to better understand the role that may play the maximum Courant number on numerical stability in different precision. In particular, we first tested the solver stability by performing a finite set of simulations with increasing maximum Courant number but, keeping the time step in each simulation fixed (using PIMPLE, Appendix D). Increasing the time step, hence the maximum Courant number, causes the numerical error, to increase rapidly, especially at

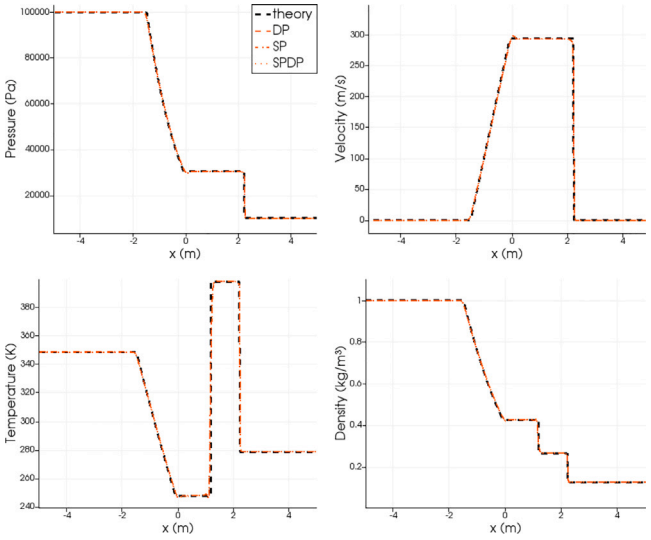


Fig. 2. Sod shock tube benchmark solved with `rhoPimpleFOAM` solver in single (SP), double (DP), and mixed precision (SPDP). Pressure, temperature, velocity, and density profiles (orange) are compared with the theoretical ones (black). At time 0, the interface dividing the high pressure (left, l) from the low pressure (right, r) zone is placed at 0 m. Initial conditions: $P_l = 0.1$ MPa, $P_r = 0.01$ MPa; $T_l = 348.432$ K, $T_r = 278.746$ K; $U_l = U_r = 0$ m/s. Heat capacity ratio $\gamma = 1.4$.

the beginning of the simulation, up to a point where the convergence of the computed solution is compromised. Indicatively, our tests show that the fluid solver is stable when the maximum Co reached in the first time steps is less than 3 (Fig. D.1). While it is unstable in a simulation that stops after one iteration with Co = 4, due to negative temperature values. Interestingly, this behaviour is observed in both double and single precision. Small differences between single and double precision can be noticed in the first few time steps by repeating the same test with a finer grid (Fig. D.2). However, in our test, these differences are not large enough to cause different behaviour in terms of stability. Therefore, the precision reduction does not seem to change significantly the stability range of the fluid solver. Moreover, regardless of the precision used, the most critical part for stability in this test case is at the beginning of the simulation (first few time steps), for which the numerical error is larger. In fact, when performing simulations keeping fixed the maximum CFL while adapting the time step, the fluid solver remains stable at all tested CFL (tested up to CFL = 20), if a sufficiently small starting time step is chosen.

2.3. Compressible decaying homogeneous and isotropic turbulence

Turbulence is one of the main non-linear complexities of fluid dynamics. For a comprehensive review of such an important topic, the interested reader may refer, among many other classical textbooks, to [21]. In brief, turbulent flows are characterized by a wide range of spatial and temporal scales, due to the presence of large eddies (defining the integral scale L) that continuously break into smaller and smaller eddies, until the dissipation scale is reached. This is the smallest scale and is called the Kolmogorov micro-scale η. The ratio between the integral and the dissipation scale grows with the Reynolds number $L/\eta \propto \mathrm{Re}^{\frac{3}{4}}$. A simulation that is able to capture all the scales (from the largest integral one, down to the smallest dissipation scale) is called Direct Numerical Simulation (DNS). When the flow is far from boundaries, the typical behaviour of turbulence is described by the energy cascade of the kinetic energy spectrum. When it is impossible to resolve all the relevant scales, one of the possibilities is to use the Large Eddy Simulations (LES) method, to take into account of the energy cascade in the sub-grid terms. A general description of the physical,

mathematical and numerical problem can be found in [17,21–30] and in references therein. Homogeneous and isotropic decaying turbulence is a classic benchmark used to test the capabilities of numerical codes to solve turbulence far from boundaries. Here, we are following the same procedure described by Cerminara et al. [19] and Cerminara [31], by using the ASHEE code in its single-phase and mixed-precision configuration. The objective is to compare the mixed-precision solution with the double-precision one, having already validated the latter with the eight-order scheme by Pirozzoli and Grasso [17]. Double precision is often needed in problems facing turbulence, to solve as accurately as possible the non-linear advection terms present in the Navier–Stokes equations. Indeed, these terms tend to grow with the spatial scale, becoming larger than the (mainly linear) dissipation terms. For this reason, double precision becomes even more important in LES, where the smallest scale is much larger than the dissipation scale and its contribution is taken into account using non-linear sub-grid scale turbulence viscosity terms [26].

The DNS test is performed using a mesh size of 256^3 cells in a cubic box with side $L = 2\pi$ and periodic boundary conditions. In this way, boundary effects can be neglected. The initial spectrum is the same described in Section 5.2 of Cerminara [31], so that the root-mean-square Mach number is $\mathrm{Ma}_{\mathrm{rms}} = 0.2$, the initial Taylor microscale is $\lambda = 0.5$, the eddy turnover time is $\tau_e = 3.66$, the Reynolds number based on the Taylor micro-scale is $\mathrm{Re}_\lambda = 116$, and the maximum wave-number is $k_{\max} = 127$. In this way, its product with the Kolmogorov micro-scale $\eta = 0.023$ is large enough to have a proper DNS. The same initial condition is mapped into a 32^3 mesh to perform a LES to be compared with the previous DNS results. LES are executed by using the Moin's model [19,31]. More details about the simulation setup (boundary conditions and numerical schemes) can be found in Appendix E (Table 8). The comparison has been performed by using both mixed and double precision. In single precision, the fluid solver does not necessarily converge. For instance, the LES with mesh size 64^3 is unstable (due to the divergence of linear algebra for the energy equation), but it is stable for mesh size 32^3. For the smaller mesh size (larger cells) numerical dissipation may be more effective in killing numerical noise that triggers the instability. Therefore, in general, the fluid solver in single precision is more sensitive to the numerical noise at the high frequency that is caused by the smallest turbulent scales that are near or smaller than the cell size. In Fig. 3, we report the spectrum of the kinetic energy for the DNS and LES simulations in double and mixed precision. Mixed precision works pretty well for this test case, with minor influence on the solution: the relative error with respect to the double precision case is $6.0 \cdot 10^{-7}$, and $7.2 \cdot 10^{-7}$, for the DNS and LES, respectively. 90% of the error is contained in the region $k < 10$, and $k < 5$, respectively for DNS and LES. By using a different LES sub-grid model, the effect of precision can have a larger impact. For example, by using the dynamic WALE model [19,31], the relative discrepancy increases to $1.4 \cdot 10^{-4}$. Finally, also for this test case, we analysed the convergence of the error with the increasing mesh size. The error for the kinetic energy spectrum is computed for each mesh size with respect to the results obtained with a large mesh (256^3). The order of accuracy, also for this test case, is quite low but very similar in mixed and double precision (Appendix C, Fig. C.3, Table 4).

2.4. Starting compressible square jet

A high-speed injection of a warm gas (330 K) into a static atmosphere with a slightly lower temperature (300 K) is considered. For simplicity, the jet fluid enters the atmospheric box through a square inlet (0.0635 m) resolved with a homogeneous structured mesh (no grid stretching). Given the low gas viscosity (1.8×10^{-5} Pa s) and high velocity (364 m/s) of the injected fluid, the jet flow is unsteady and compressible, characterized by high Mach number (>1 locally) and Reynolds number. Non-linear instabilities in the jet shear layer (such as Kelvin–Helmholtz instabilities), eddies, and pressure waves contribute to the formation of a turbulent buoyant plume in these conditions. As the jet enters the atmospheric box it forms a vortex and pushes the static air outwards, producing an intense pressure wave that propagates radially (Fig. 4). In our setup, the simulation is stopped when the first transient reaches the boundary of the computational domain and hence before the turbulent plume develops. The solver `rhoPimpleFOAM` for compressible laminar and turbulent flow is used for this test case. An LES approach combined with a one-equation eddy viscosity model [32] is used to allow the solver to deal with the unresolved scales of the turbulent flow. For testing, velocity and pressure probes are placed along the jet centerline, in the jet shear layer and far from the flow field in the atmosphere to record pressure waves. Details of the simulation setup (boundary conditions and numerical schemes) are reported in Appendix E (Table 7).

Single precision for this test case is not sufficient for linear algebra to converge. Therefore, no numerical solution is available with this precision. Mixed and double precision runs instead converge to physical solutions that result to be very similar. In particular, for a test case with a coarse mesh, the time series of pressure and velocity for all probes overlap almost completely (Fig. 5). When considering a test case with a refined mesh (double number of cells in the jet diameter), double precision and mixed precision start to be different. The differences are particularly evident in those parts of the recorded signals where high-frequency content is present on all fluid variables (e.g. Uy in Fig. 5b). The system sensitivity to tiny differences in the initial conditions may explain the changes observed in the numerical solutions with reduced precision and refined mesh. The impact of mixed precision on the accuracy of computed solutions can be evaluated by considering the statistical quantities of the fluid variables only (see Section 2.3).

2.5. Mixed precision in a real use-case: the simulation of a volcanic plume

Moving to a geophysical scale problem, we present a numerical simulation of the evolution of an explosive volcanic eruption in a still atmosphere. Volcanic plumes are characterized by high Reynolds and Mach numbers. The multiphase gas–particle mixture injected into the atmosphere by these kinds of fascinating and catastrophic events is typically very hot (above 1000 K) and denser than the surrounding air (above 3 kg/m^3). Shocks, turbulence and acoustic fluctuations start to develop immediately after the beginning of the eruption. The plume initially rises because of its initial momentum. Then, turbulent mixing decreases its density due to atmospheric air entrainment and expansion and the plume may reach a level where the buoyancy starts to be positive (buoyancy reversal). At this level, the column starts to behave as a proper plume, accelerating upwards due to its buoyancy. From the neutral buoyancy level, the updrafting mixture decelerates, to finally spread laterally into the umbrella cloud. More details on the phenomenon can be found e.g. in [31,33–35]. We now consider a test case to be solved with the OpenFOAM-based solver ASHEE [19]. The ASHEE (ASH Equilibrium Eulerian) model, based on the dynamic LES model and an asymptotic expansion strategy of the full non-equilibrium multiphase Eulerian model, solves gas-polydisperse particle turbulent flows that characterize volcanic plumes. Although this approach is valid for dilute concentrations (volume fraction smaller than 1%) of ash, larger particles can be also included in ASHEE using a Lagrangian approach with a two-way coupling regime. For testing, we consider that the fragmented magma is injected into a still atmosphere from a 500 m wide inlet, with an initial velocity, temperature, and gas mass fraction equal to 236 m/s, 1050 K, and 5 wt.%, respectively. This results in a mass flow rate equal to 2×10^8 kg/s, which is representative of a Plinian eruption. Atmospheric stratification is modelled by using the U.S. Standard Atmosphere. The computational domain is 50 km high and extends along the 2 horizontal directions for 100 km. It is discretized using 40 million cells with an orthogonal mesh with constant grading both in the vertical and horizontal directions. The temporal discretization is based on the second-order Crank–Nicolson scheme,

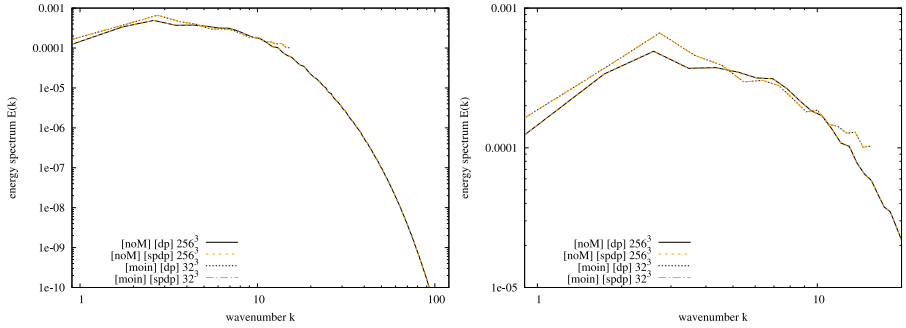


Fig. 3. Kinetic energy spectrum of the DNS (noM) and of the LES (moin) in double precision (dp) and mixed precision (spdp), after $t = 5.5\tau_c$. The right panel is just a zoom in the region resolved by the LES.

