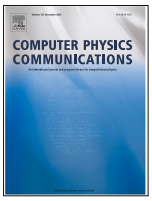
Computer Physics Communications 322 (2026) 110055

Contents lists available at ScienceDirect

Computer Physics Communications

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



Computer Programs in Physics

MFC 5.0: An exascale many-physics flow solver

Benjamin Wilfong [a,1], Henry A. Le Berre [a,1], Anand Radhakrishnan [a,1], Ansh Gupta [a], Daniel J. Vickers [a], Diego Vaca-Revelo [b], Dimitrios Adam [a], Haocheng Yu [c], Hyeoksu Lee [d], Jose Rodolfo Chreim [d], Mirelys Carcana Barbosa [e], Yanjun Zhang [d], Esteban Cisneros-Garibay [f], Aswin Gnanaskandan [b], Mauro Rodriguez Jr [e], Reuben D. Budiardja [g], Stephen Abbott [h], Tim Colonius [d], Spencer H. Bryngelson [a,c,i,*]

[a] School of Computational Science & Engineering, Georgia Institute of Technology, Atlanta, GA, 30332, USA
[b] Mechanical and Materials Engineering, Worcester Polytechnic Institute, Worcester, MA, 01609, USA
[c] Daniel Guggenheim School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA, 30332, USA
[d] Department of Mechanical and Civil Engineering, California Institute of Technology, Pasadena, CA, 91125, USA
[e] School of Engineering, Brown University, Providence, RI, 02912, USA
[f] Mechanical Science & Engineering, University of Illinois at Urbana–Champaign, Urbana, IL, 61820, USA
[g] Oak Ridge National Laboratory, 37830, Oak Ridge, TN, USA
[h] Hewlett Packard Enterprise, Bloomington, MN, 55435, USA
[i] George W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA, 30332, USA

ARTICLE INFO

Editor: Prof. Andrew Hazel

Keywords:
Computational fluid dynamics
Multiphase flow
Exascale simulation
Numerical methods and modeling

ABSTRACT

Many problems of interest in engineering, medicine, and the fundamental sciences rely on high-fidelity flow simulation, making performant computational fluid dynamics solvers a mainstay of the open-source software community. Previous work MFC 3.0 was made a published, documented, and open-source solver via Bryngelson et al. *Comp. Phys. Comm.* (2021) with numerous physical features, numerical methods, and scalable infrastructure. MFC 5.0 is a significant update to MFC 3.0, featuring a broad set of well-established and novel physical models and numerical methods, as well as the introduction of GPU and APU (or superchip) acceleration. We exhibit state-of-the-art performance and ideal scaling on the first two exascale supercomputers, OLCF Frontier and LLNL El Capitan. Combined with MFC's single-accelerator performance, MFC achieves exascale computation in practice, and achieved the largest-to-date public CFD simulation at 200 trillion grid points as a 2025 ACM Gordon Bell Prize finalist. New physical features include the immersed boundary method, N-fluid phase change, Euler–Euler and Euler–Lagrange sub-grid bubble models, fluid-structure interaction, hypo- and hyper-elastic materials, chemically reacting flow, two-material surface tension, magnetohydrodynamics (MHD), and more. Numerical techniques now represent the current state-of-the-art, including general relaxation characteristic boundary conditions, WENO variants, Strang splitting for stiff sub-grid flow features, and low Mach number treatments. Weak scaling to tens of thousands of GPUs on OLCF Summit and Frontier and LLNL El Capitan achieves efficiencies within 5% of ideal to over 90% of their respective system sizes. Strong scaling results for a 16-times increase in device count show parallel efficiencies over 90% on OLCF Frontier. MFC's software stack has undergone further improvements, including continuous integration, which ensures code resilience and correctness through over 300 regression tests; metaprogramming, which reduces code length while maintaining performance portability; and code generation for computing chemical reactions

PROGRAM SUMMARY
Program title: MFC 5.0
CPC Library link to program files: https://doi.org/10.17632/8y55zscjd3.2
Developer's repository link: https://github.com/MFlowCode/MFC
Licensing provisions: MIT license
Programming language: Fortran08 and Python

* Corresponding author.
E-mail address: sbryngelson3@gatech.edu, shb@gatech.edu (S.H. Bryngelson).
[1] Equal contribution.

https://doi.org/10.1016/j.cpc.2026.110055
Received 12 April 2025; Received in revised form 18 January 2026; Accepted 23 January 2026
Available online 30 January 2026

Journal reference of previous version: Comput. Phys. Commun. 266 (2021) 107396, doi:https://doi.org/10.1016/j.cpc.2020.107396
Does the new version supersede the previous version?: Yes
Reasons for the new version: An abundance of new features and capabilities that are challenging to convey outside of an academic article.
Nature of problem: Simulation of compressible flows requires physical model selection and treatment of spatial derivatives to keep solutions physically consistent and free of artifacts. The methods should be high-order accurate to reduce computational cost.
Solution method: The present software uses multiple physical models and numerical schemes for the treatment of multi-phase, multi-component, chemically reacting, and magnetohydrodynamic flows. Additional physical effects and models are included. The numerical method is based on the finite volume method.

1. Introduction

MFC was introduced to the literature and open source community via Bryngelson et al. [1], which described the software design and features of MFC 3.0, a CPU-based compressible multi-component flow code (MFC) for the 5-equation diffuse interface methods of Kapila et al. [2] and Allaire et al. [3] and the six equation diffuse interface model of Saurel et al. [4]. MFC 3.0 included other features that are discussed throughout this manuscript. Since MFC 3.0's release, the MFC community has added a broad set of physical models, numerical methods, GPU offloading capabilities, and performance optimizations. This paper provides a holistic overview of MFC's development over the past four years. We present physical models, numerical methods, validity, software robustness and testing, and performance on diverse architecture sets at scale on exascale machines and beyond.

1.1. The history of MFC

The MFC codebase archaeology dates back to the mid-2000s. At this time, MFC was an unnamed solver developed by Eric Johnsen. This code represented a multi-component flow using the diffuse interface method, as described in Johnsen [5] and Johnsen and Colonius [6]. Following this effort, Vedran Coralic performed a major code rewrite and adorned MFC with its name. These features adorned MFC with a multi-dimensional numerical reconstruction treatment and are described in Coralic and Colonius [7]. Jomela Meng added cylindrical coordinate systems and corresponding simulations of high-speed shock–droplet interaction [8]. Kazuki Maeda added an Euler–Lagrange representation of sub-grid bubble dynamics [9] and acoustic wave generation [10]. Kevin Schmidmayer and Spencer Bryngelson added the 5-equation model of Kapila et al. [2] and the 6-equation model of Saurel et al. [4]. These efforts were conducted in the research group of Tim Colonius at the California Institute of Technology. Spencer Bryngelson later led a restructuring of MFC at the Georgia Institute of Technology, which included additional modeling and numerical features. This effort culminated in the open-sourcing of MFC 3.0 in 2020 and is largely described in Bryngelson et al. [1]. Modeling, method, and software scalability and portability capabilities were added in subsequent years, which are discussed in this manuscript.

1.2. New MFC capabilities

To make high-fidelity CFD a practical instrument for today's many-physics problems, a solver must span liquids, gases, and solids (at least) and accommodate complex geometries and broad separations of time and length scales. At the same time, this should be accomplished efficiently on modern accelerators, which serve as the backbone of current supercomputing centers. MFC 5.0 is architected around that premise. The new capabilities unify established and novel models. Each of these capabilities is accompanied by state-of-the-art numerics and GPU/APU offloading, allowing a single code to transition from low-Mach, geometry-rich flows to shock- and detonation-dominated regimes at exascale. In this sense, the new capabilities were crafted to establish an infrastructure for credible, end-to-end simulations across engineering, medicine, and the sciences.

MFC now includes more capable physical models and numerical methods, software resistance through continuous integration and deployment, code coverage testing, and computation acceleration via GPU and APU devices. With these, MFC conducted the largest publicly known CFD simulation at 200T grid points, or 1 quadrillion degrees of freedom [11], and was an ACM 2025 Gordon Bell Prize Finalist. Physical features include six new models for phase change, non-polytropic sub-grid bubble dynamics, treatment for elastic materials, chemical reactions and combustion, and surface tension. An immersed boundary method that supports complex geometries from analytical level sets or STL files. New numerical methods include generalized characteristic boundary conditions, improved shock capturing with TENO and WENO-Z constructions, Strang splitting for stiff sub-grid dynamics, and treatment for low-Mach number flow. Significant software additions have made MFC a portable and performant solver on CPUs, GPUs, and APUs from various vendors, including Intel, AMD, and NVIDIA. Decreases in runtime are, in part, enabled via metaprogramming and static code generation.

1.3. Use of MFC

MFC has a history of being used for early access programs on flagship supercomputers. At the time of writing, these programs include JSC JUPITER (JUREAP) and the LLNL El Capitan and OLCF Frontier final and early access systems, including LLNL Tioga (AMD MI300A testbed) and OLCF Spock and Crusher (AMD MI100 and MI250X testbeds). Commodity cluster use of MFC is broad, including the ACCESS-CI [12] (formerly XSEDE [13]) clusters (PSC Bridges [14] and Bridges2 [15], SDSC Comet [16] and Expanse [17], Purdue Anvil [18], NCSA Delta [19] and DeltaAI [20], TACC Stampede1–3 [21–23], among others) university clusters, cloud computer systems (Intel's AI cloud [24]), and AMD and Cray's internal systems, among numerous others. MFC is a SPEChpc benchmark candidate, which comprises software that evaluates the performance of supercomputers [25]; it is also used as part of the OLCF Test Harness for the performance and robustness of OLCF Frontier [26].

1.4. Manuscript structure

In Section 2, we introduce MFC 5.0, describing the code and methods it builds upon while connecting it to MFC 3.0. Section 3 discusses established numerical methods that MFC builds upon. New features follow in Section 4, including physical models, numerical methods, and software tooling. We show example MFC simulations in Section 5. A discussion of other open source flow solvers with some similar features in Section 6. Section 7 summarizes the manuscript and MFC 5.0.

2. Multiphase models

2.1. Multi-component treatment

MFC uses reduced versions of the Baer–Nunziato model [27] and diffuse interface methods to model multiphase flow. The core of the

methodology, as described in this subsection, remains unchanged from MFC 3.0 [1] and readers are directed there for a more fundamental understanding and detailed algorithmic strategy. Changes to the fundamental diffuse interface methodology are limited, except for performance improvements, code testing strategies, and additional physical features that are now accommodated. These additional features are discussed in Section 4.

2.1.1. Five-equation models

For a two-fluid flow configuration, the Baer–Nunziato model reduces to the so-called five-equation model

$$\begin{aligned}
\frac{\partial \alpha_i \rho_i}{\partial t} + \nabla \cdot (\alpha_i \rho_i \boldsymbol{u}) &= 0,\\
\frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u} + p\boldsymbol{I} - \boldsymbol{T}^v) &= 0,\\
\frac{\partial \rho E}{\partial t} + \nabla \cdot \left[(\rho E + p)\boldsymbol{u} - \boldsymbol{T}^v \cdot \boldsymbol{u}\right] &= 0,\\
\frac{\partial \alpha_i}{\partial t} + \boldsymbol{u} \cdot \nabla \alpha_i &= K \nabla \cdot \boldsymbol{u},
\end{aligned}$$

of Kapila et al. [2] by assuming that both species share a velocity and pressure, and $i = 1, \ldots, N_{\text{comp.}}$ where $N_{\text{comp.}}$ is the number of fluid components. With this model, ρ, $\boldsymbol{u}$, p, and E are the mixture density, velocity, pressure, and total energy (per unit mass). The α_i and ρ_i are the volume fraction and density of component i. Viscous terms are introduced via the mixture viscous stress tensor

$$\boldsymbol{T}^v = 2\eta(\boldsymbol{D} - \operatorname{tr}(\boldsymbol{D})\boldsymbol{I}/3), \tag{1}$$

where η is the mixture shear viscosity, and the strain rate tensor is

$$\boldsymbol{D} = (\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^\top)/2. \tag{2}$$

The mixture internal energy e is

$$e = Y_1 e(\rho_1 p) + Y_2 e(\rho_2 p), \tag{3}$$

where $Y_i = \alpha_i \rho_i / \rho$ is the mass fraction of component i. The model is closed using the mixture rules

$$1 = \sum_{i=1}^{N} \alpha_i, \quad \rho = \sum_{i=1}^{N} \alpha_i \rho_i, \quad \rho e = \sum_{i=1}^{N} \alpha_i \rho_i e_i. \tag{4}$$

For two components, K is

$$K = \frac{\rho_2 c_2^2 + \rho_1 c_1^2}{\rho_1 c_1^2/\alpha_1 + \rho_2 c_2^2/\alpha_2}, \tag{5}$$

where c_i is phasic sound speed

$$c_i = \sqrt{\gamma_i (p + \pi_{\infty,i})/\rho_i}. \tag{6}$$

The term $K\nabla \cdot \boldsymbol{u}$ ensures thermodynamic consistency and accounts for the expansion and compression of each species in mixture regions. The $K\nabla \cdot \boldsymbol{u}$ is necessary to model phenomena like spherical bubble collapse, but it is non-conservative and can lead to numerical instabilities [28]. For some problems, $K\nabla \cdot \boldsymbol{u}$ is not necessary, and the model reduces to that of Allaire et al. [3]. The differences between these models are discussed in Schmidmayer et al. [28].

2.1.2. Six-equation model

The six-equation model of Saurel et al. [4] allows for pressure disequilibrium between phases and can be used to avoid numerical stability issues associated with the five-equation Kapila model while maintaining thermodynamic consistency. For two fluids, the six-equation model is

$$\begin{aligned}
\frac{\partial \alpha_i \rho_i}{\partial t} + \nabla \cdot (\alpha_i \rho_i \boldsymbol{u}) &= 0,\\
\frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u} + p\boldsymbol{I} - \boldsymbol{T}^v) &= 0.\\
\frac{\partial \alpha_1 \rho_1 e_1}{\partial t} + \nabla \cdot (\alpha_1 \rho_1 e_1 \boldsymbol{u}) + \alpha_1 p_1 \cdot \nabla \boldsymbol{u} &= -\mu p_1 (p_2 - p_1) - \alpha_1 \boldsymbol{T}_1^v : \nabla \boldsymbol{u},\\
\frac{\partial \alpha_2 \rho_2 e_2}{\partial t} + \nabla \cdot (\alpha_1 \rho_2 e_2 \boldsymbol{u}) + \alpha_2 p_2 \cdot \nabla \boldsymbol{u} &= -\mu p_1 (p_1 - p_2) - \alpha_2 \boldsymbol{T}_2^v : \nabla \boldsymbol{u},\\
\frac{\partial \alpha_1}{\partial t} + \boldsymbol{u} \cdot \nabla \alpha_1 &= \mu (p_1 - p_2).
\end{aligned}$$

Here, ρ and $\boldsymbol{u}$ are the mixture density and velocity, α_i, ρ_i, p_i, e_i, and $\boldsymbol{T}_i^v$ are the volume fraction, density, pressure, internal energy, and species viscous stress tensor of species i, and μ is a pressure relaxation parameter. The interfacial pressure P_I is

$$P_I = \frac{z_2 p_1 + z_1 p_2}{z_1 + z_2}, \tag{7}$$

where $z_i = \rho_i c_i$ is the phasic acoustic impedance, and the phasic speed of sound c_i is

$$c_i = \sqrt{\gamma (p_i + \pi_{\infty,i})/\rho_i}. \tag{8}$$

The model is closed using the same mixture rules of the five-equation model in Section 2.1.1. Following Schmidmayer et al. [28], the six-equation model in the limit of infinite relaxation can represent the same physical processes as the model of Kapila et al. [2], but avoids numerical instability issues that arise from some interface capture schemes.

2.2. Equation of state

The five- and six-equation models are closed using the stiffened gas equation of state (EOS), which accurately models liquids and gases [29]. In its simplest form, the stiffened gas EOS relates the internal energy to the pressure and density of each species i as

$$e_i = \frac{p_i + \gamma_i \pi_{\infty,i}}{(\gamma_i - 1)\rho_i},$$

where e_i, p_i, and ρ_i are the internal energy, pressure, and density of species i. Parameter γ_i and the liquid stiffness $\pi_{\infty,i}$ can be tuned to represent different liquids and gases.

2.3. Cylindrical coordinates

Axisymmetric and cylindrical coordinates are implemented via the strategies of Johnsen [5] and Meng [8]. The singularity at $r \to 0$ is handled by placing the axis at a cell boundary such that the radius is defined at the cell center. Then, the solution in cells 180 degrees apart from each other supports stencils for reconstruction. Spectral filtering in the azimuthal direction relaxes the strict time-step restrictions induced by the small cells near the axis [30].

3. Numerical methods

We next describe the method used to solve for the advective and diffusive terms in the multi-component models of MFC. The strategy employed here aligns with that of Bryngelson et al. [1], and the reader is directed to that work for further details.

3.1. Finite volume method

The models in MFC are solved using a finite volume method based on the framework of Coralic and Colonius [7]. The model equations take the form

$$\frac{\partial \boldsymbol{q}}{\partial t} + \frac{\partial \boldsymbol{F}^{(x)}(\boldsymbol{q})}{\partial x} + \frac{\partial \boldsymbol{F}^{(y)}(\boldsymbol{q})}{\partial y} + \frac{\partial \boldsymbol{F}^{(z)}(\boldsymbol{q})}{\partial z} = \boldsymbol{s}(\boldsymbol{q}) - \boldsymbol{h}(\boldsymbol{q}) \nabla \cdot \boldsymbol{u}, \tag{9}$$

where $\boldsymbol{q}$ is the vector of cell-averaged conservative variables, $\boldsymbol{s}$ and $\boldsymbol{h}(\boldsymbol{q})$ are source terms, and $\boldsymbol{F}^x$, $\boldsymbol{F}^y$, and $\boldsymbol{F}^z$ are the fluxes in the x-, y- and z-directions. Eq. (9) is integrated in space as

$$\begin{aligned}
\frac{\partial \boldsymbol{q}_{i,j,k}}{\partial t} = &\frac{1}{\Delta x_i}\left(\boldsymbol{F}^{(x)}_{i-1/2,j,k} - \boldsymbol{F}^{(x)}_{i+1/2,j,k}\right) + \frac{1}{\Delta y_j}\left(\boldsymbol{F}^{(y)}_{i,j-1/2,k} - \boldsymbol{F}^{(y)}_{i,j+1/2,k}\right)\\
&+ \frac{1}{\Delta z_k}\left(\boldsymbol{F}^{(z)}_{i,j,k-1/2} - \boldsymbol{F}^{(z)}_{i,j,k+1/2}\right) + \boldsymbol{s}(q_{i,j,k}) - \boldsymbol{h}(\boldsymbol{q}_{i,j,k})(\nabla \cdot \boldsymbol{u})_{i,j,k}.
\end{aligned} \tag{10}$$

The flux $\boldsymbol{F}^x(\boldsymbol{q}_{i+1/2,j,k})$ is calculated at the center of the finite volume face, and the other coordinate directions follow in the same fashion.

3.2. Shock and interface capturing

3.2.1. WENO reconstructions

The fluxes in finite-volume methods are calculated using the left and right states of each interface by solving the Riemann problem

$$\boldsymbol{F}^{(x)}_{i+1/2,j,k} = \boldsymbol{F}^{(x)}\left(\boldsymbol{q}^{(\mathrm{L})}_{i+1/2,j,k}, \boldsymbol{q}^{(\mathrm{R})}_{i+1/2,j,k}\right),$$
$$\boldsymbol{u}^{(x)}_{i+1/2,j,k} = \boldsymbol{u}^{(x)}\left(\boldsymbol{q}^{(\mathrm{L})}_{i+1/2,j,k}, \boldsymbol{q}^{(\mathrm{R})}_{i+1/2,j,k}\right),$$

where superscripts (L) and (R) indicate the state variables reconstructed from the left- and right-hand sides of the finite volume interface. WENO reconstructions obtain high-order spatial accuracy by considering a convex combination of lower-order reconstructions. A $(2k-1)$th order WENO reconstruction in the x-direction uses k staggered candidate polynomials to reconstruct the state variable $f_{i+1/2,j,k}$ with $f^{(r)}_{i+1/2,j,k}$ for $r = 0, 1, \ldots, k-1$. The weighted sum of a set of crafted candidate polynomials gives the reconstructed state variables. Fifth-order accuracy is maintained through mapped weights, as described by Henrick et al. [31].

3.2.2. Approximate Riemann solver

The Riemann problem at each finite volume interface is solved using either the Harten–Lax–van Leer (HLL) or the Harten–Lax-van Leer contact (HLLC) approximate Riemann solver [32]. The HLL approximate Riemann solver admits three constant states separated by two discontinuities with different wave speeds. The wave speeds are estimated using the left and right state variables $\boldsymbol{q}^{(\mathrm{L})}_{i+1/2,j,k}$ and $\boldsymbol{q}^{(\mathrm{R})}_{i+1/2,j,k}$. The HLL flux satisfies the Rankine–Hugoniot conditions that ensure the conservation of mass, momentum, and energy across the discontinuity. A shortcoming of the HLL approximate Riemann solver is that it does not account for the contact discontinuity in the region between the two wave speeds (the so-called "star" region). As a result, the HLL approximate Riemann solver has more numerical dissipation than the HLLC approximate Riemann solver, which accounts for this third contact discontinuity. Continuity of normal velocity and pressure is imposed across the contact discontinuity, which enables computations of the star states and the resulting HLLC flux. Once computed, the HLL and HLLC fluxes are used to calculate the right-hand side of Eq. (10). Identical calculations are performed in the x- and z-directions.

3.3. Boundary conditions

Extrapolation and Characteristic [33] boundary conditions are implemented using buffer regions outside the domain to support the stencils of the finite volume reconstructions and provide information outside of the domain for characteristic boundary conditions. Subsonic and supersonic free-stream and wall boundary conditions are implemented to handle various simulation configurations. Simple boundary conditions remain a mainstay of MFC, which were established in Bryngelson et al. [1]. These boundary conditions include symmetry, slip and no-slip walls, prescription of inlet conditions along partial boundary regions, and first-order accurate extrapolation for Neumann-like boundary conditions.

3.4. Time integration

The conservative variables are advanced in time using high-order total variation diminishing (TVD) and strong stability preserving (SSP) explicit Runge–Kutta time steppers [34]. First- and second-order accurate time-steppers Runge–Kutta are also available.

3.5. Convergence

The convergence of the existing numerical methods with the five- and six-equation models is verified using a one-dimensional two-component advection problem with periodic boundary conditions. The computational domain spans the interval $[0, 1]$ with a uniform pressure and velocity equal to one. The volume fraction and density of each component are defined as $\alpha_i = \alpha_i \rho_i = 0.5 + (-1)^i \sin(2\pi x)$ and sum to one everywhere in the domain.

Fig. 1 shows the RMS error after five periods of advection for 1st, 3rd, and 5th order accurate reconstructions using the HLL and HLLC approximate Riemann solvers. Convergence results can be reproduced using the automated scripts provided in the `examples/1D_convergence` example case.

4. Updated capability

This section is organized by the role each additional physical, numerical, or performance capability plays in the overarching goal of providing an end-to-end state-of-the-art simulation platform for a broad range of flow problems. First, we broaden physical coverage from MFC 3.0 (see Bryngelson et al. [1]) to capture the couplings that govern realistic flows (immersed boundaries for complex geometry; N-fluid phase change; Euler–Euler and Euler–Lagrange bubble models; fluid-structure interaction; chemically reacting mixtures; two-material surface tension; MHD/RMHD). We equip the solver with the numerical machinery required to preserve accuracy and robustness across regimes (general characteristic boundary conditions, WENO-Z/TENO reconstructions, Strang splitting for stiff sub-grid dynamics, and low-Mach treatments). Lastly, we enhance the software's performance and reproducibility, including portable GPU/APU offload, metaprogramming and code generation, continuous integration with over 500 regression tests, and extreme-scale I/O. So, these methods run consistently from a single accelerator to tens of thousands of devices.

4.1. Physical features

4.1.1. Immersed boundary method

Fluid-solid interactions with rigid bodies are simulated using the Ghost-Cell Immersed Boundary Method with an implementation similar to Tseng and Ferziger [35]. This method imposes Neumann boundary conditions for the pressure, densities, and volume fractions, as well as a no-slip or slip boundary condition for the velocity on the immersed boundary.

In MFC 5.0, the solid geometries are parsed. Each cell is characterized as being either within a fluid or a solid region. For each immersed boundary, the level set field $\boldsymbol{\varphi}$ is calculated. This field stores the vector from each cell to the closest point on the immersed boundary. The ghost points, image points, and interpolation coefficients are calculated and used to enforce the appropriate boundary conditions. These are computed before simulation execution and follow the algorithm below.

Algorithm 1 Calculation of level sets.

```
for each immersed boundary Γ_i do
    for each cell do
        if Γ_i is defined as a simple geometry then        ▷ i.e: Sphere, Cuboid, etc.
            Calculate the level set with an analytical function.
        else if Γ_i is defined by an STL/OBJ file then
            Calculate the level set using the procedure described in Section 4.1.2.
        end if
    end for
end for
```

The algorithm that applies the boundary conditions follows. MFC first parses the STL or OBJ geometry file (or level set). We determine whether each finite volume is in a solid, immersed region by examining its surface normals. We use Algorithm 1 to compute the level set field for each immersed boundary. This field stores the vector from each finite

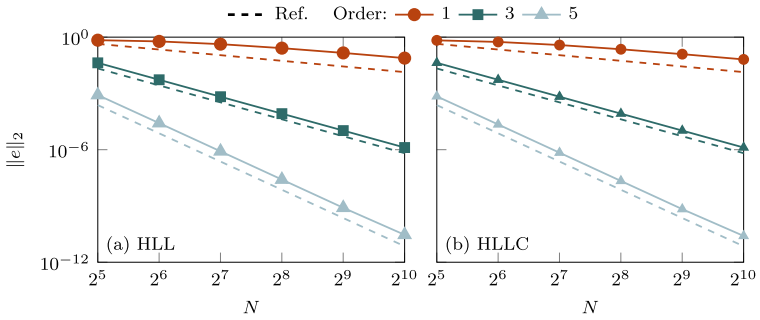


Fig. 1. Convergence results for a 1D 2-component advection problem with the (a) HLL and (b) HLLC approximate Riemann solvers.

volume center to the closest point on the immersed boundary. If each cell center in the solid regions is within three cells of a fluid cell, we characterize it as a ghost point. The fluid properties at the image points are interpolated from surrounding grid cells. We follow Algorithm 2 to compute the ghost points, image points, and the interpolation coefficients. Finite volumes in an immersed solid region are excluded from interpolation. Following Algorithm 3, we compute $q_{(\text{ip})} = q_{(\text{gp})}$ for each fluid property q with a Neumann boundary condition and set the velocity based on its boundary conditions.

The independence of ghost point treatment allows the algorithm to be readily parallelized. The computational cost of applying boundary conditions across the immersed body is negligible compared to the rest of the flow simulation.

Algorithm 2 Determining ghost points, image points, and interpolation coefficients.

for each cell **do**
  if the cell is in a solid region and it is within three cells of the fluid region **then**
    Add cell to the list of ghost points
  end if
end for
for each ghost point **do**
  $\boldsymbol{x}_{(\text{ip})} = \boldsymbol{x}_{(\text{gp})} + 2\boldsymbol{\varphi}(\boldsymbol{x}_{(\text{gp})})$ ▷ $\boldsymbol{x}_{(\text{ip})}$ is the image point position, $\boldsymbol{x}_{(\text{gp})}$ is the ghost point position
  Determine the four grid cells surrounding the image point
  for each surrounding grid cell $\boldsymbol{g}_j(\boldsymbol{x}_{(\text{ip})})$ **do**
    if $\boldsymbol{g}_j(\boldsymbol{x}_{(\text{ip})})$ is in the fluid region **then**
      $d = \|\boldsymbol{x}_{(\text{ip})} - \boldsymbol{g}_j(\boldsymbol{x}_{(\text{ip})})\|_2$ ▷ Distance from image point to grid cell
      $c_j(\boldsymbol{x}_{(\text{ip})}) = 1/d^2$
    else
      $c_j(\boldsymbol{x}_{(\text{ip})}) = 0$
    end if
  end for
  $c(\boldsymbol{x}_{(\text{ip})}) = c(\boldsymbol{x}_{(\text{ip})}) / \sum_j c_j(\boldsymbol{x}_{(\text{ip})})$ ▷ Normalize the interpolation coefficients
end for

4.1.2. Treatment of complex geometries

We use a ray-tracing algorithm to translate complex geometries in ASCII and stereolithography (STL) formats onto computational grids compatible with the immersed boundary method. The ASCII STL format represents geometric models using a triangular mesh. Each model surface is decomposed into a set of non-overlapping triangular facets. Three vertices and a normal vector define each triangular facet. Unlike simple geometries such as cubes and spheres, most complex geometric models do not have analytical solutions for their boundary normal vectors. Consequently, the level sets of complex geometries mentioned in Section 4.1.1 also lack analytical solutions. We need to approximate the geometry's boundary and boundary normal vectors using the STL vertices located on the boundary. We use an edge-manifoldness algorithm based on Weiler [36] to group the boundary STL vertices that are used to determine the level sets and image points. Coarse STL files are interpolated during ray tracing to achieve results consistent with those obtained from higher-resolution STL files. The implementation is validated for a Mach 2 helium flow over a sphere with initial flow density ρ_0 and $\text{Re} = 6.5 \times 10^5$ in Fig. 2.

Algorithm 3 Ghost point treatment at each timestep.

for each ghost point **do**
  for each fluid property q **do**
    $q(\boldsymbol{x}_{(\text{ip})}) = \sum q(\boldsymbol{g}_i(\boldsymbol{x}_{(\text{ip})})) \cdot c_i(\boldsymbol{x}_{(\text{ip})})$
    if q is velocity **then**
      if velocity boundary condition is no-slip **then**
        $\boldsymbol{u}(\boldsymbol{x}_{(\text{gp})}) = 0$
      else if velocity boundary condition is slip **then**
        $\boldsymbol{u}(\boldsymbol{x}_{(\text{gp})}) = \boldsymbol{u}(\boldsymbol{x}_{(\text{ip})}) - (\hat{\boldsymbol{n}} \cdot \boldsymbol{u}(\boldsymbol{x}_{(\text{ip})}))\hat{\boldsymbol{n}}$
      end if
    else ▷ The fluid property has a Neumann boundary condition
      $q(\boldsymbol{x}_{(\text{gp})}) = q(\boldsymbol{x}_{(\text{ip})})$
    end if
  end for
end for

4.1.3. Phase change: First order transition, liquid–vapor

The phase change model for N fluids is integrated into MFC implemented by introducing disequilibrium terms for pressure (p), temperature (T), and chemical potential (g) into $\boldsymbol{s}(\boldsymbol{q})$ in Eq. (9). This inclusion increases the system stiffness, making a fractional-step method essential [4,37,38]. In the case of infinitely fast relaxation, the governing PDEs can be replaced with nonlinear, lower-dimensional, algebraic relations. For example, for the pT-relaxation, we have [39]

$$f(p) = \sum_{i=1}^{N} \alpha_i - 1 = \sum_{i=1}^{N} \frac{\gamma_i - 1}{\gamma_i} \frac{\alpha_i \rho_i c_{p,i}}{\sum_{j=1}^{N} \alpha_j \rho_j c_{p,j}} \frac{\rho e + p - \sum_{j=1}^{N} \alpha_j \rho_i e_{*,i}}{\sum_{i=1}^{N} \alpha_i \rho_i c_{p,i}} - 1 = 0,$$

and for the pTg-relaxation

$$F_1(\alpha_1\rho_1, p) = A + \frac{B}{T} + C\ln(T) + D\ln(p + \pi_{\infty,1}) - \ln(p + \pi_{\infty,2}) = 0,$$

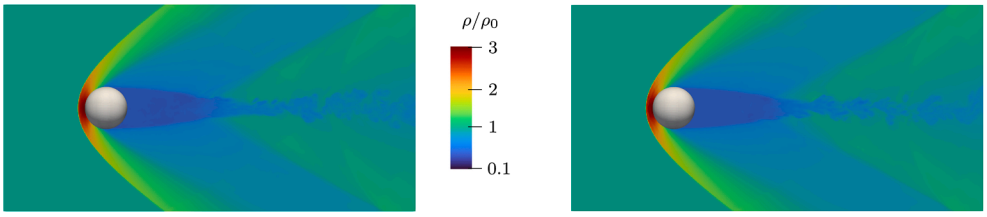
and

$$F_2(\alpha_1\rho_1, p) = \rho e + p + \alpha_1\rho_1(e_{*,2} - e_{*,1}) - \alpha_m\rho_m e_{*,2} - \sum_{i=3}^{N} \alpha_i\rho_i q_i + T\left[\alpha_1\rho_1(c_{p,2} - c_{p,1}) - \alpha_m\rho_m c_{p,2} - \sum_{i=3}^{N} \alpha_i\rho_i c_{p,i}\right] = 0,$$

in which $T = T(\alpha_1\rho_1, p)$ is the relaxed temperature

$$T(\alpha_1\rho_1, p) = \left\{ \alpha_1\rho_1 \left[\frac{c_{p,1} - c_{v,1}}{p + \pi_{\infty,1}} - \frac{c_{p,2} - c_{v,2}}{p + \pi_{\infty,2}} \right] + \alpha_m\rho_m \left[\frac{c_{p,2} - c_{v,2}}{p + \pi_{\infty,2}} \right] + \sum_{i=3}^{N} \left[\alpha_i\rho_i \frac{c_{p,i} - c_{v,i}}{p + \pi_{\infty,i}} \right] \right\}^{-1},$$

with $\alpha_m\rho_m = \alpha_1\rho_1 + \alpha_2\rho_2$, $e_{*,i}$ a reference parameter added to the stiffened gas EOS (see Section 2.2) internal energy equation, and $c_{v,i}$ and $c_{p,i}$ the



(a) Analytical (b) STL

Fig. 2. Steady density field (ρ/ρ_0) of Mach 2 helium flow over a sphere with (a) analytical level set and (b) STL-based level set.

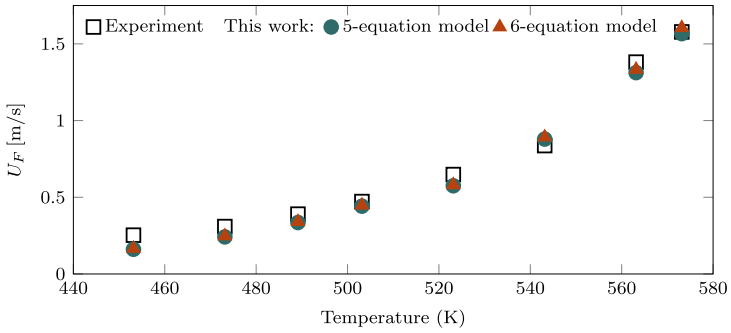


Fig. 3. Comparison between the numerical and experimental values for the evaporation front velocity using the 5- and 6-equation models for the chocked flow series of n-dodecane tests [41].

specific heat coefficient at constant volume and pressure, respectively, for species i. Subscripts $(\cdot)_{1,2}$ indicate the reacting liquid and vapor. The remaining subscripts are the inert phases. A, B, C, and D are fluid-dependent parameters [40], reproduced here for clarity:

$$A = \frac{c_{p,1} - c_{p,2} + e'_{*,2} - e'_{*,1}}{c_{p,2} - c_{v,2}}, \quad B = \frac{e_{*,1} - e_{*,2}}{c_{p,2} - c_{v,2}}, \quad C = \frac{c_{p,2} - c_{p,1}}{c_{p,2} - c_{v,2}}, \quad D = \frac{c_{p,1} - c_{v,1}}{c_{p,2} - c_{v,2}},$$

in which $e'_{*,i}$ is a reference entropy parameter of the stiffened gas EOS for species i. These strategies are compatible with five- and six-equation models and are solved via Newton's method. Once the solution is obtained, the solver advances to the next time step.