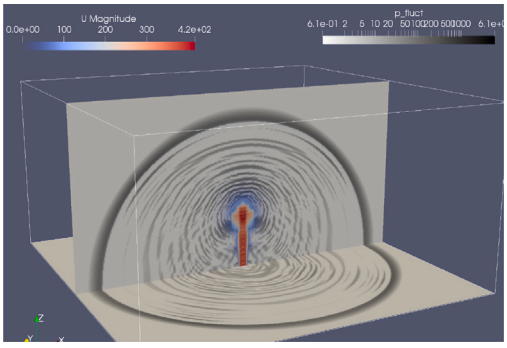
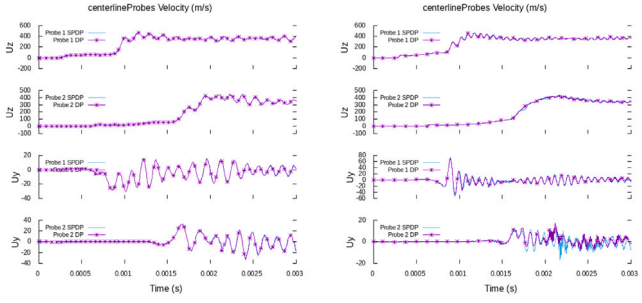


Fig. 4. Screenshot of LES simulation of the supersonic starting jet using `rhoPimpleFOAM` solver. The magnitude of the velocity field (in colour) and pressure fluctuations $\delta p = p - p_{ref}$ (black and white in log scale) are shown.

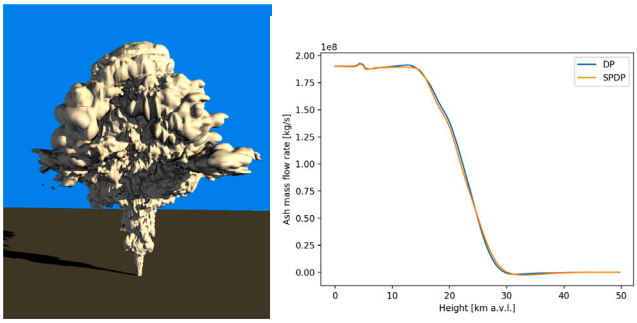


(a) (b)

Fig. 5. Times series of axial (Uz) and radial (Uy) components of the velocity field as recorded by two probes placed in the computational domain along the jet centerline for: (a) coarse mesh (16 384 000 cells); (b) refined mesh (131 072 000 cells).

with an adaptive time stepping based on the Courant number (Co $\leq$ 0.2). Numerical schemes and boundary conditions are described in detail in [19,31,34]. The bottom of the domain is treated as a thermally insulated slip wall, with a sink condition based on particle settling velocity to allow pyroclasts to deposit. The atmosphere is treated as an open boundary with input–output Dirichlet–Neumann conditions based on the velocity direction and a total pressure condition. The inlet has a prescribed velocity hyperbolic tangent profile to mimic conduit boundary effects. All these characteristics have been implemented in `ASHEE` by using the OpenFOAM infrastructure [1].

In Fig. 6a, we show a snapshot of the simulation 9 min after the onset of the eruption. The plume, described by an isosurface of ash concentration, reaches a maximum altitude of 30 km. The entire simulation covers 50 min of plume dynamics. In Fig. 6b, we show



(a) (b)

Fig. 6. Numerical simulations of a volcanic plume with OpenFOAM-based solver `ASHEE` [19]: (a) snapshot of the simulation 9 min after the onset of the eruption; (b) Comparison of vertically averaged profiles for the mass flow rate in double (DP) and mixed precision (SPDP).

vertical profiles of mass flow rate going upward inside the plume. They are obtained by averaging in time and horizontally in space over a 30 min time window. These profiles are used to compare the results in double and mixed precision. Mixed precision presents a good level of accuracy to reproduce the averaged properties of the simulated volcanic plume.

3. Performance gain of reduced precision computations

Most CFD solver algorithms, as those implemented in OpenFOAM, can be divided into two main steps: the assembly of a matrix that results from the discretisation (Finite Volumes, in OpenFOAM) of the governing partial differential equations, and the solution of the linear algebra system that brings to the numerical solution of the equations. The coupling between the different equations (continuity, momentum, and energy) can be then treated implicitly or explicitly. In the first case, all the coupled PDEs are written as a single linear system of equations that is solved at once with a linear algebra solver. In the second case, each equation is written as a single linearized system, that is solved separately (segregated strategy) and then the coupling is obtained using well-known iterative pressure- or density-based procedures (SIMPLE, PISO or PIMPLE; [15]). In particular, the PISO algorithm is used to couple the continuity and momentum equation for transient flow with a two-step iterative strategy. A predictor step, where an intermediate velocity is solved using pressure at the previous time step, is followed by a number of corrector steps that first solve for a pressure equation (derived combining continuity and momentum equations) and then correct the velocity accordingly until continuity is fulfilled. The coupling with the energy equation is also achieved with small time steps with PISO or with larger time steps using the PIMPLE algorithm (see Systems of equations from [36]). The segregated solution strategy is the

one adopted in OpenFOAM, although more recently implicitly coupled solvers have been developed (e.g. [37,38]). In this work, we have used only more classical OpenFOAM solvers that adopt a segregated solution strategy.

In the following, we will refer to "matrix assembly" as the set of all the operations necessary for the construction of the matrices. The other main portion of the code, which we will refer to as "linear algebra", includes the set of operations aimed at solving the linear system generated during the "matrix assembly" phase. Finally, the last portion of the code that we will mention is the "flux correction" phase. Passing from double to single precision in an ideal world means that all computational and communication costs involving floating point numbers are halved. Clearly, the parts of the solver involving integer numbers are not affected by the lower precision. When considering mixed precision, the computational gain is less obvious to be predicted. In this case, only the matrix assembly is performed in single precision whereas the linear algebra is still in double precision. The fraction of time spent in linear algebra may vary considerably for different applications (e.g. incompressible and compressible problems) as does the speedup. Moreover, the actual gain obtained with reduced precision depends on how much the application is memory bound and therefore on the details of the numerical setup (e.g. mesh size). Here we consider the lid-driven cavity and starting jet test cases to study and quantify the computational gain associated with reduced precision for incompressible and compressible CFD solvers (i.e. `icoFOAM` and `rhoPimpleFOAM`). The main reported metrics are the time spent for each part of the code in single and mixed precision, as compared to double precision.

A first analysis is conducted by changing the dimension of the computational domain and maintaining the number of cells per core constant (weak scaling analysis). This approach has been chosen to test the consistency of the observed gains while fixing the computational load and the memory bandwidth available per core. Subsequently, the test is repeated with an increased computational load to study its effect on the computational gain with reduced precision. The minimum amount of allocated resources considered is one socket (24 cores on CINECA's Marconi machine, see [39] for hardware specifics) to keep a constant availability of DRAM bandwidth per core.

The results are reported in Tables 1 and 2. The maximum gain provided by single precision compared with double precision is 2×. For the lid-driven cavity in single precision, the computational gain is near to 2× (≈ 1.9×) for the test case with a larger computational load per processor (166k cells/core). Instead, for the lower computational load (41k cells/core), the gain obtained is smaller. A deeper inspection reveals that this low value of the total gain is mainly related to the small gain of the linear algebra part, that in this case takes more than 90% of the total time. Hence, as recently well explained by Zounon et al. [40], the 2× speedup for linear algebra is only obtained for larger problems. In fact, the gain of linear algebra is affected by the performance of the preconditioner, the Sparse Matrix Vector Multiplication (SpMVM) kernel and by the weight of the MPI communications and the slack time due to global synchronizations and not perfect load balancing (in single precision the MPI time is reduced but not halved and it is more than 30% of the total time in the XL case). Regarding the SpMVM kernel, since it is memory bandwidth-bound, the speed-up in theory is given by the ratio of the time required for matrix storage in double precision with respect to the lower precision. For a symmetric ordered LDU-COO matrix (native format in OpenFOAM) the speed-up (S) is given by the following formula:

$$S = \frac{2 * \text{nFaces} * \text{size(int32)} + (\text{nCells} + \text{nFaces}) * \text{size(real64)}}{2 * \text{nFaces} * \text{size(int32)} + (\text{nCells} + \text{nFaces}) * \text{size(real32)}} = \frac{2 + \frac{\text{nCells}}{\text{nFaces}}}{\frac{1}{2}(3 + \frac{\text{nCells}}{\text{nFaces}})}, \quad (1)$$

where nCells and nFaces are the number of Cells and Faces respectively. For the Lid-driven cavity, M the ratio is approximately 1.4. Our tests

Table 1
Lid-driven cavity: performance gain in weak scaling tests using Single and Mixed precision vs. Double precision, with a mesh size of 10^6 cells (S), 8×10^6 cells (M) and 64×10^6 cells (XL).

Single precision vs Double precision							
Case	nCells/nCores	nCores	Assembly Phase	Solver Phase	Assembly Gain	Solver Gain	Total Gain
S	41.7K	24	4.94%	93.54%	1.88	1.26	1.29
M	41.7K	192	3.32%	95.67%	2.05	1.66	1.67
XL	41.7K	1536	1.76%	97.62%	2.02	1.63	1.63
M	166.7K	48	2.49%	96.83%	2.00	1.86	1.87
XL	166.7K	384	1.44%	98.10%	2.04	1.87	1.87
Mixed precision vs Double precision							
Case	nCells/nCores	nCores	Assembly Phase	Solver Phase	Assembly Gain	Solver Gain	Total Gain
S	41.7K	24	4.94%	93.54%	1.85	1.05	1.08
M	41.7K	192	3.32%	95.67%	2.17	1.15	1.17
XL	41.7K	1536	1.76%	97.62%	1.77	1.28	1.29
M	166.7K	48	2.49%	96.83%	1.97	1.13	1.14
XL	166.7K	384	1.44%	98.10%	1.93	1.20	1.21

have shown that the solver gain is above this theoretical value except for the S case.[1] This result is in line with the recent work of Zounon et al. [40] where they have studied the gain using single precision arithmetic in ILU and SpMVM kernels for a large set of sparse matrices. They found performances below 1.5× in the solution phase of the application of the preconditioner and 1.5× in the SpMVM kernel (using 10 cores).