To demonstrate the model capabilities, we compare numerical results to a set of shock tube experiments by Simões-Moreira and Shepherd [41] ('chocked flow series'), focusing on the $pT\mu$-relaxation: metastable liquid n-dodecane on the left half of the tube is suddenly discharged into an almost-vacuum region of 1×10^{-2} Pa, depressurizing it and initiating phase change. Consequently, an evaporation wave propagates into the liquid with a steady front mean velocity U_F, which depends on the approximately constant temperature used for each run, with $T \in [180, 300]$ °C. U_F is the ratio of the differences in momentum and density [42] after ("a") and before ("b") the evaporation wave as

$$U_F = \frac{(\rho u)_a - (\rho u)_b}{\rho_a - \rho_b}.$$

Fig. 3 compares the numerical (5- and 6-equation models) and experimental results. We observe that both numerical results match well, with some minor discrepancies noted. These are suspected to be due to the sensitivity in the data obtained to calculate U_F, as the precise locations of "a" and "b" are unclear. The numerical results quantitatively match the experimental data, with closer matching at larger T.

4.1.4. Euler–Euler sub-grid bubble dynamics

We use Euler–Euler sub-grid models for bubble dynamics, including the method of classes and the method of moments. The method of classes is based on the ensemble phase-averaged equations [43], which can be used to represent dispersions of radially oscillating bubbles as well as other particles [44–47]. Note that the method of classes approach was included in MFC 3.0 [1], though a statistical representation enabled via the moment method was not included, which we focus on here.

The void fraction of the dispersed phase α is assumed to be negligible compared to the liquid phase (α_l) under dilute assumptions. The bubbles are represented statistically via random variables R, $\dot{R}$, and R_o, corresponding to the instantaneous bubble radius, time derivative, and equilibrium bubble radius. The mixture-averaged pressure field is

$$p(\boldsymbol{x}, t) = (1 - \alpha) p_\ell + \alpha \left(\frac{\overline{R^3 p_{bw}}}{\overline{R^3}} - \rho \frac{\overline{R^3 \dot{R}^2}}{\overline{R^3}} \right),$$

where p_{bw} is the associated bubble wall pressure and $p_\ell(\boldsymbol{x}, t)$ is the liquid pressure according to the modified stiffened-gas EOS [48]:

$$\Gamma_\ell p_\ell + \Pi_{\infty,\ell} = \frac{1}{1 - \alpha} \left(E - \frac{1}{2} \rho \boldsymbol{u}^2 \right).$$

The bubble pressure p_b follows the polytropic assumption, and the bubble wall pressure p_{bw} is

$$p_{bw} = p_o \left(\frac{R_o}{R} \right)^{3\gamma} - \frac{4\mu \dot{R}}{R} - \frac{2\sigma}{R}.$$

The bubble number density per unit volume $n_{\text{bub.}}(\boldsymbol{x}, t)$ is conserved as

$$\frac{\partial n_{\text{bub.}}}{\partial t} + \nabla \cdot (n_{\text{bub.}} \boldsymbol{u}) = 0.$$

For the spherical bubbles considered here, $n_{\text{bub.}}$ is related to the void fraction α via

$$\alpha(\boldsymbol{x}, t) = \frac{4}{3} \pi \overline{R^3} n_{\text{bub.}}(\boldsymbol{x}, t),$$

and so the void fraction $\alpha(\boldsymbol{x}, t)$ transports as

$$\frac{\partial \alpha}{\partial t} + \boldsymbol{u} \cdot \nabla \alpha = 3\alpha \frac{\overline{R^2 \dot{R}}}{\overline{R^3}},$$

The right-hand side represents the change in void fraction associated with bubble growth and collapse. The over-barred terms appearing in the above equations denote average quantities of the bubble dispersion. These are averaged over N_{bin} bins of R_o using a log-normal probability distribution function (pdf). Hybrid schemes and fast integration techniques have been developed for polydisperse cases that otherwise require large bin counts [49,50]. We implement the Rayleigh–Plesset and Keller–Miksis equations [51] for the radial bubble dynamics. The Keller–Miksis dynamics are

$$R\ddot{R}\left(1 - \frac{\dot{R}}{c}\right) + \frac{3}{2}\dot{R}^2\left(1 - \frac{\dot{R}}{3c}\right) = \frac{p_{\text{bw}} - p_l}{\rho_l}\left(1 + \frac{\dot{R}}{c}\right) + \frac{R\dot{p}_{\text{bw}}}{\rho c}.$$

The method of classes incorporates stochasticity in R_o using a log-normal probability density function (pdf). However, modeling complex bubbly flow phenomena requires the expansion of the set of stochastic variables, including instantaneous bubble variables. The inaccuracy of the polytropic assumption can also limit its scope of application. The method of moments is employed for this purpose, utilizing a population balance equation (PBE) to govern the probability density function (pdf) of the bubbles. The pdf $f(R, \dot{R}, R_o)$ governs the moments μ_{lmn} as

$$\mu_{lmn} = \overline{R^l \dot{R}^m R_o^n} = \int_\Omega R^l \dot{R}^m R_o^n f(R, \dot{R}, R_o)\,\mathrm{d}R\mathrm{d}\dot{R}\mathrm{d}R_o,$$

The moments are evaluated using a conditional inversion procedure that follows Fox et al. [52]. The conditional probability density function $f(R, \dot{R}|R_o)$ without coalescence or breakup effects is governed by the PBE

$$\frac{\partial f}{\partial t} + \frac{\partial}{\partial R}(f\dot{R}) + \frac{\partial}{\partial \dot{R}}(f\ddot{R}) = 0.$$

The evolution of the raw moments $\vec{\mu}$ is obtained by integrating the population balance equation. The set of transport equations for $\vec{\mu}$ are

$$\frac{\partial n_{\mathrm{bub}}\mu_i}{\partial t} + \nabla\cdot(n_{\mathrm{bub}}\mu_i \boldsymbol{u}) = n_{\mathrm{bub}}\dot{\mu}_i = n_{\mathrm{bub}} g_i, \tag{11}$$

which are enumerated over each moment and corresponding right-hand side of the column vectors $\vec{\mu}$ and $\boldsymbol{g}$, denoted as μ_i and g_{lmn}. So,

$$g_{lmn} = l\mu_{l-1,m+1,n} + m \iiint_\Omega \ddot{R} R^l \dot{R}^{m-1} R_o^n f(\vec{\mu})\,\mathrm{d}R\mathrm{d}\dot{R}\mathrm{d}R_o,$$

and $\Omega = \Omega_R \times \Omega_{\dot{R}} \times \Omega_{R_o} = (0,\infty)\times(-\infty,\infty)\times(0,\infty)$ [46]. The integral is computed via quadrature nodes obtained by the inversion procedure.

The pdf is conditioned on R_o as

$$f(R, \dot{R}, R_o) = f(R, \dot{R}|R_o) f(R_o),$$

and the raw moments are

$$\mu_{lmn} \equiv \int_{\Omega_{R_o}} f(R_o) R_o^m \mu_{lm}(R_o)\,\mathrm{d}R_o \approx \sum_{i=1}^{N_{R_o}} w_i \hat{R}_{o,i}^n \mu_{lm}(\hat{R}_{o,i}),$$

where N_{R_o} represents the polydispersity of the bubble population. The conditional moments μ_{lm} are evaluated using a conditional hyperbolic inversion procedure (CHyQMOM). The CHyQMOM algorithm is described in Fox et al. [52], and a detailed discussion of its application to bubble dynamics can be found in Bryngelson et al. [45]. The total moments follow as

$$\mu_{lmn} = \sum_{i=1}^{N_{R_o}} w_i \hat{R}_{o,i}^n \sum_{j=1}^{N_R}\sum_{k=1}^{N_{\dot{R}}} \left[\hat{w}_{j,k} \hat{R}_j^l \hat{\dot{R}}_k^m\right]_{\hat{R}_{o,i}},$$

These moments evaluate the right-hand side of the moment evolution equation and the ensemble-phase averaged terms. The evaluation of $\overline{R^3 p_{bw}}$ requires the accounting of statistics of p_b. Without polytropic assumptions, this requires including additional internal variables in the PBE. The ODEs for p_b and m_v are initialized using an isothermal assumption and evolved at the quadrature nodes, keeping computation costs low. This strategy assumes their probability densities to be Dirac delta functions centered at the quadrature nodes for R and $\dot{R}$. The ODEs for p_b and m_v follow Ando et al. [53] as

$$\dot{p}_b = \frac{3\gamma_b}{R}\left(-\dot{R}p_b + R_v T_{bw}\dot{m}_v + \frac{\gamma_b - 1}{\gamma_b} k_{bw}\frac{\partial T}{\partial r}|_{r=R}\right) \quad \text{and}$$

$$\dot{m}_v = \frac{D\rho_{bw}}{1-\chi_{vw}}\frac{\partial \chi_{vw}}{\partial r}|_{r=R}.$$

4.1.5. Euler–Lagrange sub-grid bubble dynamics

The Euler–Lagrange model for sub-grid bubble dynamics is based on the volume-averaged equations of motion and describes the dynamics of a mixture of dispersed bubbles in a compressible liquid. Mixture properties $\overline{(\cdot)}$ are defined as $\overline{(\cdot)} = (1-\alpha)(\cdot)_l + \alpha(\cdot)_g$, where α is the volume fraction of the gas or void fraction, and the subscripts l and g denote the liquid and gas phase, respectively. Assuming zero slip-velocity between the liquid and gas phase and applying the volume averaging, yields the following set of inhomogeneous equations

$$\begin{aligned}
\frac{\partial \rho_l}{\partial t} + \nabla\cdot(\rho_l \boldsymbol{u}_l) &= \frac{\rho_l}{1-\alpha}\left[\frac{\partial \alpha}{\partial t} + \boldsymbol{u}_l\cdot\nabla\alpha\right],\\
\frac{\partial(\rho_l \boldsymbol{u}_l)}{\partial t} + \nabla\cdot(\rho_l \boldsymbol{u}_l\otimes\boldsymbol{u}_l + p\mathcal{I} - \mathcal{T}_l) &= \frac{\rho_l \boldsymbol{u}_l}{1-\alpha}\left[\frac{\partial \alpha}{\partial t} + \boldsymbol{u}_l\cdot\nabla\alpha\right] - \frac{\alpha\nabla\cdot(p\mathcal{I} - \mathcal{T}_l)}{1-\alpha},\\
\frac{\partial E_l}{\partial t} + \nabla\cdot((E_l + p)\boldsymbol{u}_l - \mathcal{T}_l\cdot\boldsymbol{u}_l) &= \frac{E_l}{1-\alpha}\left[\frac{\partial \alpha}{\partial t} + \boldsymbol{u}_l\cdot\nabla\alpha\right] - \frac{\alpha\nabla\cdot(p\boldsymbol{u}_l - \mathcal{T}_l\cdot\boldsymbol{u}_l)}{1-\alpha}.
\end{aligned} \tag{12}$$

Here, $\mathcal{T}_l$ is the viscous stress tensor of the liquid host. The advantage of the above form of equations is that the left-hand side of the equation can be seen as the conservation equations for the liquid phase (the details of which are explained in Section 2.1.1), and the right-hand side part can be seen as source terms that carry the effect of the bubbles in the host liquid.

The volumetric oscillations of the sub-grid bubbles due to pressure variations in the surrounding medium are modeled using the Keller–Miksis equation of Section 4.1.4. The Lagrangian field equation is integrated using a fourth-order accurate Runge–Kutta scheme with adaptive time stepping.

MFC communicates the bubble sizes to the carrier-flow solver via the local void fraction α. This coupling is achieved by computing an effective void fraction derived from the contribution of each bubble to its surrounding computational cells, which is illustrated in Fig. 4.

The bubble volume is smeared in the continuous field of the void fraction in the mixture at its coordinate $\boldsymbol{x}_n$ using a regularization kernel δ as

$$\alpha(\boldsymbol{x}) = \sum_{n=1}^{N_{\mathrm{bub}}} V_n \delta = \left(\frac{4}{3}\pi R_n^3\right)\delta, \tag{13}$$

where N_{bub} is the number of bubbles, and V_n is the volume of bubble n. We use the continuous, second-order, truncated Gaussian function for the kernel. With α, all the terms on the right-hand side of Eq. (12) can be computed. More details on the numerical implementation can be found in Maeda and Colonius [9] and Fuster and Colonius [54].

The Euler–Lagrange model is validated against the analytical solution of the Keller–Miksis equation reported by Maeda and Colonius [9]. In this test, we consider a single gas bubble suspended in a water tank. The bubble has an initial radius of $50\,\mu\mathrm{m}$ and is positioned at the center of the fluid domain $(0,0,0)$. It is subjected to a planar sinusoidal acoustic wave with an amplitude of $0.2\,\mathrm{MPa}$ and a frequency of $150\,\mathrm{kHz}$. To prevent acoustic reflections at the domain boundaries, we use the non-reflective boundary conditions. The temporal evolution of the bubble radius, shown in Fig. 5, is compared with the analytical Keller–Miksis solution. The results exhibit excellent agreement, yielding an RMSE error of $2.76\,\%$.

To evaluate the Euler–Euler subgrid model, we simulate the interaction between a dilute bubble screen and a planar sinusoidal acoustic wave. The results from the Euler–Euler formulation are compared directly with those obtained using our Euler–Lagrange model. The computational domain spans $x \in [-20\,\mathrm{mm}, 20\,\mathrm{mm}]$ and $y, z \in [-2.5\,\mathrm{mm}, 2.5\,\mathrm{mm}]$, with a bubble cloud centered at the origin and an initial void fraction of $\alpha_0 = 4\times10^{-5}$. In the monodisperse case, all bubbles are initialized with an identical radius of $10\,\mu\mathrm{m}$. In contrast, in the polydisperse case, the bubble radii follow a log-normal distribution centered at $10\,\mu\mathrm{m}$ with a shape parameter of $\sigma_p = .3$. A single $0.1\,\mathrm{MPa}$, $300\,\mathrm{kHz}$ acoustic wave is introduced at $x = -7.5\,\mathrm{mm}$ and propagates in the positive x-direction. Simulations use a uniform 3D grid ($400\times50\times50$) with $100\,\mu\mathrm{m}$ spacing, providing fifty cells per wavelength. For the volume-averaged model, 40 realizations are performed to ensure statistical reliability, while the ensemble-averaged model resolves 21 bins in bubble size space. Fig. 6 shows that both models produce closely matching pressure responses at the cloud center, with RMSE values below $2.1\,\%$. The shaded region represents the variability range observed across all Euler–Lagrange realizations.

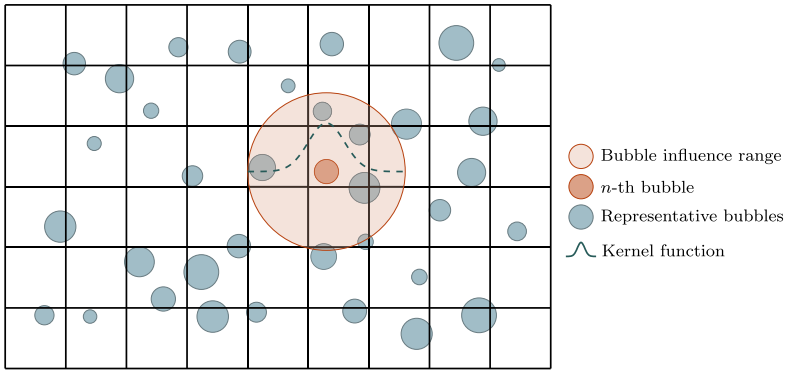


Fig. 4. Volume spreading of the nth bubble for the void fraction computations using a Gaussian kernel of characteristic radial extent (not to scale).

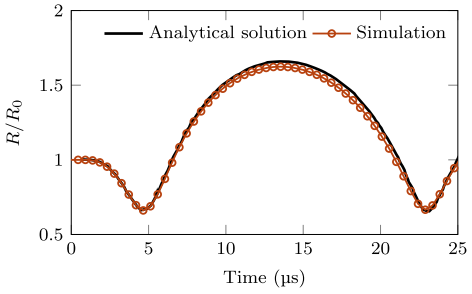


Fig. 5. Evolution of an isolated bubble in response to a single cycle of a sinusoidal pressure wave. The analytical solution of the Keller–Miksis equation was reported by Maeda and Colonius [9].