As expected instead, for the mixed precision the gain obtained in matrix assembly and correction of the fluxes (single precision) is near two, while the gain in the resolution of the linear algebra (double precision) is near 1. Interestingly a small gain can also be observed in the solver phase for both the lid-driven cavity and the starting jet. Overall, the total gains (matrix assembly and solve) for the lid-driven cavity and starting jet are significantly different (1.14× versus 1.40× respectively, in the M test case with high computational load). The reason for this difference lies in the fact that the two applications have completely different time distributions between the matrix assembly and linear algebra phases (Fig. 7). For the lid-driven cavity, most of the time (>90%) is spent solving the linear algebra (mostly for the pressure equation), while for the starting jet the time is split roughly in half between the two phases. Hence, the gain obtained in the lid-driven cavity is mainly due to the small gain obtained in the solver phase. The total gain for the starting jet is mainly due to the halving of the time spent in the assembly phase. Furthermore, from the Tables 1 and 2, it can be noticed that the algebra solver phase percentage increases by increasing the number of cores used. This could be attributed to the increased weight of the communication as the number of cores increases. In particular, this increase might be primarily ascribed to the global MPI calls contained in the solver phase (i.e. `MPIAllReduce`). As expected, in mixed precision an increase of the time spent in linear algebra causes the total gain for both applications to decrease.

4. Parallel computations and precision reduction

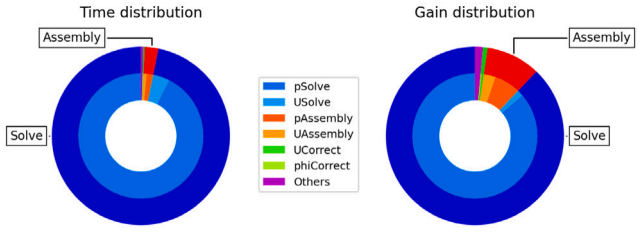
4.1. A theoretical model for strong scalability

We here discuss a relatively simple theoretical framework that can be used to describe and analyse the scaling behaviour of an application

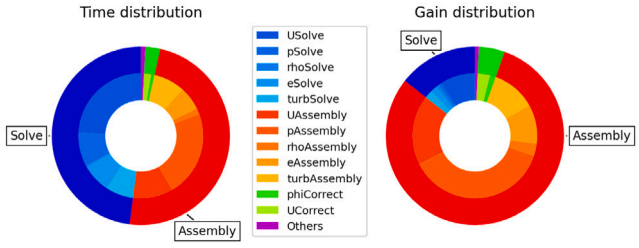
[1] For the lid-driven cavity, we used the Conjugate Gradient preconditioned by a diagonal-based Incomplete Cholesky for symmetric equations and the Bi-Conjugate Gradient preconditioned by an incomplete LU factorization for asymmetric equations.

Table 2
Starting jet: performance gain in weak scaling tests using Mixed precision vs. Double precision, with a mesh size of 2 048 000 cells (S), 16 384 000 cells (M), and 131 072 000 cells (XL).

Mixed precision vs Double precision							
Case	nCells/nCores	nCores	Assembly Phase	Solver Phase	Assembly Gain	Solver Gain	Total Gain
S	42.7K	48	53.05%	43.25%	2.14	1.30	1.66
M	42.7K	384	48.69%	47,60%	2.10	1.26	1.58
XL	42.7K	3072	34.40%	60.48%	1.86	1.15	1.34
M	170.7K	96	48.51%	47,99%	1.90	1.09	1.40
XL	170.7K	768	43.45%	52.80%	1.94	1.16	1.43



(a) *Lid-driven cavity - Dimension M - 48 Cores.*



(b) *Starting jet - Dimension M - 96 Cores.*

Fig. 7. CPU-time distribution and performance gain using Mixed precision in the case M of the lid-driven cavity and the compressible starting jet.

on parallel machines, including the effect of reduced precision and hardware specifics (e.g. memory bandwidth or processor). The main outcome here is a model for the strong scalability, that is, the time gain (or the speedup) obtained by increasing the number of processors for a fixed problem size.

Let's start considering a generic numerical problem that is solved using a variable number of processors P (MPI processes). On a first approximation, the total time (T) required by a single processor (p) to solve its own sub-problem is a function of: the time to transfer the data from the RAM memory of a process to another (intra-node and inter-node), the time to transfer the data from the RAM memory of a process to the registries, the time to process the data. Assuming that these three events occur in three non-overlapped time phases (e.g. Succi et al. [9]), the total time is then given by the sum of three terms:

$$T = \frac{F}{\dot{F}} + \frac{B}{\dot{B}} + \frac{C}{\dot{C}} \tag{2}$$

where F is the sum of floating point instructions done by the processor in FLOP (both in single and double precision), $\dot{F}$ the processor speed in FLOP/s, B the amount of data, in Bytes, to be transferred from the RAM to the registries, $\dot{B}$ the memory bandwidth in Byte/s, C the amount of data, in Byte, to be transferred from the RAM memory of a process to another, and $\dot{C}$ the network bandwidth, in Byte/s. Now, in order to understand the strong scaling behaviour of an application we need to understand how each term in Eq. (2) scales with the number of processors.

The first term ($F/\dot{F}$) represents the time required for pure calculations (no communication). Increasing the total number of cores simply reduces in general the computational work (F) assigned to the single processor. However, determining its exact behaviour is not an easy task. The actual load balancing and the ratio between the number of internal and boundary cells is a function of the algorithm used for domain decomposition. Moreover, the number of iterations for the preconditioned linear solver to converge can increase when switching from a serial to a parallel implementation. Therefore here we simply consider that a perfectly robust solver is used (i.e. the number of iterations does not change from the serial to the parallel case), and the number of internal cells is much higher than boundary cells and is the same among the processors (perfect load balancing). In this case, F can be modelled simply as inversely proportional to the total number of MPI processors (P):

$$F = \frac{k_F}{P} \tag{3}$$

where k_F is a constant expressing the total number of FLOP, in single and double precision, required to solve the entire numerical problem. By definition, the processor speed ($\dot{F}$) is independent of the number of processors and is expressed as the product of the average CPU frequency (f_{CPU} in Hz) and the Floating Point Instructions retired Per Cycle (k_{IPC})

$$\dot{F} = k_{IPC} f_{CPU} = k_{\dot{F}} \tag{4}$$

where k_{IPC} and f_{CPU}, hence also $k_{\dot{F}}$, are assumed to be constant.

For the same reasoning described for F, on a first approximation also B can be modelled as inversely proportional to the number of processors:

$$B = \frac{k_B}{P} \tag{5}$$

where the constant k_B represents the total amount of data for communications with memory for all processors. Regarding the memory bandwidth ($\dot{B}$), we need instead to distinguish the intra-node from the inter-node case.

Within a computing node, the RAM memory channels are shared among the cores and memory bandwidth contention cannot be avoided. In such a case, the memory bandwidth increases with the number of cores until saturation (Appendix Fig. A.1a). The behaviour is almost linear for a few cores and then starts to bend towards the asymptotic value. The memory bandwidth per core slowly decreases until saturation and then drops (Appendix Fig. A.1b). We modelled this behaviour with an expression for the inverse of the memory bandwidth equivalent to a first-order Taylor polynomial function of the number of processors (Eq. (6)).

When considering multi-node simulations, the RAM channels are still shared inside the node but not among the nodes. Therefore, in an inter-node strong scaling, we can assume that the memory bandwidth for each processor is constant and equal to the value for the single node (considering the full node as a reference and multiples of a single node to measure the speedup). However, the presence of cache levels (L1, L2 and L3) increases the value of $\dot{B}$ as the number of processors grows, due to an increase in the cache efficiency due to an increase in the total cache size available in the system [41]. The bigger the number of processors, the smaller the size of the data representing our discrete problem, until it fits into the cache. In this limit case, we get the maximum memory bandwidth of the system. We modelled this performance augmentation as a linear function (Eq. (7)). As a result, for the intra-node and inter-node scaling test we have two different equations expressing the memory bandwidth available for each processor:

$$\frac{1}{\dot{B}_{intra}} = \frac{1}{k_{\dot{B}}(P)} = k_1 + k_2(P-1) \tag{6}$$

$$\dot{B}_{inter} = k_{\dot{B}}(P) = k_3(1 + k_4(\frac{P}{n} - 1)) \tag{7}$$

where n is the number of cores in a node, $k_{\dot{B}}$ is the memory bandwidth and k_1, k_2, k_3 and k_4 are model constants. When $P = 1$ in the intra-node case and $P = n$ in the inter-node case, k_1 and k_3 coincides with the memory bandwidth of the serial case.

Finally, we need to consider the third term of Eq. (2), the time spent by the single process in sending and receiving data from other processes. As it is common in parallel CFD applications, OpenFOAM make use of MPI (e.g. Open MPI) for both point-to-point and collective communications. Asynchronous Point-to-Point communications are required for instance in the assembly phase or in a matrix–vector multiplication, while collective communications are used in `MPIAllreduce` functions required by linear algebra solvers. The total time spent in MPI communications depends on the number of cores in a strong scaling experiment. Typically, as the number of cores is sufficiently small, the wall-time fraction spent in the MPI library ($\frac{C}{\dot{C}}$ in Eq. (2)) is very small, almost negligible. Instead, MPI overhead starts to be significant for a large number of cores or, to be more precise and general, when the number of cells per core is small. For instance, in our simulations with OpenFOAM the MPI communications take less than the 5% of the total time if the number of cells per core is above around 150k. However, when the number of cells per core approaches around 20k, MPI communications become more and more important and may take up to 50% of the total time for many cores (Fig. B.1). However, even if the relative weight of MPI increases as the number of core increases, the absolute total time spent in MPI decreases. Indeed, in our test when the number of cells per core is sufficiently small the time spent in MPI communications becomes important and approaches an asymptotic value (Fig. B.1). As a result, we model the third term of Eq. (2) as:

$$\frac{C}{\dot{C}} \asymp k_C \tag{8}$$

where k_C is a constant. Let us enforce that this approximation cannot be used in a strong scaling for runs with a large number of cells per core. However, in this case the time spent in MPI communications is not significant with respect to the total time and as a first approximation can be simply neglected. A more accurate approach would require to model the exact behaviour of the dominant type of MPI calls (e.g. `MPIAllReduce`) as a function of the number of cores/cells per core as described in Appendix A. Here for practical reasons we simply express $\frac{C}{\dot{C}} \approx k_C w(P)$ where $w(P)$ is a weight function that varies from zero to one passing from runs with large to small number of cells per core.

Now, using Eqs. (3), (4) and (5) one can write the equation for the time required to solve the problem by a single processor as a function of the number of processors (P):

$$T_{intra}(P) = \frac{k_F}{P k_{\dot{F}}} + \frac{k_B}{P k_{\dot{B}}(P)} + w(P) k_C \tag{9}$$

where $k_{\dot{B}}(P) = 1/(k_1 + k_2(P-1))$ for the intra-node and $k_{\dot{B}}(P) = k_3(1+k_4(nP-n))$ for the internode. It is then straightforward, obtain the formula for the speedup, ($S(P) = T(1)/T(P)$) that is commonly measured in strong scaling experiments for both the intra and inter-node test cases:

$$S(P) = \frac{k_F/k_{\dot{F}} + k_B/k_{\dot{B}}(1) + w(1)k_C}{k_F/(P k_{\dot{F}}(P)) + k_B/(P k_{\dot{B}}(P)) + w(P)k_C} \tag{10}$$

The above equation is relatively general and, given that all the parameters are known, could be used to predict the scaling behaviour of an application, including the effect due to a change in floating point precision and hardware specifics (e.g. memory bandwidth). In fact, a precision reduction would change the value of most of the parameters appearing in the equation. However, a precise determination of all these constants is beyond the scope of the present work. Instead, in the next sections we will focus on showing how this model can be used to provide qualitative description of the speedup observed in strong scaling experiments of real CFD applications, such as those implemented in OpenFOAM.

4.2. Intra-node scalability

We now demonstrate and analyse the beneficial effects of using reduced precision on the scaling properties at the intra-node level. A scaling experiment with the starting jet test case compiled in double and mixed precision is considered. The size of the test case is relatively large to keep the number of cells per core above the threshold of 40k. In this way, memory communications are expected to play a significant role whereas the contribution from MPI communications should be negligible. The observed speedup deviates quite quickly from the ideal behaviour for both double and mixed precision (Fig. 8), clearly indicating an inefficient use of the computational resources. However, reduced precision helps to improve both the scalability and wall time of the application. In particular, the mixed precision scales better than double precision and provides a computational gain (up to 1.6×) that increases across the node.

To understand the reasons behind this behaviour, we now reconsider the scaling model described in the previous paragraph (Eq. (10)). Neglecting the effect of MPI communications ($w(P) = 0$) and applying the approximation for $k_{\dot{B}}$ described in Eq. (6), after some algebraic manipulation we obtain a simplified form of Eq. (10):

$$S(P) = \frac{1}{\frac{1}{P}(1 + a(P-1))} \tag{11}$$

where the constant a is defined as

$$a = (k_B k_2)/(k_F/k_{\dot{F}}(1) + k_B/k_{\dot{B}}(1)) \tag{12}$$

Eq. (11) is plotted in Fig. 8 for different values of a. It is clear that the lower the value of a the closer the curve is to the ideal speed-up. The overall effect of decreasing the parameter a therefore is similar to the effect of the precision reduction observed in our scaling tests (Fig. 8). In other words, one may argue that mixed precision scales better than double precision at the intra-node level thanks to a lower value of the parameter a. The decreased value of this parameter with reduced precision can be easily justified by the reduction of the amount of data to be transferred during matrix assembly (lower value of k_B) and the increase of the transfer rate of data due to a reduction of memory stalls (lower value of k_2), as also confirmed by the level 1 of the Top-Down Microarchitecture Analysis Method (TMAM) analysis (Appendix A, Fig. A.2). We instead assumed that the other parameters of Eq. (12) are not influenced by the type of precision: the number of instructions is the same, as a consequence of the same algorithm; the memory bandwidth and the CPU frequency are identical due to the same hardware; IPC is also the same since we assume that floating-point operations are implemented natively in the hardware for both single and double precisions (and vectorization is not present). Thus, the decrease of scalability observed at the intra-node level seems related to memory communication and hence it can be improved by using reduced precision.

4.3. Inter-node scalability analysis

Similarly to the previous section, we discuss the effect of precision reduction on the inter-node scalability by considering the starting jet case compiled in double and mixed precision (OpenFOAM version v2006). In the present scaling analysis, we consider the execution time of the application without pre-processing (e.g. mesh generation) and initialisation time delay at the beginning of the simulation.[2]

The results are reported in (Fig. 9). Both full and reduced precision runs display very similar speedup, with a clear superlinear behaviour

[2] The initialisation time is usually very small compared with the total execution time, especially for long production runs. However, for our setup, the execution time is also quite small and therefore it might become comparable with the initialisation time.

up to 10^4 cells/core (Fig. 9a,b). Below this threshold, MPI communications start to be more important, the scalability quickly deteriorates and double precision scales better than mixed precision. Nonetheless, it is important to notice that mixed precision is always faster than double precision. The computational gain in fact steadily increases from 1.4× up to 1.6× at 20^4 cells/core and then rapidly decreases to 1.2×, in the MPI-bounded region (Fig. 9c). Hence, as expected, mixed precision does not directly improve the parallel efficiency of the application as in the intra-node case, but it is still computationally convenient up to the number of cells/core where the application present a good parallel efficiency.

For the inter-node case, the MPI overhead cannot be neglected and has to be considered. For simplicity here, in the third term of Eq. (9), we set $w(P) = 1$ for multiple node runs and $w(1) = 0$ for the reference case (single node). As a result, Eq. (10) can be written as

$$S(P) = \frac{k_F/k_{\dot{F}}(1) + k_B/k_{\dot{B}}(1)}{k_F/(Pk_{\dot{F}}(P)) + k_B/(Pk_{\dot{B}}(P)) + k_C}. \tag{13}$$

Then, assuming $k_{\dot{F}}$ to be constant and using Eq. (7), after some algebraic manipulation we get

$$S(P) = \frac{1}{\frac{b_1}{P} + \frac{b_2}{P(1+k_4(\frac{P}{n}-1))} + b_3} \tag{14}$$

where the following definitions hold

$$b_1 = \frac{1}{1 + \frac{k_B k_{\dot{F}}}{k_{\dot{B}}(1) k_F}}, \tag{15}$$

$$b_2 = \frac{1}{1 + \frac{k_F k_{\dot{B}}(1)}{k_{\dot{F}} k_B}}, \tag{16}$$

$$b_3 = \frac{k_C}{k_F/k_{\dot{F}(1)} + k_B/k_{\dot{B}}(1)}. \tag{17}$$

Eq. (14) is plotted in Fig. 9d. A good agreement between the model and the curve plotted in Fig. 9b has been found for the Marconi cluster using the parameters $b_1 = 0.3$, $b_2 = 0.5$, $k_4 = 0.003$, $b_3 = 0.00032$ for the mixed precision and $b_1 = 0.22$, $b_2 = 0.85$, $k_4 = 0.002$, $b_3 = 0.00025$ for the double precision. For the same reasons discussed for the intra-node case, the constant b_3 is higher in mixed precision with respect to double precision. Regarding the variation of the parameters b_1 and b_2, it depends on the ratio $\frac{k_B k_{\dot{F}}}{k_{\dot{B}} k_F}$. If the ratio $\frac{k_B}{k_{\dot{B}}}$ decreases more than $\frac{k_F}{k_{\dot{F}}}$, the variation of b_2 is negative and of b_1 is positive, and viceversa. Since we can reasonably assume that the variation of $\frac{k_F}{k_{\dot{F}}}$ is less pronounced than $\frac{k_B}{k_{\dot{B}}}$ (in the previous paragraph we assumed no variation for the term $\frac{k_F}{k_{\dot{F}}}$), b_1 and b_2 are higher and lower in the mixed precision case, respectively. Regarding the parameter k_4, we can expect a higher value of this parameter in mixed precision due to a stronger cache effect when working with floats.

In conclusion, the two phenomena that influence the scaling plot are super-linearity and MPI communication. The super-linearity is due to the presence of levels of cache that, as the number of nodes grows, increasingly reduces the memory bandwidth bottleneck and hence provides an additional speedup. The super-linearity is slightly more pronounced in mixed precision at a low number of cores. When the MPI communication starts to play a significant role, the mixed precision scales worst. The reason is that the MPI time is not affected by the mixed precision since most MPI calls are from the linear algebra solvers which is double precision. Moreover, since the MPI time is related to synchronization lags instead of the amount of data to be exchanged by processors, the same conclusion may be valid also if single precision is used everywhere in the code.

5. The role of mixed precision in a hybrid CPU–GPU implementation

In the last decade, a number of GPU implementations of OpenFOAM have been attempted by individual research groups with different strategies and different results in terms of computational performance (e.g. [42–44]). However, none of these attempts have been included yet in the official OpenFOAM distributions.

More recently, Bnà et al. [6] and Zampini et al. [45] have developed an interface library named PETSc4FOAM that has been included in the OpenFOAM official release as a module. PETSc4FOAM extends the list of available linear solvers with the ones embedded in PETSc and other packages (e.g. HYPRE and ML), with and without GPU support. It basically works with any OpenFOAM solver and allows the user to take advantage of GPU hardware without requiring any change to the source code. Certainly, only the solution of the linear system can be offloaded to the GPU, the assembly of the matrix is still performed on CPU using the original OpenFOAM routines. Therefore this hybrid CPU–GPU

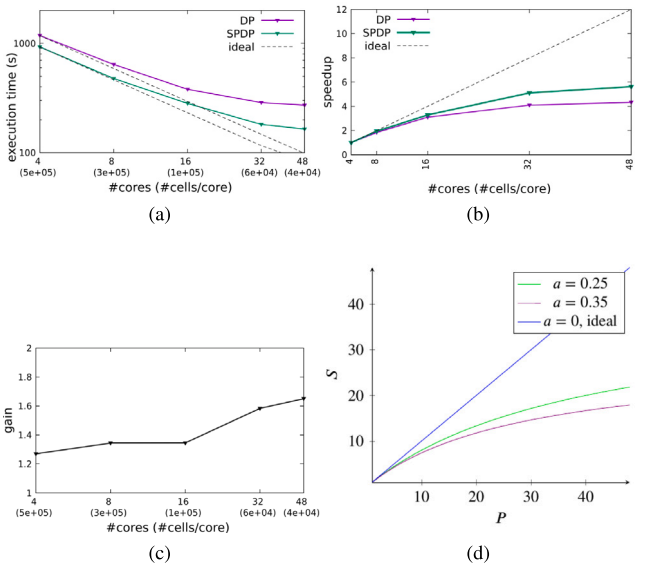


Fig. 8. Intra-node strong scaling for the starting jet using a mesh size of $\approx 2 \times 10^6$ cells performed on Marconi machine (a,b,c). The speedup obtained with the theoretical model (d), is also reported for a qualitative comparison with the experimental one (b).