4.1.6. Fluid–elastic structure interaction

Fluid–structure interaction with hypoelastic materials is modeled by adding the elastic shear stress $\boldsymbol{T}^e$ to the Cauchy stress tensor of the five-equation and six-equation models described in Sections 2.1.1 and 2.1.2, respectively. The model relies on transforming strains to strain rates using the Lie objective temporal derivative [55,56]. The strain rates are computed using high-order accurate central finite differences. The mixture energy is also updated to include the elastic contribution, which gives

$$E = e + \frac{\|\boldsymbol{u}\|^2}{2} + \frac{\boldsymbol{T}^e : \boldsymbol{T}^e}{4\rho G},$$

where G is the shear modulus of the medium. The third term on the right-hand side is the hypoelastic energy term. The Lie objective temporal derivative of the elastic stresses

$$\mathring{\boldsymbol{T}}^e = \frac{D\boldsymbol{T}^e}{Dt} - \boldsymbol{l}\cdot\boldsymbol{T}^e - \boldsymbol{T}^e\cdot\boldsymbol{l}^\top + \boldsymbol{T}^e\mathrm{tr}(\boldsymbol{D}),$$

where $\boldsymbol{l}$ is the velocity gradient, and for an isotropic Kelvin–Voigt material $\mathring{\boldsymbol{T}}^e = 2G\boldsymbol{D}^d$ [57] are used to derive a time evolution equation for the elastic stresses. Here, the deviatoric component of the velocity gradient is $\boldsymbol{D}^d = \boldsymbol{D} - \mathrm{tr}(\boldsymbol{D})\boldsymbol{I}/3$. Appending this evolution equation to the five-equation model gives the five-equation model with hypoelasticity

$$\begin{aligned}
\frac{\partial \alpha_i\rho_i}{\partial t} + \nabla\cdot(\alpha_i\rho_i\boldsymbol{u}) &= 0,\\
\frac{\partial \rho\boldsymbol{u}}{\partial t} + \nabla\cdot(\rho\boldsymbol{u}\otimes\boldsymbol{u} + p\boldsymbol{I} - \boldsymbol{T}^e) &= \nabla\cdot\boldsymbol{T}^v,\\
\frac{\partial \rho E}{\partial t} + \nabla\cdot\left[(\rho E + p)\boldsymbol{u} - \boldsymbol{T}^e\cdot\boldsymbol{u}\right] &= \nabla\cdot(\boldsymbol{T}^v\cdot\boldsymbol{u}),\\
\frac{\partial \alpha_i}{\partial t} + \boldsymbol{u}\cdot\nabla\alpha_i &= K\nabla\cdot\boldsymbol{u},\\
\frac{\partial(\rho\boldsymbol{T}^e)}{\partial t} + \nabla\cdot(\rho\boldsymbol{T}^e\otimes\boldsymbol{u}) &= \boldsymbol{S}^e,
\end{aligned}$$

where $\boldsymbol{S}^e$ is the elastic source term:

$$\boldsymbol{S}^e = \rho\left(\boldsymbol{l}\cdot\boldsymbol{T}^e + \boldsymbol{T}^e\cdot\boldsymbol{l}^\top - \boldsymbol{T}^e\mathrm{tr}(\boldsymbol{D}) + 2G\boldsymbol{D}^d\right).$$

Details of the hypoelastic model implementation can be found in Spratt [58].

Fluid–structure interaction with hyperelastic materials is modeled with an evolution equation for the reference map using the reference map technique (RMT) of [59]. The gradient of the reference map $\boldsymbol{\xi}$, the inverse of the deformation gradient $\boldsymbol{F}$,

$$\boldsymbol{F} = (\nabla\boldsymbol{\xi})^{-1},$$

is computed using a high-order central finite-difference scheme. The evolution equation of the reference map is combined with the conservation of mass equation to obtain a conservative form,

$$\frac{\partial(\rho\boldsymbol{\xi})}{\partial t} + \nabla\cdot(\rho\boldsymbol{\xi}\otimes\boldsymbol{u}) = \boldsymbol{0},$$

and solved with the system of equations in Sections 2.1.1 and 2.1.2. The deformation gradient $\boldsymbol{F}$, the left Cauchy–Green strain $\boldsymbol{b}$,

$$\boldsymbol{b} = \boldsymbol{F}\boldsymbol{F}^\top,$$

and the elastic deviatoric contribution to the Cauchy stress tensor $\boldsymbol{T}^e$ is computed. For a compressible neoHookean material model, the elastic deviatoric part of the Cauchy stress tensor is

$$\boldsymbol{T}^e = \frac{G}{J}\left(\boldsymbol{b} - \frac{1}{3}\mathrm{tr}(\boldsymbol{b})\boldsymbol{I}\right),$$

where J is the determinant of $\boldsymbol{b}$ and $\boldsymbol{I}$ is the identity tensor and the hyperelastic energy is $e^e = G(I_b - 3)/2$, where I_b is the first invariant of $\boldsymbol{b}$. Further details of the hyperelastic model implementation can be found in Barbosa et al. [60].

4.1.7. Chemical reactions and combustion

High-fidelity simulations of reacting flows are critical to designing efficient propulsion systems. Such simulations require models that account for the effects of chemical reactions on the flow. There are two major differences with respect to simulations of inert flow. First, the mixture composition varies locally due to chemical reactions, and as a result, fluid properties, such as viscosity, also change. This effect must be accounted for to achieve acceptable accuracy; it is typically accomplished using transport equations for individual species. Second, because chemical reactions release heat, temperature changes can become substantial; therefore, variable thermodynamic properties are necessary for accuracy. As we expand on this, we incorporate these effects through detailed models in the gas phase.

The gas phase is a mixture of N_{species} species with mass fractions $\{Y_m\}_{m=1}^{N_{\text{species}}}$. Without molecular transport, these evolve as

$$\frac{\partial \rho_g Y_m}{\partial t} + \frac{\partial \rho_g u_i Y_m}{\partial x_i} = W_{(m)}\dot{\omega}_m, \quad m = 1,\ldots,N_{\text{species}},$$

where ρ_g is the density of the gas phase, W_m is the molecular weight of the m^{th} species (with parenthesized subscript obviating Einstein summation) and $\dot{\omega}_m$ its net production rate by chemical reactions—the chemical source term.

Next, we provide detailed expressions for the chemical source term. Their implementation uses code generation and is discussed in Section 4.3.7. We simulate gas-phase combustion by integrating Section 4.1.7 along with the equations for conservation of momentum and

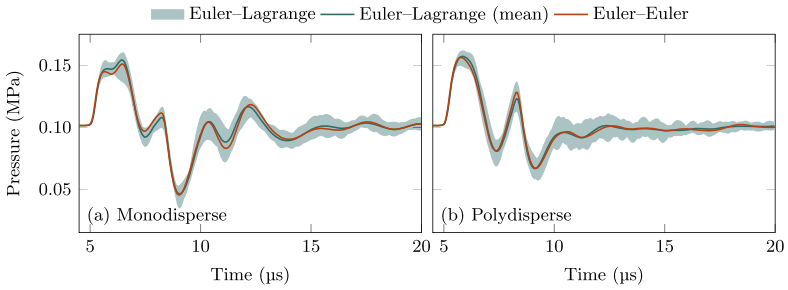


Fig. 6. Pressure profiles at the origin of the bubble screen using the Euler–Lagrange and Euler–Euler subgrid models. Mono and polydisperse (log-normally distributed with $\sigma_p = .3$) bubble clouds are tested as labeled.

total energy density (1). Plans to extend this capability to multi-phase combustion with liquid or solid fuels entail the implementation of detailed expressions for phase changes [61–63].

The net production rate in Section 4.1.7 is

$$\dot{\omega}_m = \sum_{n=1}^{N_{\text{species}}} (\nu''_{mn} - \nu'_{mn}) R_n, \quad m = 1, \ldots, N_{\text{species}}, \tag{14}$$

where ν'_{mn}, ν''_{mn} are the forward and reverse stoichiometry of the m^{th} species in the n^{th} reaction, and R_n the net reaction rate of progress. By the law of mass action,

$$R_n = k_n(T)\left[\prod_{j=1}^{N}\left(\frac{\rho_g Y_j}{W_j}\right)^{\nu'_{jn}} - \frac{1}{K_n(T)}\prod_{k=1}^{N}\left(\frac{\rho_g Y_k}{W_k}\right)^{\nu''_{kn}}\right] \tag{15}$$

where k_n and K_n are the rate coefficient and the equilibrium constant. Expressions for the rate coefficient are reaction-dependent but conventionally take a modified Arrhenius form

$$k_n(T) = A_n T^{b_n} \exp(-T_{a,n}/T), \tag{16}$$

where A_n, b_n, $T_{a,n}$ are the pre-exponential factor, temperature exponent, and activation temperature of the n^{th} reaction. The equilibrium constant in Eq. (15) is determined from standard equilibrium thermodynamics; detailed expressions can be found elsewhere [64,65].

As stated, one must consider the effect of the mixture thermodynamics on combustion. For the equation of state, we assume a mixture of ideal gases, so

$$p = \rho R_u T / W, \tag{17}$$

where R_u is the universal gas constant and

$$W = \left(\sum_{m=1}^{N_{\text{species}}} \frac{Y_m}{W_m}\right)^{-1} \tag{18}$$

is the mixture molecular weight. The temperature follows from

$$e_g - \sum_{m=1}^{N_{\text{species}}} e_m(T)\, Y_m = 0, \tag{19}$$

where e_g is the gas-phase internal energy per unit mass (obtained from (1)), and $\{e_m(T)\}_{m=1}^{N_{\text{species}}}$ the species internal energies. These are

$$e_m(T) = \frac{\hat{h}_m(T) - R_u T}{W_m}, \qquad m = 1, \ldots, N_{\text{species}}, \tag{20}$$

where $\{\hat{h}_m(T)\}_{m=1}^{N_{\text{species}}}$ are the species molar enthalpies, modeled using NASA polynomials [66]:

$$\frac{\hat{h}_m}{R_u T} = \frac{\hat{C}_0}{T} + \sum_{r=1}^{5} \frac{\hat{C}_r}{r} T^{r-1}, \tag{21}$$

with fit coefficients $\{\hat{C}\}_{r=0}^{5}$ determined from experiments and collision integrals [67]. The fitting procedure for Eq. (21) includes heats of formation, so, in concert with Eqs. (19) and (20), it accounts for combustion heat release. In practice, to obtain the temperature, Eq. (19) is solved iteratively using a standard Newton method.

The order accuracy is verified via a test problem: the advection of a two-species, calorically perfect mixture. The flow is initialized with a sinusoidal density and temperature profile, as well as spatially uniform species mass fractions. The errors compared to a reference solution are in Fig. 7, which shows the expected order of accuracy for all numerical methods.

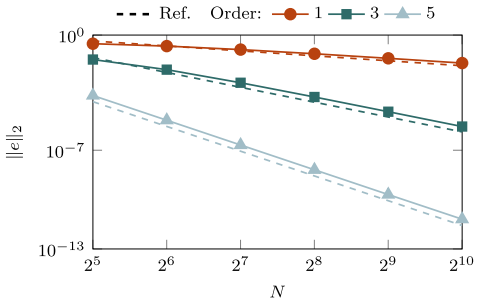


Fig. 7. Convergence results for a 1D 2-component advection problem with constant specific heats.

To validate our implementation, we simulate a one-dimensional reacting shock tube and reproduce the published results of Martínez-Ferrer et al. [68]. We uniformly discretize the domain of size $L = 12\,\text{cm}$ with 400 grid cells. The left boundary represents a reflecting wall, while the right boundary is an outflow. Reactions are modeled using the San Diego mechanism [69]. The initial shock is left-going, reflects from the wall, and ignites the mixture. The combustion wave and the shock coalesce into a detonation wave. The comparison between our implementation and the results of Martínez-Ferrer et al. [68] is shown in Fig. 8. The agreement is sufficient for validation, considering the uncertainties associated with different numerical methods and combustion mechanisms.

4.1.8. *Surface tension at diffuse material interfaces*

Surface tension for two-fluid flows is implemented using the model of Schmidmayer et al. [70]. An advection equation for the color function c, which is 0 in one fluid and 1 in the other, is added to the system of equations. This color function is then used to calculate the capillary stress tensor

$$\boldsymbol{\Omega} = -\sigma\left(\|\nabla c\|\boldsymbol{I} - \frac{\nabla c \otimes \nabla c}{\|\nabla c\|}\right),$$

where σ is the surface tension coefficient. The capillary stress tensor defines the contribution of surface tension to the momentum and mixture energy equations. The six-equation model with surface tension is

$$\begin{aligned}
&\frac{\partial \alpha_i \rho_i}{\partial t} + \nabla \cdot (\alpha_i \rho_i \boldsymbol{u}) = 0,\\
&\frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u} + p\boldsymbol{I} + \boldsymbol{\Omega} - \boldsymbol{T}^v) = 0,\\
&\frac{\partial \alpha_1 \rho_1 e_1}{\partial t} + \nabla \cdot (\alpha_1 \rho_1 e_1 \boldsymbol{u}) + \alpha_1 p_1 \cdot \nabla \boldsymbol{u} = -\mu p_I (p_2 - p_1) - \alpha_1 \boldsymbol{T}_1^v : \nabla \boldsymbol{u},\\
&\frac{\partial \alpha_2 \rho_2 e_2}{\partial t} + \nabla \cdot (\alpha_1 \rho_2 e_2 \boldsymbol{u}) + \alpha_2 p_2 \cdot \nabla \boldsymbol{u} = -\mu p_I (p_1 - p_2) - \alpha_2 \boldsymbol{T}_2^v : \nabla \boldsymbol{u},\\
&\frac{\partial (\rho E + \varepsilon_0)}{\partial t} + \nabla \cdot \left((\rho E + \varepsilon_0 + P)\boldsymbol{u} + (\boldsymbol{\Omega} - \boldsymbol{T}^v) \cdot \boldsymbol{u}\right) = 0,
\end{aligned}$$

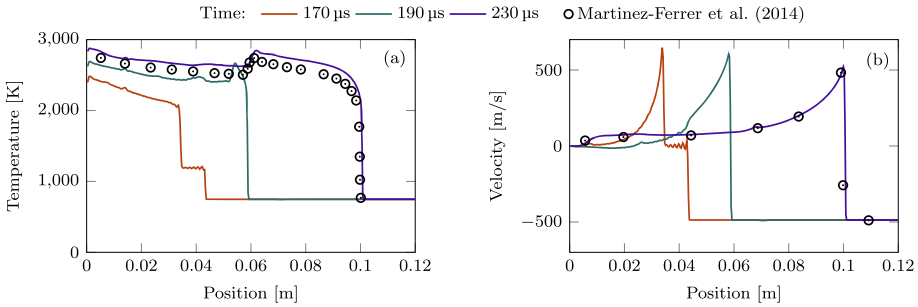


Fig. 8. (a) Temperature and (b) velocity profiles of a one-dimensional reactive shock tube compared to the results of Martínez-Ferrer et al. [68].

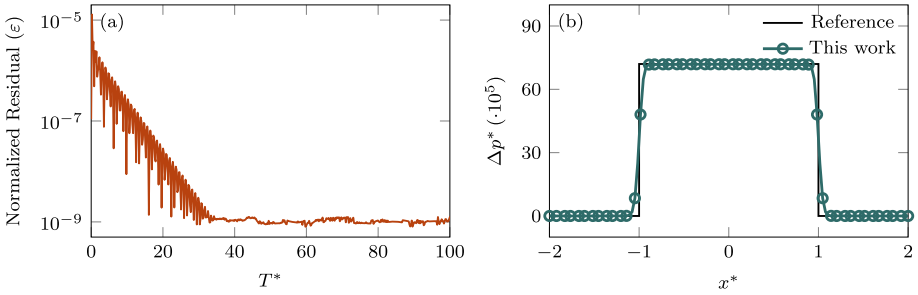


Fig. 9. Validation results for the surface tension implementation using a Laplace pressure jump problem with a 128×128 grid. (a) shows the normalized residual as a function of time; (b) shows the pressure along the centerline for an $R^* = 1$ water column with a surface tension coefficient $\sigma^* = 7.2 \times 10^{-4}$.

$$\frac{\partial \alpha_1}{\partial t} + \boldsymbol{u} \cdot \nabla \alpha_1 = \mu(p_1 - p_2),$$
$$\frac{\partial c}{\partial t} + \boldsymbol{u} \cdot \nabla c = 0,$$

where ε_0 is the capillary mixture energy. The six-equation model with surface tension is conservative, except for the pressure relaxation terms, and follows the second law of thermodynamics. The implementation of surface tension is validated by resolving the Laplace pressure jump inside a liquid water column in air. The theoretical pressure jump in two dimensions is given by $\Delta P = \sigma / R_\mathrm{d}$ where R_d is the radius of the water column. Fig. 9 shows the normalized pressure residual

$$\varepsilon = \max\left[\frac{|P_{i,j}^N - P_{i,j}^{N-1}|}{P_{i,j}^N}\right]$$

as a function of time and the pressure across the centerline for an $R^* = 1$ water column with a surface tension coefficient $\sigma^* = 7.2 \times 10^{-4}$ for a 128×128 grid. The pressure jump is resolved exactly, and the residual is time-stable.

4.1.9. Magnetohydrodynamics

Magnetohydrodynamics (MHD) governs the behavior of conducting fluids under the influence of magnetic fields. The governing equations for ideal MHD are [71]:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0,$$
$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot \left[\rho \mathbf{u} \otimes \mathbf{u} + \left(p + \frac{1}{2}|\mathbf{B}|^2\right)\mathbf{I} - \mathbf{B} \otimes \mathbf{B}\right] = 0,$$
$$\frac{\partial E}{\partial t} + \nabla \cdot \left[\left(E + p + \frac{1}{2}|\mathbf{B}|^2\right)\mathbf{u} - (\mathbf{u} \cdot \mathbf{B})\mathbf{B}\right] = 0,$$
$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \cdot (\mathbf{u} \otimes \mathbf{B} - \mathbf{B} \otimes \mathbf{u}) = 0.$$

Here, p is the thermal pressure and $\mathbf{B}$ is the magnetic field. The total energy density is $E = \rho e + (\rho|\mathbf{u}|^2 + |\mathbf{B}|^2)/2$.