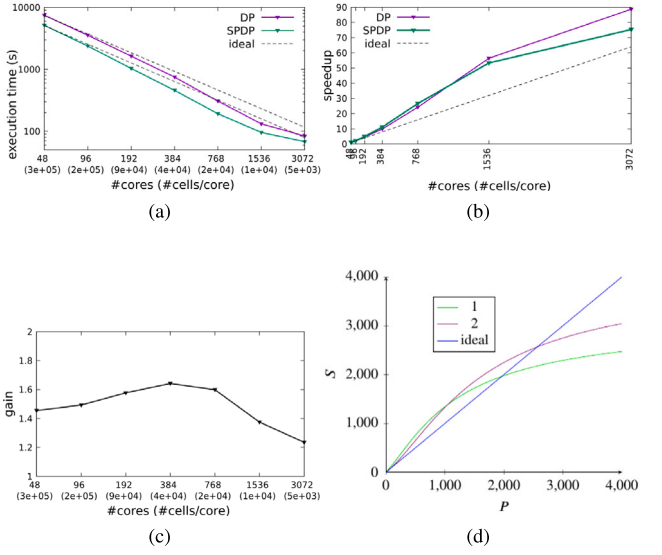


Fig. 9. Inter-node strong scaling for the starting jet using a mesh size of $\approx 16 \times 10^6$ cells performed on Marconi (a,b,c). The speedup obtained with the theoretical model (d), is also reported for a qualitative comparison with the experimental one (b). The inter-node speed-up from the theoretical model is computed with $b_1 = 0.3$, $b_2 = 0.5$, $k_4 = 0.003$, $b_3 = 0.00032$ (line 1, green) and $b_1 = 0.22$, $b_2 = 0.85$, $k_4 = 0.002$, $b_3 = 0.00025$ (line 2, violet).

strategy provide a speedup that primarily depends on the fraction of time spent in linear algebra by a specific application. For instance, roughly 90% of the computational work ("solve" in Fig. 7a) can be accelerated on GPUs for the Lid-driven cavity solved with `icoFoam`. This fraction reduces to 50% (Fig. 7b) when considering the starting jet test case with `rhoPimpleFoam`. Moreover, one has also to consider the time spent in CPU–GPU communication that may have a significant cost. OpenFOAM applications use a segregated solution strategy, that means that each equation is solved separately and then coupled through iterative procedures. The time spent in periodic transfer of the LDU matrix values to the GPU, every time a linear system has to be solved, may therefore increase dramatically with the number of equations to be solved. Nevertheless, the use of mixed precision may come in to rescue this hybrid CPU–GPU strategy. Indeed, with the mixed precision, the amount of data involved in CPU–GPU communication is halved and matrix assembly on CPU becomes twice as fast. Moreover, if the matrix sparsity pattern does not change from one iteration to the next one, it can be cached on GPU and only the matrix values have to be offloaded (in single precision, in the SPDP case).

If we neglect the time spent for the data transfer and the conversion,[3] we can approximately write the total time as the sum of the time spent in the solution of the linear system and the time spent in the assembly of the linear operators. The default in OpenFOAM is to use double precision for all the computations and a solver based on CPU. In the formula we have

$$T_{tot} = T_{solver\_CPU} + T_{assembly\_DP} \tag{18}$$

The fraction of time spent in the assembly is defined as

$$F = \frac{T_{assembly\_DP}}{T_{tot}} \tag{19}$$

As we already said, the hybrid CPU–GPU approach has no impact on the assembly loop in terms of computational time but it can be easily combined with mixed precision. On the contrary, mixed precision has no impact on the solution of the linear system but strongly influences the execution time of the matrix assembly part. Therefore, by defining the speedup obtained with the use of GPU linear algebra solvers and mixed precision as:

$$S_{GPU} = \frac{T_{solver\_CPU}}{T_{solver\_GPU}} \qquad S_P = \frac{T_{assembly\_DP}}{T_{assembly\_SPDP}} \tag{20}$$

one can apply Amdahl's law and predict the total speedup of mixed precision and GPU linear algebra as:

$$S = \frac{T_{solver\_CPU} + T_{assembly\_DP}}{T_{solver\_GPU} + T_{assembly\_SPDP}} = \frac{1}{\frac{1}{S_{GPU}}(1-F) + \frac{1}{S_P}F} \tag{21}$$

In Table 3 we report the total Execution Time and the total Gain of the starting jet use-case using a different combination of precision and number of equations offloaded to the GPUs. In particular, here we use a recent extension of PETSc4FOAM [46] that provides the possibility to use NVIDIA algebra solvers with GPU support of the AMGx library [47]. The best gain is achieved using the mixed precision and the solution of all equations on GPU. The gain can also be estimated using formula (21). In Section 3 we showed that a reasonable value for F and S_P is 0.5 and 2, respectively. Using these values, the estimate of the total gain achieved using only the mixed precision is 1.333 (the same value in Table 3 is 1.31). If the linear equations are solved on GPU, a reasonable value for S_{GPU} taken from our experiments is 3, which gives us an estimate of the total gain equal to 2.4 (compared to 2.43 in Table 3).

[3] The matrix has to be converted from the native LDU-COO format to CSR each time the solver is called. This operation is performed efficiently on GPU using the radix sort algorithm.

6. Discussions and conclusions

We have analysed the effect of reduced precision on different test cases considering incompressible, compressible as well as multiphase flow problems. Overall, single precision provides numerical solutions that are in good agreement with double precision or the theoretical ones for test cases where the flow is laminar (the lid driven cavity and the shock tube benchmark). Moreover, the low order of accuracy of the tested OpenFOAM solvers seems not to be affected significantly by the precision reduction. Single precision therefore may be expected to be sufficiently accurate for most laminar fluid problems except for those cases where the initial conditions require double precision or the solution is required with a particular degree of accuracy. When considering turbulent flows, single precision seems not to be an option instead. In this case, the nonlinear terms in the governing equations amplify the numerical noise, and in some cases (e.g. depending on the grid size) compromise the convergence of linear algebra solvers. In particular, `rhoPimpleFoam` and `ASHEE` solvers have not converged to physical solutions when considering the dynamics of decaying isotropic turbulence, starting compressible jet and turbulent multiphase flow of a volcanic plume. However, for these cases, one may rely on the implementation of a mixed precision concept in which linear algebra solvers work in double precision. In this case, all the aforementioned computations converge to consistent physical solutions. The spectrum of the kinetic energy of decaying isotropic turbulence in mixed precision results to be accurate with respect to double precision (error $\approx 10^{-}7$) for both DNS and LES simulations. As well, in complex volcanic plume simulations, the average properties of flow are well recovered with mixed precision.

In general, the computational gain one may obtain in reduced precision varies significantly depending on the characteristics of the CFD application and the computational load per core of the test case. The latter, in particular, is directly related to how much the application is bounded by memory or MPI related communications. Test with the lid-driven cavity in single precision shows that a speedup near to the expected 2× is observed only for the test cases with a larger computational load (166k cells/core). Smaller test cases (40k cells/core) present much smaller gains (1.1–3×), due to the smaller speedup of the linear algebra part of the fluid solver. Changing the application may also affect the computational gain in reduced precision, since the fraction of time spent on different parts of the fluid solver may change significantly. For instance, with respect to the lid-driven cavity where the linear algebra takes more than >90% of the total time, the starting jet requires only half of the total time. This aspect becomes even more important when considering the computational gain one may obtain with OpenFOAM in mixed precision, for which only the matrix assembly is done in single precision and linear algebra in double precision. The lid-driven cavity solved with `icoFoam` presents indeed in general a much smaller maximum gain (1.3×) than the starting jet with `rhoPimpleFoam` (1.7×) or volcanic plume simulations with ASHEE. Despite all this variability, in reduced precision (single or mixed) the matrix assembly is observed to be significantly faster for all test cases (1.7–2.2×). As well, in mixed precision linear algebra is also a bit faster (1.1–1.3×).

Reduced precision has also an impact on scalability. This aspect has been studied here interpreting the results of the scaling tests with the aid of an ad hoc developed theoretical model. Despite the strong assumptions, the model qualitatively reproduces the observed scaling behaviours when passing from full to reduced precision and allows us to understand and generalise the test outcomes. At the intra-node level, the use of reduced precision improves the scalability, because the latter is affected by memory communications, as also confirmed by the modelling results and TMAM analysis. This aspect seems to be relevant for OpenFOAM (v1912) which is characterised by a low intra-node scalability performance. Moreover, the computational gain of reduced precision increases with the number of cores since the

Table 3
Execution time and gain of the starting jet benchmark using different types of algebra solver and a single node of Marconi100 [48].

Mixed vs. Double precision for CPU–GPU OpenFOAM			
Solver	Precision	Execution time	Gain
PBiCG.DILU	DP	1744	–
PBiCG.DILU	SPDP	1329	1.31×
PBiCGStab.JACOBI_L1 Equation (p) on GPU	SPDP	1237	1.41×
PBiCGStab.JACOBI_L1 Equation (p|U) on GPU	SPDP	931	1.87×
PBiCGStab.JACOBI_L1 Equation (p|U|e) on GPU	SPDP	833	2.09×
PBiCGStab.JACOBI_L1 All equations on GPU	SPDP	733	2.38×
PBiCGStab.BLOCK_JACOBI All equations on GPU	SPDP	718	2.43×

memory bandwidth per core decreases across the node. At the inter-node level, the scalability is more complex since both memory and MPI communications need to be considered. Supported by our scalability model results, we have shown that memory communications are mainly responsible for the superlinear behaviour observed for both double and mixed precision up to $\approx 10^4$ cells/core. In this memory bound region, the application is indeed scaling better than in the ideal case since, as the number of nodes grows, the total cache available in the system increases, thus reducing the memory bandwidth bottleneck. The overall effect is that the application receives an additional performance boost and hence an additional contribution to the speedup. In the memory bound region of the speedup plot, mixed and double precision present similar scalability, with mixed precision performing slightly better. It is important also to note that the degree of super-linearity in a speedup plot is related to how much the reference case (in our case the one run on a single node) is bounded by memory bandwidth. Indeed, when the reference case is not strongly affected by the bandwidth bottleneck (in our case, on multiple nodes) the super-linear behaviour is not observed at all. When MPI communications start to play a role (i.e., when the number of cells/core becomes too small), the parallel efficiency decreases quite rapidly as more computational nodes are used and the full precision even scales better than reduced precision. Nevertheless, the computational gain in reduced precision with respect to the full precision is significant, up to the number of cells/core for which the application presents good parallel scalability. Although the conclusions regarding the scalability are based on the comparison between double and mixed precision, we believe they can apply also for the single precision case.

Finally, reduced precision computation is also desirable when working with GPUs, given that efficient data communication is needed to properly exploit the potential of such hardware. In the near future, all computationally intensive applications will be required to use GPUs, since they are becoming an increasingly common and fundamental part of modern heterogeneous cluster architectures. In this work, we have demonstrated the power of using mixed precision for a hybrid CPU–GPU OpenFOAM implementation, which, at the moment, is the only option available to the end user (from the up-to-date official OpenFOAM repository). In the hybrid CPU–GPU approach, only the linear algebra part is offloaded on the GPU. Therefore, the total speedup achievable is primarily a function of the fraction of time spent by the application on the linear solver. The latter is application dependent and, in our tests, it may vary from 90% for the lid-driven cavity to 50% with the compressible starting jet. Neglecting CPU–GPU communication cost and applying Amdahl's Law, it is easy to predict the total speedup for an application. When considering the starting jet (i.e., the worst-case scenario in terms of the time spent in the linear algebra solvers), the combination of mixed precision with the hybrid CPU–GPU implementation may theoretically provide up to 2.4× speedup (w.r.t double precision on pure CPU architecture). This value in fact matches the one measured in our single node tests. Although preliminary, this result may represent a first indication for the end user who wants to speed up CFD simulations with hybrid CPU–GPU and mixed precision implementation. Multi-node GPU scaling tests are left for future work. As well as more work remains to be done to test the use of mixed precision linear algebra solvers with GPUs or tensor cores that can even work in half precision [11]. Interestingly, these solvers may further speed up the resolution of linear algebra without compromising the convergence or accuracy of computed solutions. Finally, in this study, we did not consider IO and power consumption, two important aspects that we believe deserve dedicated works. Let us also recall that all the fluid solvers used in this work use a segregated solution strategy (explicit equation coupling), which is the one traditionally adopted in OpenFOAM. Only, more recently, a few solvers with implicit coupling have been developed [37,38]. Depending on the flow problem type, the coupling strategy (explicit vs. implicit) may play a role in determining the accuracy, stability, and computational performances of a fluid solver [37], and therefore may also change the impact of a precision reduction. However, the evaluation of this aspect is left for future investigations.