The fast magnetosonic speed, which represents the maximum wave propagation speed, is

$$c_f = \sqrt{\frac{1}{2}\left(c_s^2 + v_A^2 + \sqrt{\left(c_s^2 + v_A^2\right)^2 - 4c_s^2 v_A^2 \cos^2\theta}\right)},$$

where c_s is the sound speed and $v_A = \sqrt{|\mathbf{B}|^2/\rho}$ is the Alfvén speed.

A one-dimensional Brio–Wu MHD shock-tube validation of this implementation is shown in Fig. 10 (a), which closely matches the reference [72]. The convergence test in Fig. 11 also shows the expected order of accuracy for the smooth Alfvén-wave problem [73], confirming the robustness of the implementation.

4.1.10. Relativistic magnetohydrodynamics

Relativistic magnetohydrodynamics (RMHD) extends the classical MHD approach to high-energy density plasmas where fluid velocities approach the speed of light. The governing equations can be expressed in conservative form using the conserved variable vector:

$$\mathbf{U} = \begin{pmatrix} D \\ \mathbf{m} \\ \tau \\ \mathbf{B} \end{pmatrix} = \begin{pmatrix} \Gamma\rho \\ \Gamma^2 \rho h \mathbf{u} + |\mathbf{B}|^2 \mathbf{u} - (\mathbf{u} \cdot \mathbf{B})\mathbf{B} \\ \Gamma^2 \rho h - p + \left[|\mathbf{B}|^2 + |\mathbf{u}|^2|\mathbf{B}|^2 - (\mathbf{B} \cdot \mathbf{u})^2\right]/2 - \Gamma\rho \\ \mathbf{B} \end{pmatrix}.$$

Here, ρ denotes the rest-mass density, $\mathbf{u}$ is the spatial 3-velocity, Γ is the Lorentz factor, p is the thermal pressure, and $h = 1 + e + p/\rho$ is the specific enthalpy. We adopt natural units, where the speed of light is c and defined by reference as $c = 1$. The rest-mass contribution is subtracted from the total energy density to reduce numerical errors in the non-relativistic limit.

The governing equations are [74]:

$$\frac{\partial D}{\partial t} + \nabla \cdot (D\mathbf{u}) = 0,$$
$$\frac{\partial \mathbf{m}}{\partial t} + \nabla \cdot \left[\mathbf{m} \otimes \mathbf{u} + \left(p + \frac{1}{2}\left(\frac{|\mathbf{B}|^2}{\Gamma^2} + (\mathbf{u} \cdot \mathbf{B})^2\right)\right)\mathbf{I} - \frac{\mathbf{B} \otimes \mathbf{B}}{\Gamma^2} - (\mathbf{u} \cdot \mathbf{B})\mathbf{u} \otimes \mathbf{B}\right] = 0,$$
$$\frac{\partial \tau}{\partial t} + \nabla \cdot (\mathbf{m} - D\mathbf{u}) = 0,$$
$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \cdot (\mathbf{u} \otimes \mathbf{B} - \mathbf{B} \otimes \mathbf{u}) = 0.$$

Recovery of the primitive variables from these highly nonlinear conservative variables requires an iterative scheme. In this work, we solve a single nonlinear equation using the Newton–Raphson method [74].

To validate the implementation, we consider a relativistic one-dimensional RMHD shock-tube problem based on Mignone and Bodo [74]; the resulting density profile in Fig. 10 (b) aligns closely with the reference, as do the other variables (not shown).

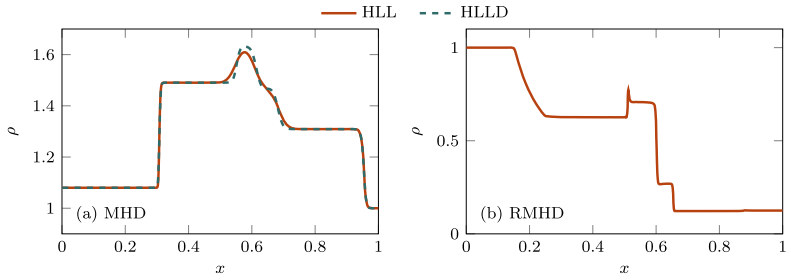


Fig. 10. Density profiles for (a) ideal MHD and (b) RMHD shock-tube problems. Panel (a) follows Miyoshi and Kusano [72] (their Fig. 5), and both solvers align closely with the reference solution. Panel (b) follows Mignone and Bodo [74] (their Fig. 3), and shows similarly tight agreement; the RMHD case here uses WENO3 rather than the reference's second-order scheme. Only density is shown; other variables behave comparably.

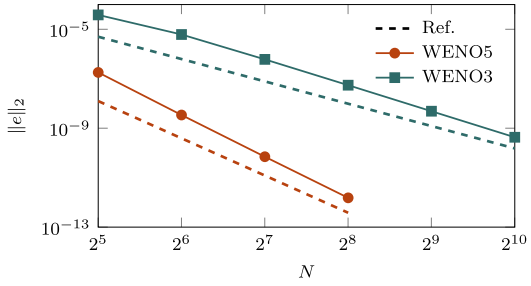


Fig. 11. Convergence test for the smooth Alfvén-wave problem [73] using WENO3 and WENO5, showing errors decreasing at the expected rate.

4.2. Numerical methods

4.2.1. General characteristic boundary conditions

The priority for time-dependent boundary conditions for hyperbolic equations is to ensure a smooth egress of the outgoing artifacts with minimal impact on the interior solution. For this purpose, the non-reflecting characteristic boundary conditions [33] are used, which perform a characteristic decomposition of the governing equation at the boundary. Ignoring the contribution of the viscous and transverse terms, this is

$$\frac{\partial q_c}{\partial t} + R_x \Lambda_x L_x \frac{\partial q_c}{\partial x} = 0,$$

where q_c is the set of conservative variables, $\Lambda_x = \mathrm{diag}(\lambda_j) = \mathrm{diag}(u-c, u, u, u, u+c)$ are the eigenvalues of the Jacobian, and R_x and L_x are the right and left eigenvectors. The contribution of the characteristics can be further decomposed into incoming and outgoing waves, depending on the sign of the eigenvalue, as

$$\lambda_j^i = \lambda_j - n_x|\lambda_j| \quad \text{and} \quad \lambda_j^o = \lambda_j + n_x|\lambda_j|,$$

with $n_x = 1$ for the right boundary and $n_x = -1$ for the left boundary. The non-reflecting characteristic boundary conditions subsequently ignore the contribution of the incoming wave to get

$$\frac{\partial q_c}{\partial t} = -R_x \Lambda_x^o L_x \frac{\partial q_c}{\partial x}.$$

In many applications, the primitive variables q_p outside the computational domain are not known, necessitating a more general approach. The evolution equation for the conservative variables at the boundary can be cast into primitive form as

$$\frac{\partial q_c}{\partial t} = -R_x \Lambda_x L_x \frac{\partial q_c}{\partial x} = -R_x \Lambda_x L_x P \frac{\partial q_p}{\partial x},$$

where $P = \partial_{q_p} q_c$ is the conversion Jacobian matrix. The characteristics $\mathcal{L} = \Lambda_x L_x P \partial_x q_p$ are

$$\mathcal{L}_1 = (u-c)\left(\frac{\partial p}{\partial x} - \rho c \frac{\partial u}{\partial x}\right), \quad \mathcal{L}_2 = u\left(c^2 \frac{\partial \rho}{\partial x} - \frac{\partial p}{\partial x}\right),$$

$$\mathcal{L}_3 = u \frac{\partial v}{\partial x}, \quad \mathcal{L}_4 = u \frac{\partial w}{\partial x}, \quad \mathcal{L}_5 = (u+c)\left(\frac{\partial p}{\partial x} + \rho c \frac{\partial u}{\partial x}\right),$$

with v and w being the transverse velocities. The generalized characteristic boundary conditions make use of the solution outside the domain at ghost points ($q_p^{(g)}$) to incorporate the contribution of the incoming characteristics [75]. The characteristics $\mathcal{L}$ in this case are

$$\mathcal{L} = -\Lambda_x^o L_x \frac{\partial q_c}{\partial x} - \Lambda_x^i L_x \frac{\partial q_c}{\partial x} = -\Lambda_x^o L_x \frac{\partial q_c}{\partial x} + n_x \lambda_x^i L_x P \frac{(q_p^{(g)} - q_p)}{\Delta}$$

where Δ is the grid spacing between the boundary (q_p) and the ghost point ($q_p^{(g)}$). In practice, Δ is set to be the grid spacing between the boundary and the first interior point for accuracy and stability. For a subsonic outflow at the right boundary, coefficients $\mathcal{L}_2^i$ through $\mathcal{L}_5^i$ are set to 0 as these correspond to outgoing waves. The incoming characteristic $\mathcal{L}_1^i$ is calculated as

$$\mathcal{L}_1^i = (u-c)((p^{(g)} - p) - \rho c(u^{(g)} - u)),$$

where $p^{(g)}$ and $u^{(g)}$ are the pressure and normal velocity in the exterior of the domain.

One can extend the 1D approach implemented here to treat strong oblique waves. Roughly, one would add a dependence on tangential wavenumbers at the boundary, following the oblique non-reflecting formulations of Giles [76] and the 3D NSCBC extensions by Lodato et al. [77]. Impedance in the time domain can be enforced via a stable filter [78] and time-domain impedance [79], resulting in a non-local time closure that is efficient.

4.2.2. WENO-Z, TENO, and high-order non-uniform reconstruction

The WENO-Z scheme [80] enhances classical WENO methods by improving their spectral properties while being faster than WENO-M. The $(2k-1)$th order WENO-Z weight $\omega_r^{(Z)}$ for the rth stencil is computed as

$$\omega_r^{(Z)} = \frac{\alpha_r}{\sum_{i=0}^{k-1} \alpha_i}, \quad \alpha_r = d_r\left(1 + \left(\frac{\tau}{\beta_r + \epsilon}\right)^q\right), \quad \tau = |\beta_0 - \beta_{k-1}|,$$

where d_r are the ideal weights, τ is the global smoothness indicator, and the parameter q (typically set to 1 for fifth-order reconstruction) influences the convergence rate at the critical points and the spectral properties. This formulation applies to schemes of order 3, 5, and 7.

The Targeted Essentially Non-Oscillatory (TENO) scheme further reduces numerical dissipation by employing a stencil selection process similar to the ENO method [81]. A Kth order accurate TENO scheme uses $K-2$ candidate stencils of increasing width. Its equations are given by

$$\omega_r^{(T)} = \frac{d_r \delta_r}{\sum_{i=0}^{K-3} d_i \delta_i}, \quad \delta_r = \begin{cases} 0, & \text{if } \chi_r < C_T \\ 1, & \text{otherwise} \end{cases}, \quad \chi_r = \frac{\gamma_r}{\sum_{i=0}^{K-3} \gamma_i},$$

$$\gamma_r = d_r\left(1 + \frac{\tau}{\beta_r + \epsilon}\right)^6,$$

where C_T is a smoothness threshold for stencil activation and τ is the WENO-Z parameter.

The explicit expressions for the seventh-order WENO reconstruction in our code are omitted here due to their considerable length. Instead, we present a generalized approach for reconstructing a $(2k-1)$-order scheme from cell averages on non-uniform grids. For each stencil, the

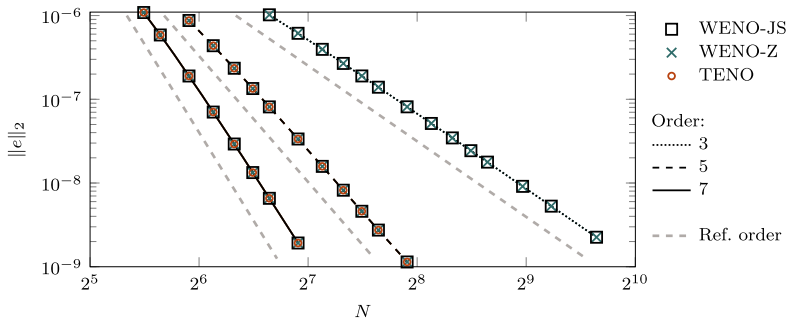


Fig. 12. Convergence results for a 1D linear advection problem with WENO-JS, WENO-Z, and TENO schemes at orders of accuracy 3, 5, and 7.

reconstruction polynomial is obtained using the Lagrange interpolation method [82]:

$$p(x) = \sum_{j=0}^{k-1} \underbrace{\Delta x_j \sum_{m=j+1}^{k} \left(\frac{\sum_{l=0, l\neq m}^{k} \prod_{q=0, q\neq m,l}^{k} \left(x - x_{q-\frac{1}{2}} \right)}{\prod_{l=0, l\neq m}^{k} \left(x_{m-\frac{1}{2}} - x_{l-\frac{1}{2}} \right)} \right)}_{c_j(x)} f_j.$$

Here, x_j denotes cell-center location, $x_{j+1/2}$ the cell-boundary location, and $\Delta x_j := x_{j+1/2} - x_{j-1/2}$ is the cell width. For each stencil $r = 0, 1, \ldots, k-1$, the k-cell stencil spans the interval from $x_{-1/2}$ to $x_{k-1/2}$. Consequently, the overall $(2k-1)$-cell stencil is centered at x_{k-r-1}, and the right flux is $p_r := p(x_{k-r-1/2})$.

To reduce computational cost, we rewrite $p(x)$ in terms of the differences $\Delta f_j := f_{j+1} - f_j$, so that $p(x) = \sum_{j=0}^{k-2} \hat{c}_j(x) \Delta f_j + f_0$. For flux evaluation at $x_{k-r-1/2}$, the coefficients $\hat{c}_{r,j} := \hat{c}_j(x_{k-r-1/2})$ for each Δf_j and stencil r are precomputed at the start of the program.

The smoothness indicator for stencil r is defined by

$$\beta_r = \sum_{l=1}^{k-1} \Delta x_{k-r-1}^{2l-1} \int_{x_{k-r-\frac{3}{2}}}^{x_{k-r-\frac{1}{2}}} \left(\frac{\partial^l p(x)}{\partial x^l} \right)^2 \mathrm{d}x.$$

Rewriting this in terms of the difference formulation yields

$$\beta_r = \sum_{j=0}^{k-2} \sum_{m=j}^{k-2} \underbrace{(2-\delta_{jm}) \sum_{l=1}^{k-1} \Delta x_{k-r-1}^{2l-1} \int_{x_{k-r-\frac{3}{2}}}^{x_{k-r-\frac{1}{2}}} \frac{\partial^l \hat{c}_j(x)}{\partial x^l} \frac{\partial^l \hat{c}_m(x)}{\partial x^l} \mathrm{d}x}_{\hat{b}_{r,j,m}} \Delta f_j \Delta f_m,$$

where δ_{jm} is the Kronecker delta. The coefficients $\hat{b}_{r,j,m}$ for each cross term $\Delta f_j \Delta f_m$ and stencil r are precomputed. Reformulating in terms of Δf reduces the number of terms from k to $k-1$ for p_r, and from $k(k+1)/2$ to $(k-1)k/2$ for β_r, thereby reducing the computational cost.

The convergence study in Fig. 12 demonstrates that WENO-Z and TENO recover their intended design orders, consistent with the original WENO-JS for 3rd-, 5th-, and 7th-order reconstruction.

4.2.3. Strang splitting for stiff sub-grid dynamics

Sub-grid bubbles often exhibit rapid dynamics, necessitating a small time step for stable computation of the bubble source term. When the time scale of the background flow significantly exceeds that of the sub-grid bubbles, fully resolving the temporal dynamics becomes computationally prohibitive. To address this, we implement the Strang splitting scheme, which separates the time integration of the background flow and the sub-grid bubbles [83]. The implementation supports a single resolved phase with sub-grid bubbles governed by

$$\frac{\partial \boldsymbol{q}}{\partial t} = \nabla \cdot \boldsymbol{F}(\boldsymbol{q}) + \boldsymbol{s}(\boldsymbol{q}).$$

The Strang splitting scheme integrates the equation over one time step by integrating three sub-equations for $\boldsymbol{q}^*$, $\boldsymbol{q}^{**}$, and $\boldsymbol{q}^{***}$ as

$$\frac{\partial \boldsymbol{q}^*}{\partial t} = \boldsymbol{s}(\boldsymbol{q}), \qquad t \in [0, \Delta t/2], \quad \boldsymbol{q}^*(0) = \boldsymbol{q}^n,$$

$$\frac{\partial \boldsymbol{q}^{**}}{\partial t} = -\nabla \cdot \boldsymbol{F}(\boldsymbol{q}^{**}), \quad t \in [0, \Delta t], \quad \boldsymbol{q}^{**}(0) = \boldsymbol{q}^*(\Delta t/2),$$

$$\frac{\partial \boldsymbol{q}^{***}}{\partial t} = \boldsymbol{s}(\boldsymbol{q}^{***}), \qquad t \in [0, \Delta t/2], \quad \boldsymbol{q}^{***}(0) = \boldsymbol{q}^{**}(\Delta t), \boldsymbol{q}^{n+1} = \boldsymbol{q}^{***}(\Delta t/2).$$

To solve for $\boldsymbol{q}^*$ and $\boldsymbol{q}^{***}$, where the stiff bubble source term is evaluated, we use a third-order accurate embedded Runge–Kutta scheme with adaptively controlled step sizes [84]. The step size, h, is updated based on the estimated error, $\boldsymbol{e}$, calculated as

$$\boldsymbol{q}^{n+1} = \boldsymbol{q}^n + \frac{h}{6}(k_1 + k_2 + 4k_3),$$

$$\hat{\boldsymbol{q}}^{n+1} = \boldsymbol{q}^n + \frac{h}{8}(3k_1 + 3k_2 + 2k_3),$$

$$\boldsymbol{e} = \boldsymbol{q}^{n+1} - \hat{\boldsymbol{q}}^{n+1} = -\frac{5}{24} h (k_1 + k_2 - 2k_3),$$

where $k_1 = \partial \boldsymbol{q}^n / \partial t$, $k_2 = \partial \boldsymbol{q}^{(1)} / \partial t$, and $k_3 = \partial \boldsymbol{q}^{(2)} / \partial t$. Here, $\boldsymbol{q}^{n+1}$ represents a solution of the third-order TVD Runge–Kutta scheme, and $\hat{\boldsymbol{q}}^{n+1}$ is a 2nd-order accurate solution obtained using the same k_1, k_2, and k_3 that requires minimal additional computational cost. More details of the adaptive step size control algorithm can be found in Hairer et al. [84].

We demonstrate the utility of the operator splitting scheme with adaptive time stepping for a 0D simulation of bubble dynamics. When a bubble is located in a high-pressure liquid, it collapses in radial size and rebounds (size increases) repeatedly, eventually damping due to viscous dissipation. Here, we test our implementation via a $\boldsymbol{u} = \boldsymbol{0}$ case and initial pressure field such that the liquid pressure is 1427-times larger than the bubbles' internal pressure p_b. The bubbles are assumed to be monodisperse in size and air-filled with equilibrium radius $R_{\mathrm{eq}} = 5\,\mathrm{mm}$. The 0D simulations are implemented as spatially homogeneous 1D cases.

Fig. 13 shows the temporal evolution of bubble radius. An unsplit scheme with time step size $\Delta t = 0.008\,\mu\mathrm{s}$ can resolve the collapse and rebound of bubbles. However, $\Delta t = 0.8\,\mu\mathrm{s}$ is insufficiently small to resolve the collapse event and leads to simulation failure. The failure occurs when the bubble radius becomes negative due to the time step sizes that are too large for the bubble radial velocity. The use of an operator-splitting scheme with adaptive time stepping enables the simulation to represent the rapid collapse and rebound with a time step size of $\Delta t = 0.8\,\mu\mathrm{s}$.

4.2.4. Low-Mach number treatment

Godunov-type Riemann solvers, including the HLLC Riemann solver, are known to lose their accuracy at low Mach numbers due to numerical dissipation in the discrete solution [86]. To address this limitation, we use two numerical treatments based on the approaches of Thornber et al. [87] and Chen et al. [88], allowing users to select their preferred scheme.

Thornber et al. [87] proposed a simple modification to the reconstructed velocities of the left and right states as

$$u_{\mathrm{L}}^* = \frac{u_{\mathrm{L}} + u_{\mathrm{R}}}{2} + z \frac{u_{\mathrm{L}} - u_{\mathrm{R}}}{2},$$

$$u_{\mathrm{R}}^* = \frac{u_{\mathrm{L}} + u_{\mathrm{R}}}{2} + z \frac{u_{\mathrm{R}} - u_{\mathrm{L}}}{2},$$

$$z = \min\left(\max\left(\frac{|u_{\mathrm{L}}|}{c_{\mathrm{L}}}, \frac{|u_{\mathrm{R}}|}{c_{\mathrm{R}}} \right), 1 \right),$$

where the modified velocities u_{L}^* and u_{R}^* replace the original velocities u_{L} and u_{R} in the calculation of the signal velocity and fluxes. This strategy

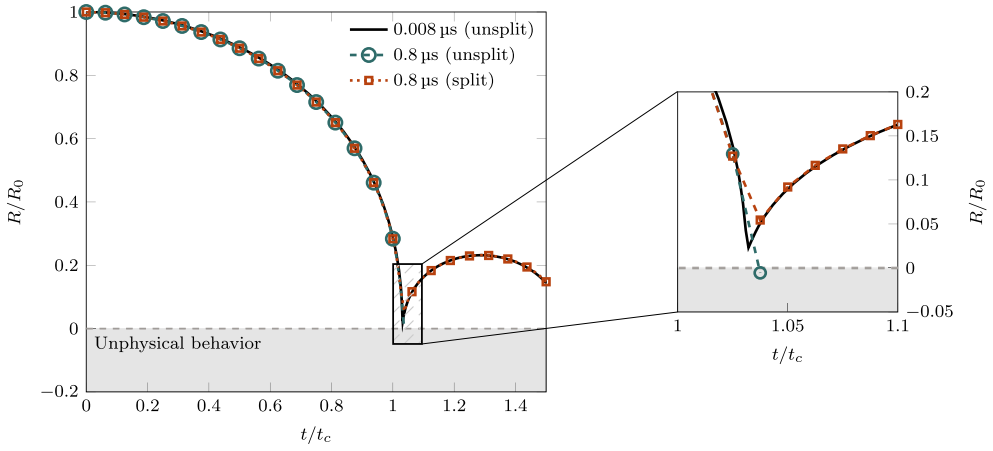


Fig. 13. Evolution of bubble radius at pressure ratio $p_\infty/p_b = 1427$ with and without the Strang splitting scheme and adaptive time stepping. t_c is the total collapse time [85]. The right plot is a magnified view of the boxed region near the rebound in the left plot.

ensures the correct scaling of the pressure and density fluctuations in the low Mach number regime.

On the other hand, Chen et al. [88] identified a source term, p_d, which is responsible for the wrong scaling at the low Mach number limit. To address this, they proposed an anti-dissipation pressure correction (APC) term, which is incorporated into the flux calculation. For the HLLC Riemann solver, the terms p_d and APC are

$$p_d = \frac{\rho_L \rho_R (S_L - u_L)(S_R - u_R)(u_R - u_L)}{\rho_R (S_R - u_R) - \rho_L (S_L - u_L)},$$
$$\mathrm{APC}_{\rho u} = (z-1) p_d \left(n_x, n_y, n_z\right)^\top,$$
$$\mathrm{APC}_{\rho E} = (z-1) p_d S_*,$$
$$z = \min\left(\max\left(\frac{|u_L|}{c_L}, \frac{|u_R|}{c_R}\right), 1\right).$$

Here, $\mathrm{APC}_{\rho u}$ and $\mathrm{APC}_{\rho E}$ represent the correction terms for the momentum and energy equations, respectively, and n_x, n_y, and n_z denote the components of the unit normal vector. Then, the corrected HLLC flux is

$$\boldsymbol{F}^{\mathrm{HLLC+APC}} = \begin{cases} \boldsymbol{F}(\boldsymbol{q}^{(\mathrm{L})}) & S_L \geq 0, \\ \boldsymbol{F}^{(*\mathrm{L})} + \boldsymbol{APC} & S_L \leq 0 \leq S_*, \\ \boldsymbol{F}^{(*\mathrm{R})} + \boldsymbol{APC} & S_* \leq 0 \leq S_R, \\ \boldsymbol{F}(\boldsymbol{q}^{(\mathrm{R})}) & 0 \geq S_R. \end{cases}$$

The HLLC Riemann solver with correction schemes is validated with the 2D Gresho vortex, which is an exact solution to the incompressible Euler equations [89]. The angular velocity and pressure are given by

$$u_\phi = u_r \begin{cases} r/R, & 0 \leq r < R, \\ 2 - r/R, & R \leq r \leq 2R, \\ 0, & 2R \leq r, \end{cases}$$

$$p = p_r + \rho_r u_r^2 \begin{cases} (r/R)^2/2, & 0 \leq r < R, \\ (r/R)^2/2 + 4(1 - (r/R) + \ln(r/R)), & R \leq r \leq 2R, \\ -2 + 4\ln 2, & 2R \leq r, \end{cases}$$

where $r = \sqrt{x^2 + y^2}$ is the radial coordinate, $u_r = 2\pi R M_r$ and $M_r = u_r / \sqrt{\gamma\left(p_r/\rho_r + u_r^2/2\right)}$ are reference velocity and reference Mach number, respectively. The corresponding reference time scale is $t_r = 2\pi R/u_r = 1/M_r$, which is the time for one vortex turn around. Since the velocity and pressure fields are in equilibrium, the flow is expected to be time-stationary. Fig. 14 shows the local Mach number at $t/t_r = 1$ at four reference Mach numbers: $M_r = 10^{-1}$, 10^{-2}, 10^{-3}, and 10^{-4}. The standard HLLC approximate Riemann solver shows a loss of accuracy for smaller M_r. In contrast, the HLLC solver with correction schemes preserves the flow field to visual accuracy for all tested Mach numbers.

4.2.5. HLLD Riemann solver

For the MHD equations, the HLLD (Harten–Lax–van Leer–Discontinuities) Riemann solver [72] builds on the HLL framework to resolve five of the seven characteristic waves, thereby reducing numerical dissipation and improving accuracy.

In this approach, the Riemann fan is divided into four intermediate states $\mathbf{U}_L^*$, $\mathbf{U}_L^{**}$, $\mathbf{U}_R^{**}$, and $\mathbf{U}_R^*$. These states are separated by two Alfvén waves (S_L^* and S_R^*) and a middle contact discontinuity (S_M). The fastest left and right waves (S_L and S_R) form the outer boundaries of the entire Riemann fan.

Similar to the HLL solver, S_L and S_R are approximated by

$$S_L = \min(u_L - c_{f,L}, u_R - c_{f,R}) \quad \text{and} \quad S_R = \max(u_L + c_{f,L}, u_R + c_{f,R}).$$

The HLLD solver enforces two key constraints: normal velocity and total pressure remain constant across the Riemann fan:

$$u_L^* = u_L^{**} = u_R^{**} = u_R^* = S_M,$$
$$p_{T,L}^* = p_{T,L}^{**} = p_{T,R}^{**} = p_{T,R}^* = p_T^*.$$

The middle wave speed is calculated as:

$$S_M = \frac{(S_R - u_R)\rho_R u_R - (S_L - u_L)\rho_L u_L - p_{T,R} + p_{T,L}}{(S_R - u_R)\rho_R - (S_L - u_L)\rho_L}.$$

For the outer intermediate states, we first compute the density:

$$\rho_\alpha^* = \rho_\alpha \frac{S_\alpha - u_\alpha}{S_\alpha - S_M},$$

where $\alpha = \mathrm{L}$ or R. For brevity, explicit expressions for tangential velocities (v_α^*, w_α^*) and tangential magnetic fields ($B_{y,\alpha}^*$, $B_{z,\alpha}^*$), as well as the energy computation, are not shown here but are available in Miyoshi and Kusano [72]. The Alfvén wave speeds are determined by:

$$S_L^* = S_M - \frac{|B_x|}{\sqrt{\rho_L^*}}, \quad S_R^* = S_M + \frac{|B_x|}{\sqrt{\rho_R^*}}.$$

For inner states, the tangential velocity is:

$$v^{**} = \frac{\sqrt{\rho_L^*} v_L^* + \sqrt{\rho_R^*} v_R^* + (B_{y,R}^* - B_{y,L}^*)\,\mathrm{sign}(B_x)}{\sqrt{\rho_L^*} + \sqrt{\rho_R^*}}.$$

The corresponding expressions for w^{**}, B_y^{**}, and B_z^{**} in the inner states, along with the associated energy calculation, follow a similar pattern and can be found in Miyoshi and Kusano [72].

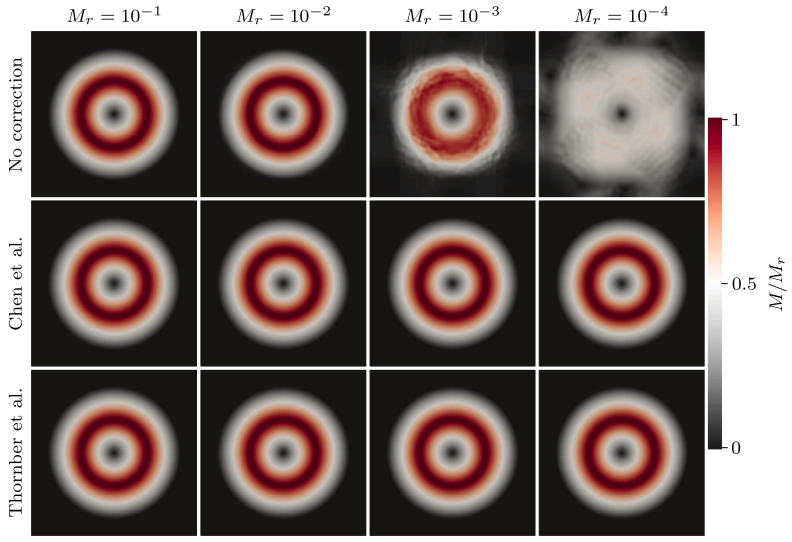


Fig. 14. Ratio of local Mach number, M, to reference Mach number, M_r, for the Gresho vortex at $t/t_r = 1$ with $M_r = 10^{-1}$, 10^{-2}, 10^{-3} and 10^{-4}: (top row) without correction; (middle row) Chen et al. [88]; (bottom row) Thornber et al. [87].

Table 1
Compiler support for GPU offload to Nvidia hardware, AMD hardware, and for unified shared memory mode on AMD hardware. NVHPC SDK 25.9, CCE 19.0.0, and AMD Flang Preview 7.0.5 are used to populate this table. Support for both the AMD and CCE compilers is provided to enable AMD GPU offloading when CCE is not available.

	OpenMP			OpenACC		
Compiler	NVIDIA GPU	AMD A/GPU	AMD USM	NVIDIA GPU	AMD A/GPU	AMD USM
AMD	✗	✔	✔	✗	✗	✗
CCE	✔	✔	✔	✔	✔	✗
NVHPC	✔	✗	✗	✔	✗	✗

The numerical flux is chosen based on the region of the Riemann fan that contains the cell interface:

$$\mathbf{F}_{\text{HLLD}} = \begin{cases} \mathbf{F}_{\text{L}} & \text{if } S_{\text{L}} > 0, \\ \mathbf{F}^{*}_{\text{L}} & \text{if } S_{\text{L}} \le 0 \le S^{*}_{\text{L}}, \\ \mathbf{F}^{**}_{\text{L}} & \text{if } S^{*}_{\text{L}} \le 0 \le S_{\text{M}}, \\ \mathbf{F}^{**}_{\text{R}} & \text{if } S_{\text{M}} \le 0 \le S^{*}_{\text{R}}, \\ \mathbf{F}^{*}_{\text{R}} & \text{if } S^{*}_{\text{R}} \le 0 \le S_{\text{R}}, \\ \mathbf{F}_{\text{R}} & \text{if } S_{\text{R}} < 0. \end{cases}$$

To validate the HLLD implementation, we use the standard one-dimensional MHD Brio–Wu shock-tube problem from Miyoshi and Kusano [72]. The density profile in Fig. 10 (a) closely matches the reference, as do the other variables (not shown).

4.3. Software and high-performance computing

4.3.1. APU and GPU offloading with OpenACC and OpenMP

Vendor portable GPU offloading on AMD and NVIDIA hardware is implemented using OpenACC [90] and OpenMP [91]. OpenACC and OpenMP are directive-based tools that enable developers to generate GPU kernels by adding directive clauses around regions of parallelism identified in the code. Table 1 summarizes the current state of compiler support for OpenMP and OpenACC on NVIDIA and AMD GPUs, and for unified shared memory mode on AMD hardware. OpenACC is the preferred offloading tool in MFC due to its superior performance over OpenMP on NVIDIA and AMD hardware [11]. OpenMP is used as a fallback when unified shared memory (USM) is required on AMD hardware. Support for AMD compilers is provided for using AMD GPUs when CCE is not available. Compilers like GNU 14 and Flang 18 support OpenACC, though their relative immaturity at the time of writing makes them insufficient to support MFC.

Support for OpenACC and OpenMP is implemented through a single source code version using the Fortran preprocessor Fypp [92] (additional uses of Fypp in MFC are described in Section 4.3.6). Listing 1 shows an example of a typical GPU kernel in MFC. Parallel regions are wrapped with the `GPU_PARALLEL_LOOP` and `END_GPU_PARALLEL_LOOP` macros, which expand to OpenACC or OpenMP directives depending on the selected offloading tool. The outer loops over `j`, `k`, and `l` iterate through $\mathcal{O}(100)$ elements each, corresponding to the grid cells in the MPI subdomain. The inner loop over `i` iterates through each equation in the current model. For two-phase flow problems, this loop spans $\mathcal{O}(1)$ elements. When additional features like hypo- and hyperelasticity or chemistry are added to the problem, the extent of this inner loop can be $\mathcal{O}(10)$.

Listing 2a shows the OpenACC directives generated by the Fypp macros in Listing 1. OpenACC describes parallelism using terminology in which gangs, workers, and vectors refer to blocks, warps, and threads in CUDA notation. By default, when OpenACC encounters a `parallel loop` clause, it distributes the workload across gangs, leaving each gang to utilize a single vector. Appending the `gang vector` clause to the `parallel loop` tells the compiler to split the work across multiple gangs with a fixed vector length, which more efficiently uses available resources and increases execution speed. Appending `collapse(3)` to the three outer loops instructs the compiler to collapse those three loops and select the optimal gang and vector sizes based on the current problem and architecture. Our tests show that serialization of the innermost loop `i` increases performance, in part due to its relatively small range.