CRediT authorship contribution statement

F. Brogi: Conceptualization, Methodology, Simulations, Scaling analysis, Theoretical model, Software, Writing – original draft, Writing – review & editing. **S. Bnà:** Methodology, Simulations, Scaling analysis, Theoretical model, Software, Writing – original draft, Writing – review & editing. **G. Boga:** Methodology, Simulations, Theoretical model, Software, Writing – original draft, Writing – review & editing. **G. Amati:** Methodology, Simulations, Scaling analysis, Writing – review & editing. **T. Esposti Ongaro:** Writing – review & editing, Funding acquisition. **M. Cerminara:** Methodology, Simulations, Software, Writing – original draft, Writing – review & editing.

Declaration of competing interest

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

Data availability

Data will be made available on request.

Acknowledgements and financial support

This work has been supported by Istituto Nazionale di Geofisica e Vulcanologia (INGV) and by SuperComputing Application and Innovation Department (CINECA). This research has received funding from European Union's Horizon 2020 research and innovation programme under the ChEESE project, grant agreement no 823844, from the exaFOAM project, grant agreement no 956416, and from INGV Pianeta Dinamico grant CHOPIN.

Appendix

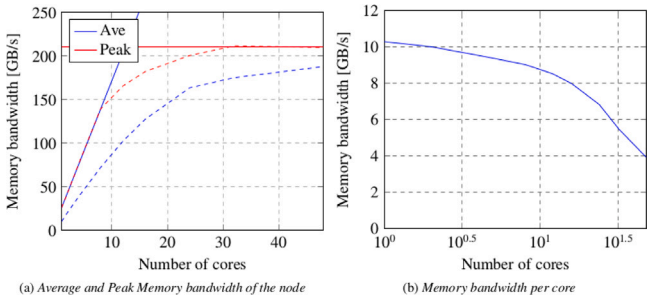


Fig. A.1. Average and Peak Memory bandwidth registered in the MARCONI cluster using the 3D Lid-driven cavity benchmark (case M).

Appendix A. Top-Down Microarchitecture Analysis Method (TMAM) of an OpenFOAM application

We performed Top-Down Microarchitecture Analysis Method (TMAM) of the lid-driven cavity benchmark varying the number of cores (aka strong scaling). According to the TMAM analysis [49], a slot in a CPU pipeline can be found in a binary state, stalled or not. Our analysis has found that Retiring and Back End bound are the two most frequent states in the first level of the TMAM. Bad speculation and Front End Bound can be neglected. At the second level, Memory Bound is the dominant state and Core Bound can be neglected. Fig. A.2 shows the slot fraction of Retiring and Memory Bound for the M-version of the lid-driven cavity run in double and mixed precision. Both intra-node and inter-node tests have shown that the slot fraction of Retiring is shifted higher in the mixed precision than the double precision. As a consequence, the complementary slot fraction of Memory Bound is shifted lower. In other words, the mixed precision increases the efficiency of the simulation but does not influence the trend with the variation of the number of cores. This trend is characterized by an increasing of the slot fraction of the memory stalls in the intra-node case due to the memory bandwidth contention. In the inter-node case, due to the increasing of the RAM and cache size of the system and the decreasing of the problem size per node, the efficiency increases until saturation, see Fig. A.2.

Appendix B. Modelling MPI communication in strong scaling experiment

Our profiler analysis has shown that in double precision the most expensive and dominant MPI function is `MPI_Allreduce` (see Fig. B.1). The same profiling analysis repeated with the code compiled in mixed precision provides evidence that `MPI_Recv` is the most used. However, we think that this result is only due to the way the mixed precision code is implemented.

In [50] the authors have shown that the behaviour of `MPI_Allreduce` is asymptotically logarithmic. However, if the number of cores is relatively low ($P < P*$, where $P*$ is in the order of thousands), the time is constant. This behaviour has not been registered by our tests since they include the slack time due to load unbalance; this term is not negligible, as confirmed by our tests, but decreases of importance as the number of cells per core decreases (see Fig. B.1).

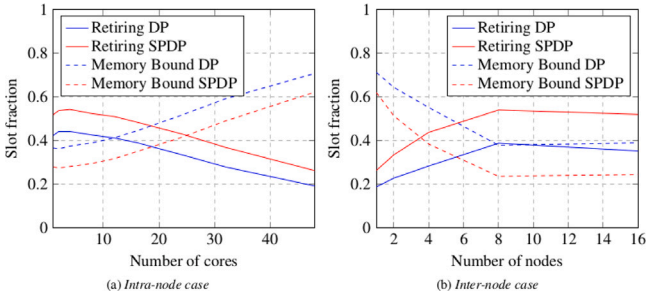


Fig. A.2. Retiring and Memory Bound slot fraction in the TMAM analysis of the 3D lid-driven cavity benchmark (case M) using double and mixed precision.

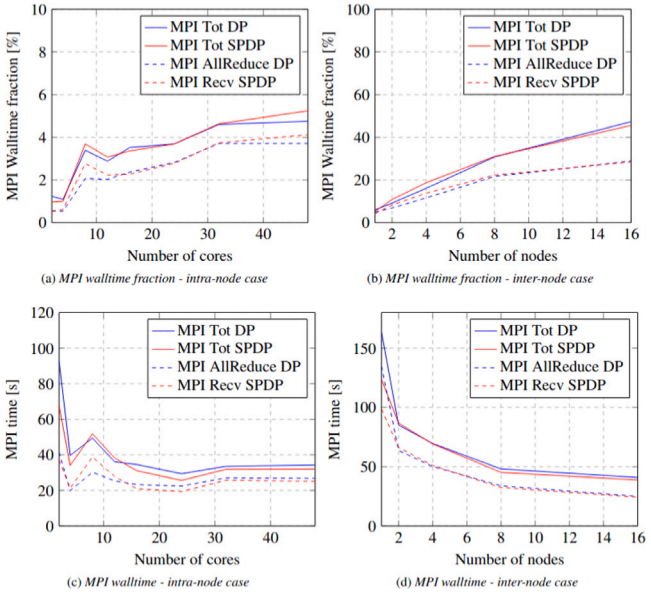


Fig. B.1. MPI walltime fraction and absolute walltime in the strong scaling test (intra-node and inter-node) of the 3D lid-driven cavity benchmark (case M) using double and mixed precision.

Appendix C. Convergence analysis and order of accuracy

Here, we report the order of accuracy (Table 4), that is the convergence of the error with respect to the mesh size, for three different test cases: the lid-driven cavity (Fig. C.1), the shock tube (Fig. C.2) and isotropic turbulence (Fig. C.3). The error is computed as an L2-norm:

$$E_{L2} = \sqrt{\frac{\sum_i^{Ncells}(V_i - V_i^{ref})^2}{\sum_i^{Ncells}(V_i^{ref})^2}} \tag{22}$$

where V_i is a generic fluid variable (density, velocity, pressure or temperature) at ith cell. As a reference solution (V^{ref}), we used the theoretical solution for the shock tube, and the solution with larger mesh size for the lid-driven cavity (8^6 cells) and isotropic turbulence (256^3 cells).

Appendix D. Stability of computed solution and maximum Courant number

Here, we report the test for the maximum Courant number with the shock tube benchmark in single and double precision for two different grids (Figs. D.1, D.2). In particular, here we tested the solver stability by performing a finite set of simulations with increasing the maximum

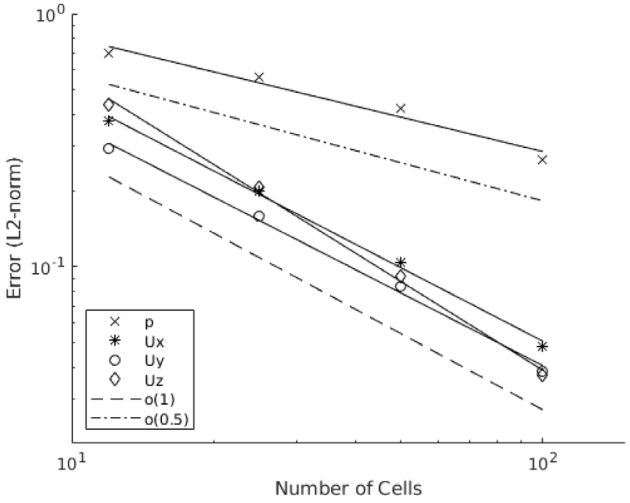


Fig. C.1. Convergence test for the lid-driven cavity (Re = 100) in double precision. In single precision, the figure looks exactly the same since the error values are very similar. Differences in the order of accuracy are reported in Table 4.

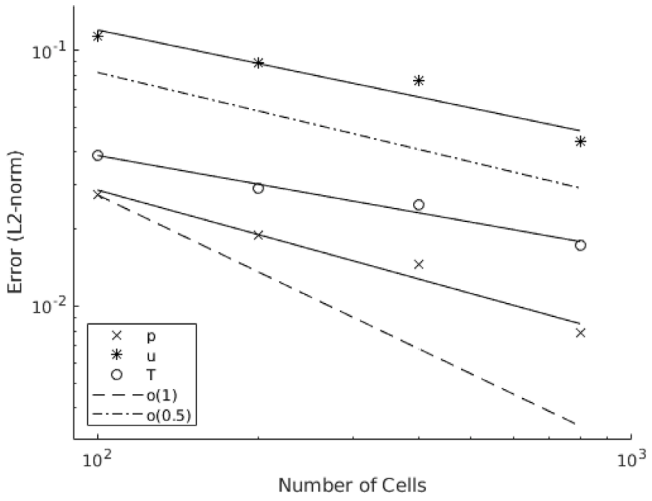


Fig. C.2. Convergence test for the shock tube in double precision. In single precision, the figure looks exactly the same since the error values are very similar. Differences in the order of accuracy are reported in Table 4.

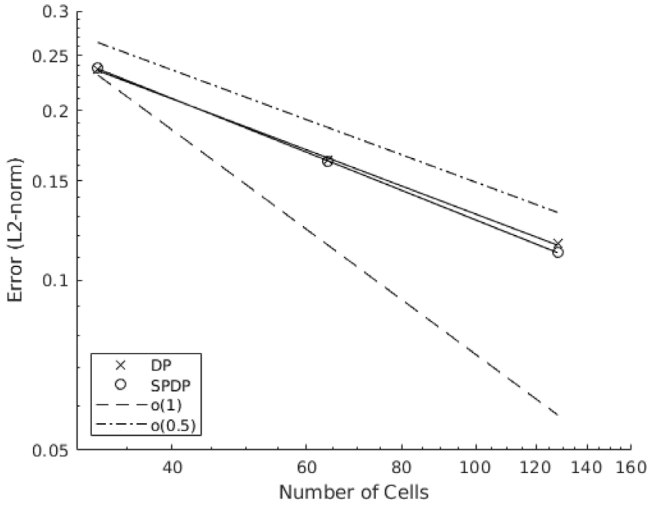


Fig. C.3. Convergence test for the isotropic turbulence test in double and single precision. The order of accuracy is reported in Table 4.