Listing 2b shows the OpenMP directives generated for the same kernel by the Fypp macros in Listing 1. OpenMP offloads parallel regions to the GPU when it encounters an `omp target` directive. Adding `teams` after `omp target` instructs the compiler to launch a league of teams,

```
$:GPU_PARALLEL_LOOP(collapse=3, private='[...]')
do l = 0, p ! Third coordinate direction
    do k = 0, n ! Second coordinate direction
        do j = 0, m ! First coordinate direction
            $:GPU_LOOP(parallelism='[seq]')
            do i = 1, num_PDEs
                ! Core kernel here
                ! O(100) arithmetic operations
            end do
        end do
    end do
end do
$:END_GPU_PARALLEL_LOOP$
```

Listing 1. Directive setup for a typical MFC OpenACC kernel. The Fypp macros `GPU_PARALLEL_LOOP` and `END_GPU_PARALLEL_LOOP` expand to OpenACC or OpenMP directives depending on the selected offloading tool.

which typically map to CUDA thread blocks on NVIDIA GPUs. The combined `teams distribute parallel do` construct then further decomposes the iteration space: `distribute` partitions iterations across teams (i.e., across blocks), and `parallel do` partitions iterations across threads within each team. The `defaultmap` clauses specify how scalars, aggregates, allocatables, and pointers should be implicitly mapped to the device, since OpenMP requires more explicit data-mapping semantics than OpenACC. Adding `collapse(3)` collapses the three nested loops into a single iteration space, improving load balance and work distribution across teams and threads, just as in OpenACC.

```
!$acc parallel loop vector gang collapse(3) &
!$acc default(present) private(...)
do l = 0, p      ! Z-direction
  do k = 0, n    ! Y-direction
    do j = 0, m  ! X-direction
        !$acc loop seq
        do i = 1, num_PDEs
            ! Core kernel
            ! O(100) - O(1000) operations
        end do
    end do
  end do
end do
!$acc end parallel loop
```

Listing 2a. OpenACC directives generated by Fypp macros.

```
!$omp target teams distribute parallel do &
!$omp simd defaultmap(firstprivate:scalar) &
!$omp defaultmap(tofrom:aggregate) &
!$omp defaultmap(present:allocatable) &
!$omp defaultmap(present:pointer) &
!$omp collapse(3) private(...)
do l = 0, p      ! Z-direction
  do k = 0, n    ! Y-direction
    do j = 0, m  ! X-direction
        !$omp loop bind(thread)
        do i = 1, num_PDEs
            ! Core kernel
            ! O(100) - O(1000) operations
        end do
    end do
  end do
end do
!$omp end target teams distribute parallel do
```

Listing 2b. OpenMP directives generated by Fypp macros.

MFC also uses NVIDIA and AMD's vendor-provided libraries `cuTensor` and `hipBLAS` to perform hardware-optimized array reshaping for coalescing memory before the most expensive kernels. Memory coalescence before the WENO reconstruction and approximate Riemann solver kernels results in a ten-fold speedup due to the increased throughput of high-bandwidth memory (HBM). `cuTENSOR` [93] performs similarly to fully collapsed OpenACC loops on NVIDIA hardware, providing only marginal speedup. `hipBLAS` [94] performs array transposes approximately seven times faster than fully collapsed OpenACC loops. NVIDIA and CCE's `cuFFT` [95] and `hipFFT` [96] perform fast Fourier transforms on GPU devices, and `FFTW` [97] is used for CPU-based simulations. Code enclosed by OpenACC `host_data use_device` and `end host_data` or OpenMP `target data` and `end target data` clauses are executed on the GPU using these vendor-provided libraries, reducing the need for hardware-specific optimizations.

4.3.2. Performance on various CPU, GPU, and APU architectures

MFC 5.0 has been benchmarked on a broad range of CPUs, GPUs, and APUs (also known as superchips). Table 2 reports the single-device performance in terms of its grind time, which is computed as time per grid point, PDE, and right-hand side evaluation. The grind times are calculated for one GPU, APU, or CPU socket. The benchmarks were performed using a similar strategy to that employed in published testbed reports [98]. These quantities are for a typical, representative compressible multi-component problem: 3D, inviscid, 5-equation model problem with two advected species (8 PDEs) and 8M grid points (158-cubed 3D uniform grid). The equation is solved via fifth-order accurate WENO finite volume reconstruction and the HLLC approximate Riemann solver. This case is in the MFC source code under `examples/3D_performance_test`. One can execute it via

```
./mfc.sh run examples/3D_performance_test/case.py -t pre_process
    simulation --case-optimization -n <num_cores_or_gpus> -j
    <num_build_threads>
```

This command can be executed for CPU cases. It builds an optimized version of the code for this case and then executes it. For benchmarking GPU devices, one will likely want to use `-n <num_gpus>` where the usual case for single GPU device benchmarking leaves `<num_gpus>` as unity. Similar performance is also observed for other problem configurations, such as the Euler equations (4 PDEs). All results are for the compiler that gave the best performance, including AOCC, Intel, GCC, CCE, and NVHPC.

CPU results may be performed on CPUs with more cores than reported in Table 2; we report results for the best performance, given the full processor die, by checking performance for different core counts on that device. CPU results are the best performance we achieved using a single socket (or die). GPU results are for a single GPU device. For single-precision (SP) GPUs, we performed computations in double-precision via compiler/software conversion; these numbers are not representative of single-precision computations. AMD MI250X and MI300A devices have multiple graphics compute dies per socket; we report results for two GCDs for the AMD MI250X and the entire APU (6 XCDs) for the AMD MI300A.

Fig. 15 shows the roofline performance for the most expensive kernels on NVIDIA and AMD accelerators. At the time of writing, roofline analysis is not available on the MI300A APU and therefore is not shown. NVHPC 24.5 compilers and NVIDIA Nsight Compute profilers produce roofline results on NVIDIA devices, and CCE 18 with Omniperf profiles of the roofline on the AMD MI250X GPU. The WENO and approximate Riemann kernels achieve 37% and 14% of peak performance on the NVIDIA A100. On the NVIDIA GH200, the WENO and approximate Riemann kernels achieve 24% and 5% of peak compute performance. The AMD MI250X achieves 22% of peak compute performance in the WENO kernel and 3% of peak performance in the approximate Riemann problem.

The breakdown of time spent in the two most expensive kernels, time spent packing arrays, and time spent doing all other computations and communication is shown in Fig. 16. The WENO and approximate Riemann kernels account for about 50% of the total wall time on NVIDIA GPUs. On AMD GPU devices, the time spent in the WENO and approx-

Table 2
Observed grind time performance (time), reported as nanoseconds per grid point per PDE per right-hand side evaluation. Using more GPU devices or CPU sockets may not improve the grind time for the same problem, as the problem will be made smaller per marshaled device (and so grind time is expected to be approximately constant).

Hardware	Type	Usage	Time	Hardware	Type	Usage	Time
NVIDIA GH200	APU	1 GPU	0.32	Intel Xeon 6740E	CPU	92 cores	4.2
NVIDIA H100 SXM5	GPU	1 GPU	0.38	NVIDIA A10	GPU	1 GPU	4.3
NVIDIA H100 PCIe	GPU	1 GPU	0.45	AMD EPYC 7713	CPU	64 cores	5.0
NVIDIA B200	GPU	1 GPU	0.46	Intel Xeon 8480CL	CPU	56 cores	5.0
AMD MI250X	GPU	1 GPU	0.55	Intel Xeon 6454S	CPU	32 cores	5.6
AMD MI300A	APU	1 APU	0.57	Intel Xeon 8462Y+	CPU	32 cores	6.2
NVIDIA A100	GPU	1 GPU	0.62	Intel Xeon 6548Y+	CPU	32 cores	6.6
NVIDIA V100	GPU	1 GPU	0.99	Intel Xeon 8352Y	CPU	32 cores	6.6
NVIDIA A30	GPU	1 GPU	1.1	Ampere Altra Q80-28	CPU	80 cores	6.8
AMD EPYC 9965	CPU	192 cores	1.2	AMD EPYC 7513	CPU	32 cores	7.4
AMD MI100	GPU	1 GPU	1.4	Intel Xeon 8268	CPU	24 cores	7.5
AMD EPYC 9755	CPU	128 cores	1.4	AMD EPYC 7452	CPU	32 cores	8.4
Intel Xeon 6980P	CPU	128 cores	1.4	NVIDIA T4	GPU	1 GPU	8.8
NVIDIA L40S	GPU	1 GPU	1.7	Intel Xeon 8160	CPU	24 cores	8.9
AMD EPYC 9654	CPU	96 cores	1.7	IBM Power10	CPU	24 cores	10
Intel Xeon 6960P	CPU	72 cores	1.7	AMD EPYC 7401	CPU	24 cores	10
NVIDIA P100	GPU	1 GPU	2.4	Intel Xeon 6226	CPU	12 cores	17
Intel Xeon 8592+	CPU	64 cores	2.6	Apple M1 Max	CPU	10 cores	20
Intel Xeon 6900E	CPU	192 cores	2.6	IBM Power9	CPU	20 cores	21
AMD EPYC 9534	CPU	64 cores	2.7	Cavium ThunderX2	CPU	32 cores	21
NVIDIA A40	GPU	1 GPU	3.3	Arm Cortex-A78AE	CPU	16 cores	25
Intel Xeon Max 9468	CPU	48 cores	3.5	Intel Xeon E5-2650V4	CPU	12 cores	27
NVIDIA Grace CPU	CPU	72 cores	3.7	Apple M2	CPU	8 cores	32
NVIDIA RTX6000	GPU	1 GPU	3.9	Intel Xeon E7-4850V3	CPU	14 cores	34
AMD EPYC 7763	CPU	64 cores	4.1	Fujitsu A64FX	CPU	48 cores	63

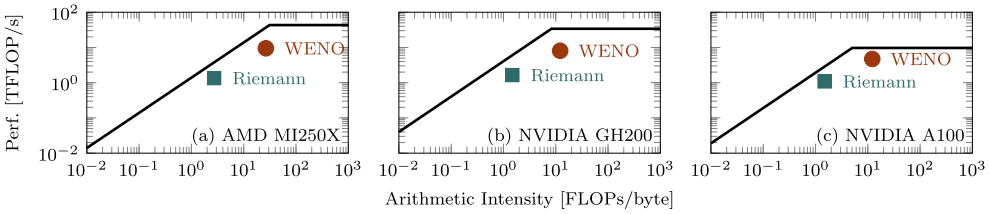


Fig. 15. The roofline performance of the fifth-order accurate WENO reconstruction and HLLC approximate Riemann solve kernels on the AMD MI250X, NVIDIA GH200, and NVIDIA A100.

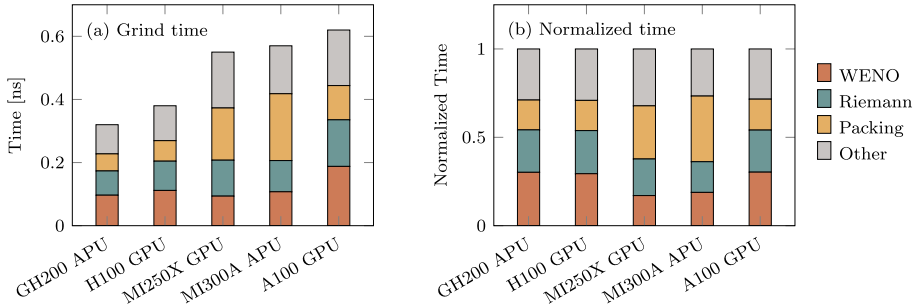


Fig. 16. Grind time and normalized breakdown of the most expensive kernels, array packing, and all other computations on NVIDIA (GH200, H100, A100) and AMD (MI250X, MI300A) GPUs/APUs.

imate Riemann kernels is similar to that of NVIDIA devices. Still, these kernels account for about a third of the total wall time due to an increase in the time spent packing arrays for coalesced memory access. The profiles show a high number of cache misses on the AMD MI250X, which we expect is due to the smaller device caches compared to those of the shown NVIDIA device.

4.3.3. Exascale capabilities and beyond

MFC efficiently scales to tens of thousands of GPUs on the world's fastest computers. Fig. 17 shows the weak scaling performance of MFC on OLCF Summit (now retired) with NVIDIA V100 GPUs, OLCF Frontier with AMD MI250X GPUs, and LLNL El Capitan with AMD MI300A APUs. MFC scales to 13,824 NVIDIA V100 GPUs on OLCF Summit with 97% efficiency, to 65,536 AMD MI250X GCDs (32768 MI250X GPUs) on OLCF Frontier with 95% efficiency, and to 196,608 AMD MI300As XCDs (32768 MI300A APUs) on LLNL El Capitan.

The communication patterns in MFC lead to high strong scaling efficiencies. Specifically, the communication involves only a small halo region around each uniformly sized domain decomposition chunk, and the relatively large amount of computation associated with each grid point. The decomposed grid chunks, assigned one-to-one per MPI rank, are not expanded or contracted. Since each computing device handles

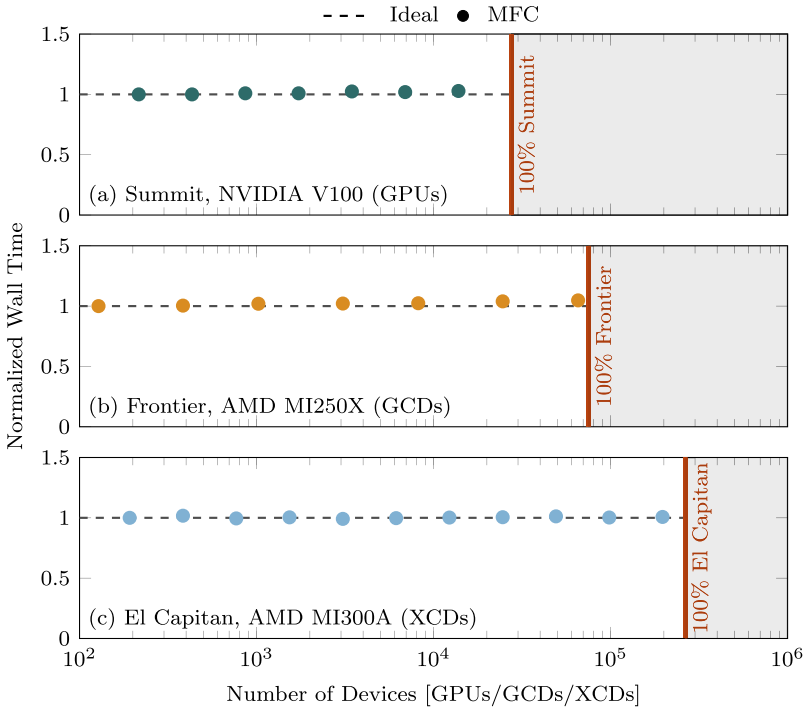


Fig. 17. Weak scaling results on OLCF Summit and Frontier and LLNL El Capitan. MFC Scales from 128 to 13,824 NVIDIA V100 GPUs with 97% efficiency on Summit, from 128 to 65,536 AMD MI250Xs (GCDs) with 95% efficiency on Frontier, and 192 to 196,608 AMD MI300As (XCDs) with 99% efficiency on El Capitan (XCDs).

the same number of elements, the halo exchange cost is the same for each MPI rank, which also explains the ideal weak scaling behavior: The number of domain decomposition chunks increases linearly with problem size, so there is no opportunity for weak scaling inefficiencies.

Fig. 18 shows the strong scaling performance of MFC on OLCF Summit and OLCF Frontier and compares weak scaling with and without GPU-aware MPI. The problem sizes in Fig. 18 (a) correspond to approximately 100% and 50% usage of available GPU memory per device in the base case. In the base case, with 8M grid cells per device, strong scaling efficiencies of 79% and 51% are observed on OLCF Summit when increasing the device count by a factor of 8 or 16 without GPU-aware MPI. For the case with 32 million grid cells per device in the base case, strong scaling efficiencies of 81% and 77% when increasing the device count by a factor of 8 and 16 without GPU-aware MPI. The higher efficiencies seen with the AMD MI250X on OLCF Frontier are partly due to the increased computation-to-communication ratio resulting from larger problem sizes and the improved internode communication on Frontier. With GPU-aware MPI, the strong scaling efficiencies on OLCF Summit increase to 84% and 60% when increasing the device count by a factor of 8 and 16. On Frontier, the efficiencies are 96% and 92% when the device count is increased by the same factor.

4.3.4. *Parallel I/O at extreme scales*

Parallel I/O is implemented using the MPI I/O library to facilitate shared file parallel I/O and file-per-process parallel I/O. Shared file parallel I/O was sufficient for handling up to 10K processes, but a significant slowdown in I/O times was observed when scaling to over 65K processes on OLCF Frontier. For simulations that use over 10K processes, each process writes a file for each I/O operation. To alleviate file system pressure resulting from metadata creation for tens of thousands of files, MFC accesses the file system in waves of 128 processes, separated by a fixed number of floating-point operations.

4.3.5. *Software resilience and continuous integration*

Each pull request to MFC goes through a suite of continuous integration actions. The correctness, performance, and code quality are evaluated through this continuous integration suite. MFC's test suite, comprising over 500 cases, is compiled and run using GNU and Intel compilers on macOS and Ubuntu operating systems, utilizing GitHub Actions and Runners. The same tests are run using NVHPC on CPUs and NVIDIA GPUs, as well as with CCE on AMD GPUs via self-hosted runners. All of these tests are compared to the same set of golden files to ensure that MFC's code provides the same results on CPUs and GPUs, regardless of compiler or operating system. The coverage of the test suite is evaluated using Codecov to help ensure coverage of the test suite for additional (or removed) code. Each pull request is benchmarked against the main MFC branch on CPUs and GPUs to prevent performance regressions; it also checks for compiler warnings and formatting to ensure code consistency.

4.3.6. *Metaprogramming*

Metaprogramming is implemented using the Fortran preprocessor Fypp [92]. Fypp is a Python-based preprocessor that inlines subroutines via directives and collapses repetitive code. MFC uses Fypp for several tasks, including automating deep copies of derived types for allocation on the GPU device. NVIDIA's NVHPC Fortran compiler automatically allocates derived types on the device, whereas CCE adheres more closely to the OpenACC technical specification (version 3.2 at the time of writing) and requires manual allocation. If one holds many host variables, this process is cumbersome.

Listing 3 shows how Fypp is used to generate repetitive directives required for deep copying derived types. Without metaprogramming, each component of a derived type would need to be copied individually. The macro `@:SETUP_VFs(var1,var2,…,varN)` is used after allocating the vector fields `var1,var2,…,varN` on the host. This macro automatically traverses the derived type fields and inserts the correct enter data and create statements, ensuring that all vector fields and their subfields are allocated on the device with minimal manual coding. This macro also performs deep copies, reducing the number of lines required to initialize the variables at runtime.