Table 4
Order of accuracy of the fluid solvers used in this study for the lid-driven cavity (`icoFoam`), shock tube (`rhoPimpleFoam`), isotropic turbulence (`ASHEE`).

Cavity	Ux	Uy	Uz	p
DP	0.9639	0.95444	1.1649	0.4507
SP	0.96393	0.95439	1.1648	0.45057

Shock tube	U	p	T
DP	0.43603	0.57843	0.37251
SP	0.43533	0.57767	0.37228

Iso. Turbulence	Kinetic energy spectrum
DP	0.51334
SP	0.54116

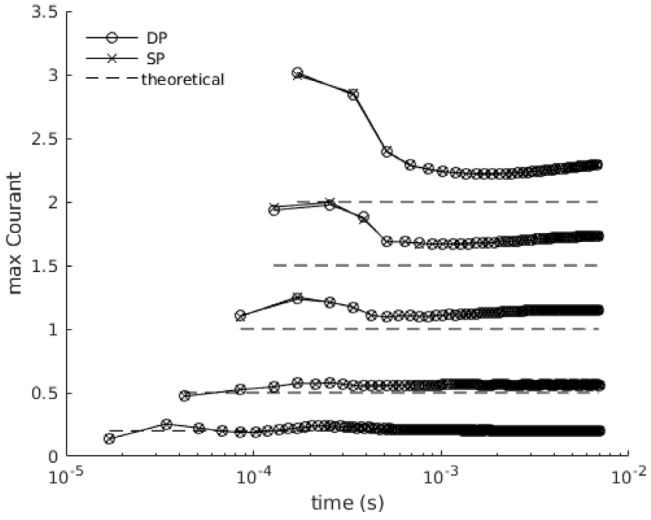


Fig. D.1. Evolution of the maximum Courant number in the shock tube simulations with 400 cells, in single and double precision, with five different constant time steps. For each simulation, the theoretical Courant number (dashed line), computed using the theoretical maximum velocity, is also reported. The time axis is set with a logarithmic scale to better visualize the beginning of the simulation (first few time steps), that is when the numerical error is larger, especially for high Courant number simulations.

Courant number but keeping the time step in each simulation fixed (using PIMPLE). In this way, our test results are more general and not affected by the particular way the time step is adapted during the simulation, to keep the CFL number below a predefined threshold (which in turn depends on the time step used to start the simulation and the algorithm used to keep the CFL number below the defined threshold). Given that the analytical solution for the shock tube has a constant maximum velocity, fixing the time step, in theory, implies that the maximum Courant number is also fixed. However, in the simulations, the maximum Courant number deviates significantly from the expected one due to the numerical error in the computed velocity in a similar fashion for both single and double precision. In particular at the very beginning of the simulations (few time steps), the maximum Courant is much larger than the theoretical one. This is most probably due to a larger numerical error related to the initial conditions to which the solver has to adapt.

Appendix E. Test cases numerical setup

Here, we report boundary conditions and details of the numerical setup for the lid-driven cavity (Table 5), shock tube (Table 6), compressible decaying isotropic turbulence (Table 6) and starting compressible jet (Table 8). Please refer to the OpenFOAM user guide for the exact definition and more detailed description of boundary conditions and numerical schemes [1].

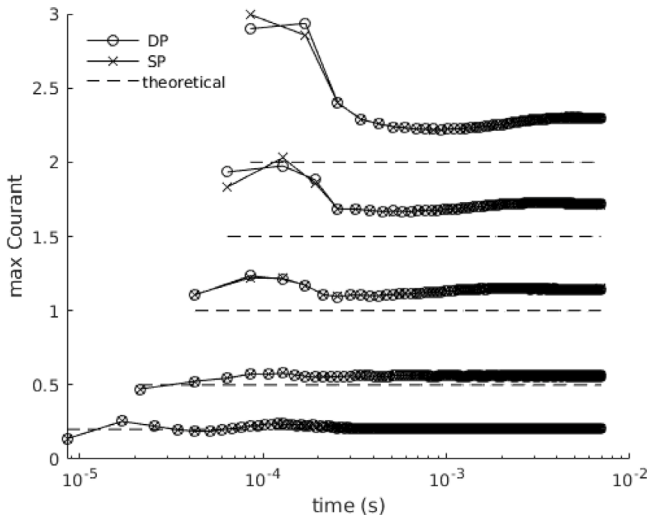


Fig. D.2. Evolution of the maximum Courant number in the shock tube simulations with 800 cells, in single and double precision, with five different constant time steps. For each simulation, the theoretical Courant number (dashed line), computed using the theoretical maximum velocity, is also reported. The time axis is set with a logarithmic scale to better visualize the beginning of the simulation (first few time steps), that is when the numerical error is larger, especially for high Courant number simulations.

Table 5
Numerical setup for the lid-driven cavity solved with `icoFoam`. Please refer to the OpenFOAM user guide for the exact definition and more detailed description of boundary conditions and numerical schemes [1].

Mesh			
3D homogeneous grid (cubic cells)			
Boundary conditions			
	Moving top wall	Walls	
U	Fixed value (U_{max})	Fixed value ($U = 0$)	
p	Zero-gradient	Zero-gradient	
Numerical schemes			
ddt	Gradient	Laplacian	Interpolation
Euler	Gauss linear	Gauss linear	Linear
Divergence			
div(phi,U) Gauss linear			

Table 6
Numerical setup for the shock tube solved with `rhoPimpleFoam`. Please refer to the OpenFOAM user guide for the exact definition and more detailed description of boundary conditions and numerical schemes [1].

Mesh			
1D homogeneous grid (cubic cells)			
Boundary conditions			
	Tube end walls	Tube side walls	
U	Zero-gradient	Empty	
p	Zero-gradient	Empty	
T	Zero-gradient	Empty	
Numerical schemes			
ddt	Gradient	Laplacian	Interpolation
Backward	Gauss linear	Gauss linear	Linear
Divergence			

div(phi,U) Gauss linearUpwindV default;
div(phi,e) Gauss limitedLinear 1;
div(phi,K) Gauss linear;
div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;			

Table 7
Numerical setup for the compressible starting jet solved with `rhoPimpleFoam`. Please refer to the OpenFOAM user guide for the exact definition and more detailed description of boundary conditions and numerical schemes [1]. The test case setup is also available in the repository https://develop.openfoam.com/committees/hpc/-/tree/develop/compressible/rhoPimpleFoam/forcedPlume.

Mesh			
3D homogeneous grid (cubic cells)			
Boundary conditions			
	Inlet (bottom)	Bottom	Vertical/top
U	Fixed value	Fixed value	PressureInletOutletVel.
p	Fixed value	Zero-gradient	TotalPressure
T	Fixed value	Zero-gradient	InletOutlet
Numerical schemes			
ddt	Gradient	Laplacian	Interpolation
Backward	Gauss linear	Gauss linear corr.	Linear
Divergence			

div(phi,U) Gauss linear;
div(phi,e) Gauss limitedLinear 1;
div(phid,p) Gauss limitedLinear 1;
div(phi,K) Gauss linear;
div(phiv,p) Gauss linear;
div(phi,k) Gauss limitedLinear 1;
div(phi,B) Gauss limitedLinear 1;
div(phi,muTilda) Gauss limitedLinear 1;
div(B) Gauss linear;
div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;			

Table 8
Numerical setup for the decaying isotropic turbulence solved with `ASHEE`. Please refer to the OpenFOAM user guide for the exact definition and more detailed description of boundary conditions and numerical schemes [1]. Additional informations about the model and numerical setup can be found in [34].

Mesh			
3D homogeneous grid (cubic cells)			
Boundary conditions			
Cyclic (periodic boundary conditions for all variables in all the 6 walls)			
Numerical schemes			
ddt	Gradient	Laplacian	Interpolation
Backward	Gauss linear	Gauss linear	Linear
Divergence			

div(phi,U) Gauss linear;
div(phi,e) Gauss limitedLinear 1;
div(phi,K) Gauss linear;
div(phid,p) Gauss limitedLinear 1;			

References

[1] OpenFOAM, OpenFOAM project web page of the OpenCFD LTD, 2022, URL: https://www.openfoam.com.

[2] OpenFOAM, OpenFOAM project web page of OpenFOAM foundation, 2022, URL: https://www.openfoam.org.

[3] B. Di Paolo, J.L. Lara, G. Barajas, Í.J. Losada, Waves and structure interaction using multi-domain couplings for Navier-Stokes solvers in OpenFOAM®. Part II: Validation and application to complex cases, Coast. Eng. 164 (2021) 103818.

[4] M. Rauter, L. Hoße, R.P. Mulligan, W. Take, F. Løvholt, Numerical simulation of impulse wave generation by idealized landslides with OpenFOAM, Coast. Eng. 165 (2021) 103815.

[5] G. Axtmann, U. Rist, Scalability of OpenFOAM with large eddy simulations and DNS on high-performance systems, in: High Performance Computing in Science and Engineering´ 16, Springer, 2016, pp. 413–424.

[6] S. Bnà, I. Spisso, M. Olesen, G. Rossi, PETSc4FOAM: A Library to plug-in PETSc into the OpenFOAM Framework, PRACE White paper, 2020.

[7] A. Folch, C. Abril, M. Afanasiev, G. Amati, M. Bader, R.M. Badia, H.B. Bayraktar, S. Barsotti, R. Basili, F. Bernardi, et al., The EU center of excellence for exascale in solid earth (ChEESE): Implementation, results, and roadmap for the second phase, Future Gener. Comput. Syst. 146 (2023) 47–61.

[8] A. Abdelfattah, H. Anzt, E.G. Boman, E. Carson, T. Cojean, J. Dongarra, A. Fox, M. Gates, N.J. Higham, X.S. Li, J. Loe, P. Luszczek, S. Pranesh, S. Rajamanickam, T. Ribizel, B.F. Smith, K. Swirydowicz, S. Thomas, S. Tomov, Y.M. Tsai,

U.M. Yang, A survey of numerical linear algebra methods utilizing mixed-precision arithmetic, Int. J. High Perform. Comput. Appl. 35 (4) (2021) 344–369, http://dx.doi.org/10.1177/10943420211003313, arXiv:https://doi.org/10.1177/10943420211003313.

[9] S. Succi, G. Amati, M. Bernaschi, G. Falcucci, M. Lauricella, A. Montessori, Towards exascale lattice Boltzmann computing, Comput. & Fluids 181 (2019) 107–115.

[10] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, S. Tomov, Accelerating scientific computations with mixed precision algorithms, Comput. Phys. Comm. 180 (12) (2009) 2526–2533.

[11] A. Haidar, S. Tomov, J. Dongarra, N.J. Higham, Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers, in: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2018, pp. 603–613.

[12] A. Abdelfattah, S. Tomov, J. Dongarra, Towards half-precision computation for complex matrices: A case study for mixed precision solvers on gpus, in: 2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA), IEEE, 2019, pp. 17–24.

[13] R. Sakamoto, M. Kondo, K. Fujita, T. Ichimura, K. Nakajima, The effectiveness of low-precision floating arithmetic on numerical codes: A case study on power consumption, in: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, in: HPCAsia2020, Association for Computing Machinery, New York, NY, USA, 2020, pp. 199–206, http://dx.doi.org/10.1145/3368474.3368492.

[14] M. Lehmann, M.J. Krause, G. Amati, M. Sega, J. Harting, 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 (2022) 015308, http://dx.doi.org/10.1103/PhysRevE.106.015308, URL: https://link.aps.org/doi/10.1103/PhysRevE.106.015308.

[15] J.H. Ferziger, M. Perić, R.L. Street, Computational Methods for Fluid Dynamics, Vol. 3, Springer, 2002.

[16] P. Shankar, M. Deshpande, Fluid mechanics in the driven cavity, Annu. Rev. Fluid Mech. 32 (1) (2000) 93–136.

[17] S. Pirozzoli, F. Grasso, Direct numerical simulations of isotropic compressible turbulence: Influence of compressibility on dynamics and structures, Phys. Fluids 16 (12) (2004) 4386, http://dx.doi.org/10.1063/1.1804553.

[18] G.A. Sod, A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws, J. Comput. Phys. 27 (1) (1978) 1–31, http://dx.doi.org/10.1016/0021-9991(78)90023-2, URL: https://www.sciencedirect.com/science/article/pii/0021999178900232.

[19] M. Cerminara, T. Esposti Ongaro, L.C. Berselli, ASHEE-1.0: a compressible, equilibrium–Eulerian model for volcanic ash plumes, Geosci. Model Dev. 9 (2) (2016) 697–730.

[20] H.C. Ku, R.S. Hirsh, T.D. Taylor, A pseudospectral method for solution of the three-dimensional incompressible Navier-Stokes equations, J. Comput. Phys. 70 (2) (1987) 439–462.

[21] S.B. Pope, Turbulent Flows, Cambridge University Press, 2000, p. 771.

[22] G.A. Blaisdell, N.N. Mansour, W.C. Reynolds, Numerical Simulation of Compressible Homogeneous Turbulence, Thermosciences Div. Rep. TF-50, Stanford University, Dept. of Mech. Eng., 1991.

[23] L.-P. Wang, M.R. Maxey, Settling velocity and concentration distribution of heavy particles in homogeneous isotropic turbulence, J. Fluid Mech. 256 (-1) (1993) 27, http://dx.doi.org/10.1017/S0022112093002708.

[24] E. Garnier, M. Mossi, P. Sagaut, P. Comte, M. Deville, On the use of shock-capturing schemes for large-eddy simulation, J. Comput. Phys. 153 (2) (1999) 273–311, http://dx.doi.org/10.1006/jcph.1999.6268.

[25] A.E. Honein, P. Moin, Higher entropy conservation and numerical stability of compressible turbulence simulations, J. Comput. Phys. 201 (2) (2004) 531–545, http://dx.doi.org/10.1016/j.jcp.2004.06.006.

[26] M. Lesieur, O. Métais, P. Comte, Large-Eddy Simulations of Turbulence, Vol. 1, Cambridge University Press, 2005.

[27] W. Liao, Y. Peng, L.-S. Luo, Gas-kinetic schemes for direct numerical simulations of compressible homogeneous turbulence, Phys. Rev. E 80 (4) (2009) 046702, http://dx.doi.org/10.1103/PhysRevE.80.046702.

[28] M. Bernardini, S. Pirozzoli, A general strategy for the optimization of Runge–Kutta schemes for wave propagation phenomena, J. Comput. Phys. 228 (11) (2009) 4182–4199.

[29] M. Bernardini, Reynolds number scaling of inertial particle statistics in turbulent channel flows, J. Fluid Mech. 758 (2014) R1, http://dx.doi.org/10.1017/jfm.2014.561.

[30] M. Bernardini, D. Modesti, F. Salvadore, S. Pirozzoli, STREAmS: A high-fidelity accelerated solver for direct numerical simulation of compressible turbulent flows, Comput. Phys. Comm. 263 (2021) 107906, http://dx.doi.org/10.1016/j.cpc.2021.107906, arXiv:2004.02276.

[31] M. Cerminara, Modeling Dispersed Gas–Particle Turbulence in Volcanic Ash Plumes (Ph.D. thesis), Scuola Normale Superiore, 2016.

[32] A. Yoshizawa, Statistical theory for compressible turbulent shear flows, with the application to subgrid modeling, Phys. Fluids 29 (7) (1986) 2152–2164.

[33] A.W. Woods, Turbulent plumes in nature, Annu. Rev. Fluid Mech. 42 (1) (2010) 391–412, http://dx.doi.org/10.1146/annurev-fluid-121108-145430.

[34] M. Cerminara, T. Esposti Ongaro, A. Neri, Large eddy simulation of gas–particle kinematic decoupling and turbulent entrainment in volcanic plumes, J. Volcanol. Geotherm. Res. 326 (2016) 143–171, http://dx.doi.org/10.1016/j.jvolgeores.2016.06.018, URL: http://linkinghub.elsevier.com/retrieve/pii/S0377027316301688.

[35] A. Neri, T. Esposti Ongaro, M. de' Michieli Vitturi, M. Cerminara, Multiphase flow modeling of explosive volcanic eruptions, ISBN: 9783030685782, 2022, pp. 243–281, http://dx.doi.org/10.1007/978-3-030-68578-2_10, URL: https://link.springer.com/10.1007/978-3-030-68578-2_10.

[36] C.J. Greenshields, H.G. Weller, Note on Computational Fluid Dynamics: General Principles, CFD Direct, 2022, https://cfd.direct.

[37] S. Oliani, N. Casari, M. Carnevale, ICSFoam: An OpenFOAM library for implicit coupled simulations of high-speed flows, Comput. Phys. Comm. 286 (2023) 108673, http://dx.doi.org/10.1016/j.cpc.2023.108673, URL: https://www.sciencedirect.com/science/article/pii/S0010465523000188.

[38] G.G. Ferreira, P.L. Lage, L.F.L. Silva, H. Jasak, Implementation of an implicit pressure–velocity coupling for the Eulerian multi-fluid model, Comput. & Fluids 181 (2019) 188–207, http://dx.doi.org/10.1016/j.compfluid.2019.01.018, URL: https://www.sciencedirect.com/science/article/pii/S0045793019300143.

[39] CINECA, Marconi machine web page, 2022, URL: https://wiki.u-gov.it/confluence/pages/viewpage.action?pageId=132481870.

[40] M. Zounon, N.J. Higham, C. Lucas, F. Tisseur, Performance impact of precision reduction in sparse linear systems solvers, PeerJ Comput. Sci. 8 (2022) e778, http://dx.doi.org/10.7717/peerj-cs.778.

[41] B. John, M. Damodaran, Parallel three dimensional direct simulation Monte Carlo for simulating micro flows, 67, 2009, pp. 91–98, http://dx.doi.org/10.1007/978-3-540-92744-0_11,

[42] Z. Malecha, Ł. Mirosław, T. Tomczak, Z. Koza, M. Matyka, W. Tarnawski, D. Szczerba, et al., GPU-based simulation of 3D blood flow in abdominal aorta using openfoam, Arch. Mech. 63 (2) (2011) 137–161.

[43] A. AlOnazi, D. Keyes, A. Lastovetsky, V. Rychkov, Design and optimization of openfoam-based cfd applications for hybrid and heterogeneous hpc platforms, 2015, arXiv preprint arXiv:1505.07630.

[44] B. Krasnopolsky, A. Medvedev, Acceleration of large scale OpenFOAM simulations on distributed systems with multicore CPUs and GPUs, in: Parallel Computing: On the Road To Exascale, IOS Press, 2016, pp. 93–102.

[45] S. Zampini, S. Bnà, M. Valentini, I. Spisso, GPU-accelerated OpenFOAM simulations using PETSc4FOAM, in: 8th ESI-OpenFOAM Conference, 2020.

[46] M. Martineau, S. Bnà, S. Posey, F. Spiga, OpenFOAM with GPU solver support, in: 9th ESI-OpenFOAM Conference, 2021.

[47] M. Naumov, M. Arsaev, P. Castonguay, J. Cohen, J. Demouth, J. Eaton, S. Layton, N. Markovskiy, I. Reguly, N. Sakharnykh, V. Sellappan, R. Strzodka, AmgX: A library for GPU accelerated algebraic multigrid and preconditioned iterative methods, SIAM J. Sci. Comput. 37 (2015) S602–S626, http://dx.doi.org/10.1137/140980260.

[48] CINECA, Marconi100 machine web page, 2022, URL: https://wiki.u-gov.it/confluence/pages/viewpage.action?pageId=336727645#UG3.2:MARCONI100UserGuide-M100specificinformation.

[49] M.K. Alexander Supalov, C. Dahnken, Optimizing HPC Applications with Intel Cluster Tools: Hunting Petaflops, A Press, New York, 2014.

[50] T. Hoefler, W. Gropp, R. Thakur, J. Träff, Toward performance models of MPI implementations for understanding application scaling issues, 2010, pp. 21–30, http://dx.doi.org/10.1007/978-3-642-15646-5_3.

Federico Brogi is a computational scientist at the National Institute of Geophysics and Volcanology, Italy. He obtained his Ph.D. in 2017 at the University of Geneva, Switzerland and his master's degree in geophysics at the University of Trieste, Italy. His research is devoted to developing and optimising computational tools to model volcanic processes and associated geophysical signals by using computational fluid dynamics and high-performance computing.

Simone Bnà is a HPC specialist for academic and industrial CFD applications at the High Performance Computing Center of Italy (CINECA). He received the Ph.D. Degree in Energy, Nuclear and Environmental Control Engineering from the University of Bologna, Italy in 2014. He has published on various topics in nuclear thermal hydraulics, two-phase flow and fluid–structure interactions. In CINECA he worked on EU and privately funded projects in the scientific visualization and CFD modelling field.

Gabriele Boga is a Ph.D. student at the University of Modena and Reggio Emilia at the Engineering Department (DIEF), Italy. He received his master's degree in aerospace engineering from the University of Bologna, Italy in 2020. He worked as HPC Developer at the High Performance Computing Center of Italy (CINECA) on CFD applications. His research is devoted to the study of turbulence, both in wall-bounded and free-shear flows.

Giorgio Amati is a Senior HPC specialist for CFD applications at the High Performance Computing Center of Italy (CINECA). He received the Ph.D. Degree in Mechanical and Aerospace Engineering from the University of Rome "La Sapienza" in 1997. His research focuses on numerical models, codes and benchmarking. In CINECA he worked on EU funded projects, in CFD modelling field and as technology scouting.

Tomaso Esposti Ongaro is a senior researcher in physical and computational volcanology at the National Institute of Geophysics and Volcanology, Italy. His research interests mostly focus on the multiphase fluid dynamics of gas–particle flows in explosive volcanic eruptions and the application of three-dimensional numerical modelling to the assessment of volcanic hazards.

Matteo Cerminara is a researcher in physical and computational volcanology at the National Institute of Geophysics and Volcanology, Italy. He received the Ph.D. Degree in Applied Mathematics from the Scuola Normale Superiore, Italy in 2016. His research focuses on numerical models, codes, and workflows for volcanology, from their development to model validation and comparison of synthetic data with observations. Main applications are volcanic plumes, pyroclastic density currents and tsunamis.