Fypp simplifies memory management by wrapping `allocate` and `deallocate` into macros that allocate the variable and perform the neces-

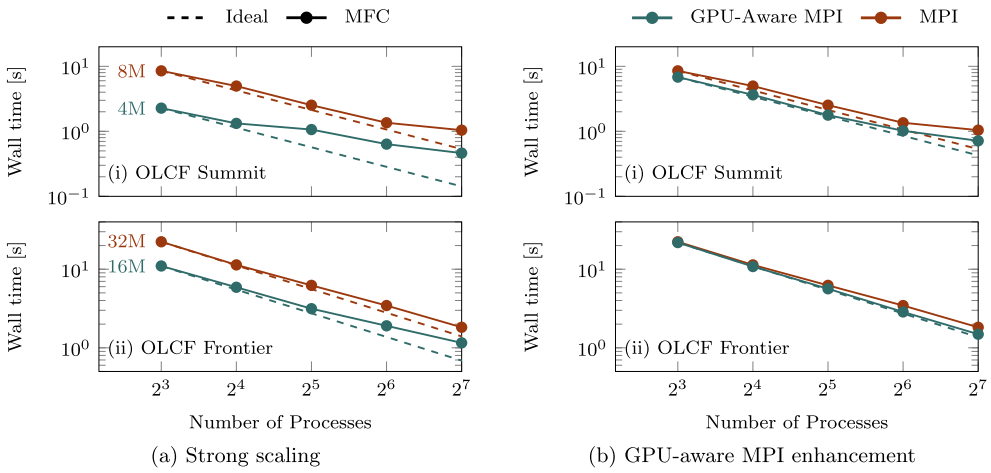


Fig. 18. Strong scaling performance on OLCF Summit and OLCF Frontier. The figures on the left show the strong scaling performance for a species problem without using GPU-aware MPI. The quantities in (b) indicate the number of grid cells per device in the base case. The figures in (b) show the improvement in scaling performance when GPU-aware MPI is used. In (b), an 8M grid point case is used for benchmarking MFC on (i) Summit, and a 32M grid point case is used for (ii) Frontier. The number of processes is commensurate with Summit's NVIDIA V100 GPUs or Frontier's AMD MI250X GPUs.

```
#:def SETUP_VFs(*args)
block
    integer :: macros_setup_vfs_i
    #:for arg in args
        $:GPU_ENTER_DATA(copyin='[' + arg + ']')
        $:GPU_ENTER_DATA(copyin='[' + arg + '%vf]')
        if (allocated(${arg}$%vf)) then
            do macros_setup_vfs_i = lbound(${arg}$%vf, 1),
                 ubound(${arg}$%vf, 1)
                if (associated(${arg}$%vf(macros_setup_vfs_i)%sf))
                     then
                    $:GPU_ENTER_DATA(copyin='[' + arg +
                         '%vf(macros_setup_vfs_i)]')
                    $:GPU_ENTER_DATA(copyin='[' + arg +
                         '%vf(macros_setup_vfs_i)%sf]')
                end if
            end do
        end if
    #:endfor
end block
#:enddef
```

Listing 3. A Fortran + Fypp macro that manually deep copies derived types from the host to the device. In the loop, Fypp expands `@:SETUP_VFs(var1, var2, ...)` into repeated OpenACC or OpenMP directives. `$:GPU_ENTER_DATA(copyin='[]')` reduces to `!$acc enter data copyin` or `!$omp target enter data copyin` depending on the desired GPU offload too. For each variable, the macro first copies the variable and its vector field array to the device. If the array is allocated, the loop iterates over its bounds, and if any scalar field pointers are associated, it performs further deep copies. This ensures a complete mirror of the host data structure on the GPU.

```
#:def ALLOCATE(*args)
    #:set allocated_variables = ', '.join(args)
    allocate (${allocated_variables}$)
    #:set cleaned = strip_parenthesis(allocated_variables)
    #:set joined = ', '.join(cleaned)
    $:GPU_ENTER_DATA(create='[' + joined + ']')
#:enddef ALLOCATE

#:def DEALLOCATE(*args)
    #:set allocated_variables = ', '.join(args)
    $:GPU_EXIT_DATA(delete='[' + allocated_variables + ']')
    deallocate (${allocated_variables}$)
#:enddef DEALLOCATE
```

Listing 4. Fypp macros for automatically allocating and deallocating memory with the associated directives for GPU memory management. `$:GPU_ENTER_DATA(create='[]')` reduces to `!$acc enter data create` or `!$omp target enter data create` and `$:GPU_EXIT_DATA(delete='[]')` reduces to `!$acc exit data delete` or `!$omp target exit data delete` depending on the desired GPU offload tool.

```
#:if CASE_OPTIMIZATION
    integer, parameter :: var = ${var}$
#:else
    integer :: var
#:endif
```

Listing 5. Fortran + Fypp code the declaration of configuration-specific parameters for compile-time optimization.

sary data movement required when using OpenACC or OpenMP for GPU offloading. Listing 4 shows this abstraction. A call such as `@:ALLOCATE(u, v, w)` expands to allocate the three variables and register them on the GPU device. The code length is further reduced by using Fypp to abstract away repetitive code via preprocessor loops [99].

Metaprogramming sets problem parameters as constant parameters in case files. This option is called *case optimization*, enabled via the flag `--case-optimization` at run time. Case optimization reduces register pressure by providing the compiler with the parameters of a configuration. On CPUs, specifying the problem parameters at compile time results in a two-fold speedup of the code. On GPUs, case optimization decreases run time by about a factor of ten.

Listing 5 demonstrates case optimization: when the option is enabled, configuration parameters are converted into compile-time constants (parameter). Otherwise, they remain runtime variables. This small change enables the compiler to perform constant folding and loop unrolling, thereby reducing register pressure and improving performance. Specifically, variables are prescribed in a file with the Fypp code `#:set var = val`. This file is included in the Fortran source code; compile-time-known parameters are declared using the Fypp logic of Listing 5.

In summary, Fypp-style metaprogramming enables MFC to automate deep copies of complex data structures, simplify memory management with consistent macros, and facilitate compile-time specialization of problem parameters. These techniques reduce boilerplate code, improve maintainability, and deliver performance improvement on CPUs and GPU devices.

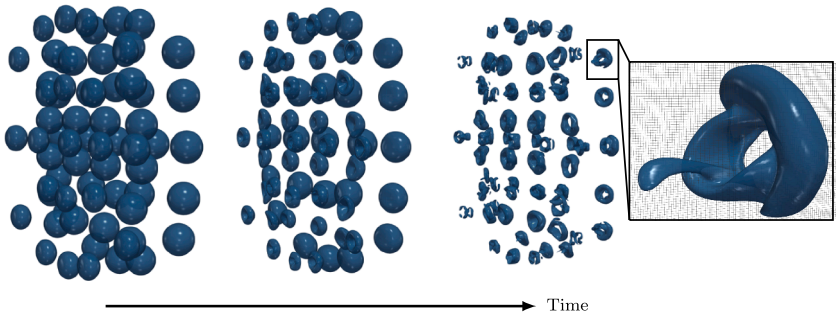
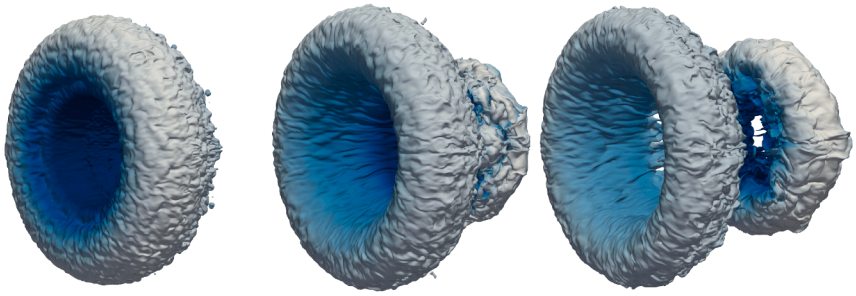


Fig. 19. Shock-induced collapse of a cloud of 75 air bubbles in water. The $\alpha = 0.5$ contour is shown at increasing points in time from left to right. The magnified view shows the mesh resolution around one of the collapsing bubbles.



(a) $t^* = 1.50$ (b) $t^* = 3.25$ (c) $t^* = 5.00$

Fig. 20. Shock-induced collapse of a helium bubble in air. The $\alpha = 0.5$ contour, representing the material interface, is shown at dimensionless times as labeled. The isosurface is colored by its velocity magnitude, with darker colors corresponding to higher velocities. The initial droplet has 640 grid cells across its diameter in each coordinate direction.

4.3.7. Code generation for efficient reactions and thermodynamics

MFC now simulates chemically reacting flows. We accommodate this feature via combustion routines that evaluate the chemical source terms and species thermodynamics. Comprehensive libraries, such as Cantera, provide such routines. However, these exist in a different compilation unit from the flow solver, so they cannot easily be offloaded via the OpenACC directive-based strategy of Section 4.3.1. We address this challenge through code generation, producing a computational representation of thermochemistry at the same level of abstraction as MFC's compressible flow solver. This strategy obviates the need to optimize comprehensive combustion libraries at link time. MFC uses Pyrometheus [64,100] for code generation, generating thermochemistry code in OpenACC-decorated Fortran.

Pyrometheus is based on a symbolic representation of the combustion formulation that takes mechanism parameters from Cantera. The symbolic representation is mapped to a user-specified target language. For Fortran, the generated code is a module that can be easily integrated and offloaded with MFC. The mapping process decorates the routines with OpenACC statements for GPU offloading, ensuring that the generated code does not require post-hoc modifications. The generated code unrolls inner loops over species and reactions and prepares compile-time constants such as Arrhenius parameters. This optimization meaningfully reduces kernel runtime [100]. In a similar vein to case optimization of Section 4.3.6, Pyrometheus assigns compile-time-known bounds to the arrays in the generated code, enabling the optimization of register allocations and reducing runtime by approximately a factor of 5 for GPU kernels.

Pyrometheus is verified against Cantera, although it is thoroughly tested itself. The testing suite is language-agnostic, so the generated Fortran code is as accurate as a C-based alternative. The generated code is mechanism-specific, but because code generation is faster than simulation time, multiple mechanisms can be preprocessed. For MFC, we generate code for established combustion mechanisms for hydrogen [69] and methane combustion [101], though we are not limited to these.

5. Example simulations

Example simulations using MFC 5.0 are included below. Many of these could not be conducted using prior MFC releases, or would have been prohibitively expansive, as MFC 5.0 has improved distributed computation and compiler and non-NVIDIA GPU hardware support has been added with MFC 5.0.

5.1. Shock–bubble-cloud interaction

The first example simulation shows the interaction between a cloud of 75 randomly placed 3 mm bubbles and a shockwave in water. The computational domain is $[-9D, 9D] \times [0, 10D] \times [0, 10D]$ before grid stretching and is discretized as $(N_x, N_y, N_z) = (2000, 1000, 1000)$, or about 2B grid points. The grid points are stretched away from the bubbles to avoid boundary effects via

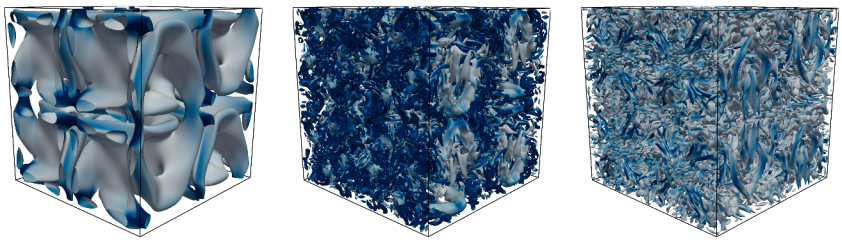
$$x_{\text{stretch}} = x + \frac{x}{a_x}\left[\log\left[\cosh\left(\frac{a_x(x - x_a)}{L}\right)\right] + \log\left[\cosh\left(\frac{a_x(x - x_b}{L}\right)\right] - 2\log\left[\cosh\left(\frac{a_x(x_b - x_a}{2L}\right)\right]\right],$$

where a_x is the stretching parameter, L is the domain length, and x_a and x_b control the stretching location.

Fig. 19 shows the $\alpha = 0.5$ isosurface at increasing points in time from left to right. The detail view shows the grid resolution around one of the collapsing bubbles. The initial condition has 100 uniform-sized grid points across each bubble diameter. The simulation was performed using 1024 AMD MI250X GCDs (512 AMD MI250X GPUs) on OLCF Frontier and completed in approximately 30 min.

5.2. Shock–bubble interaction

The second example simulation shows the interaction between a helium bubble in air impinged by a shock wave. The computational domain has spatial extents $[0, 20D] \times [-5D, 5D] \times [-5D, 5D]$ before grid stretching



(a) $t/t_c = 4$ (b) $t/t_c = 12$ (c) $t/t_c = 20$

Fig. 21. The isosurface with zero Q-criterion of a Re = 1600, Ma = 0.1 Taylor–Green vortex at dimensionless times t/t_c as labeled. The isosurface is colored by the vorticity magnitude, with darker colors corresponding to higher vorticity.

Fig. 22. A rendering showing the injection and fluid dynamics of three Ma = 1.5 jets into quiescent air.

and is discretized with $(N_x, N_y, N_z) = (3200, 1600, 1600)$, which is over 8B grid points. The domain boundaries are moved far away from the bubble to avoid boundary effects using grid stretching (described in Section 5.1). Fig. 20 shows the $\alpha = 0.5$ isosurface, which represents the interface between air and helium, at dimensionless times $t^* = tU/D = 1.5$, 3.25, and 5.0 for shock speed U and bubble diameter D. The isosurface is colored by velocity magnitude, with darker colors corresponding to higher velocities. The bubble is resolved with 640 grid cells in its initial diameter. The simulation was performed using 144 NVIDIA H200 GPUs in 16 h.

5.3. Taylor–Green vortex

The third example simulation is a Re = 1600, Ma = 0.1 Taylor–Green vortex. The initial condition follows that of Hillewaert [102]. The computational domain is a cube with side lengths of $L = 2\pi$ and is discretized with $N_x = N_y = N_z = 1600$, resulting in approximately 4 billion grid points. Fig. 21 shows the isosurface with zero Q-criterion colored by vorticity magnitude at dimensionless times $t/t_c = 4$, 12, and 20, with darker colors corresponding to higher vorticity magnitudes. The convective timescale t_c is defined as $t_c = L/(c_0 \mathrm{Ma})$, where c_0 is the free stream speed of sound. The simulation was performed using 144 NVIDIA H200 GPUs in 28 h.

5.4. Supersonic gas injection and plume dynamics

Three jets inject into the domain via Dirichlet boundary conditions. The domain has size $[0, 16D] \times [0, 10D] \times [0, 10D]$ where D is the jet diameter and the first dimension is the mean flow direction (horizontal in Fig. 22). The domain is discretized into a $600 \times 400 \times 400$ grid of cells. The jets have an incoming velocity (V_{jet}), pressure, and density that satisfy the Mach 1.5 normal shock conditions; they are organized in an equilateral triangle. The jet Reynolds number is $\mathrm{Re} = \rho V_{\mathrm{jet}} D/\mu = 2.5 \times 10^6$.

The simulation is executed until dimensionless time $40D/(\mathrm{Ma}\, c)$, where Ma is the Mach number and c is the ambient speed of sound. The last time step is visualized in Fig. 22. The visualization shows a volume rendering of the injected fluid after it has meaningfully intruded into the domain.

5.5. Validation and verification

The presented examples above are for demonstration of simulation capability and integrity. The shock–bubble dynamics cases of Sections 5.1 and 5.2, including the model and numerics, were validated at the scale of single-bubble dynamics by Bryngelson et al. [1] and the cited art within. The numerics themselves have been validated in prior works, such as Johnsen and Colonius [6]. The Taylor–Green vortex of Section 5.3 is validated against standard turbulence statistics. The jet case in Section 5.4 was verified through a grid study, but is primarily presented as an example of computational readiness.

6. Solvers with some shared capabilities

Other open-source codes for CFD exist. These and their relative trade-offs compared to MFC 5.0 are briefly discussed here. URANOS-2.0 (De Vanna and Baldan [103]) is a modern Fortran code that uses OpenACC for GPU offloading to NVIDIA and AMD GPUs. URANOS-2.0 incorporates some of MFC's shock-capturing capabilities and features compressible turbulence models. STREAmS-2 (Sathyanarayana et al. [104]) is also a Fortran code for compressible flow, utilizing CUDA Fortran and hipFORT to offload computations to NVIDIA and AMD GPUs. There is also an OpenMP port of STREAmS-2 that makes it portable to Intel GPUs. AFiD-GPU of Zhu et al. [105] is similar in this context. Neko (Jansson et al. [106]) is a particularly modern object-oriented solver that uses multiple levels of abstraction for offloading to GPUs as well as more extensive extensions for vector processors and FPGAs. The above solvers have offloading capabilities similar to those of MFC, although their physical models and sophistication differ. MFC offers a broad range of modeling capabilities and numerical methods for simulating multi-phase and multi-species flows.

7. Conclusion

Since MFC 3.0 [1], the code has transitioned from a specialized research code to an exascale-capable framework for multiphysics and multiphase flows. The implementation of physical models, numerical methods, software infrastructure, and high-performance computing tools embodies this transition.

Version 5.0 introduces six phase-change formulations for vapor–liquid systems and treatments of reacting flows. The implementation extends to non-polytropic sub-grid bubble dynamics models capable of representing sophisticated bubble dynamic processes. Hypo- and hyper-elastic material treatments are now available for solids under large strain rates. A generalized surface tension framework is now included, which is both conservative and numerically robust in our studies.

Numerical advancements accompany MFC's extended feature set. For shock-capturing, the implementation of TENO, WENO-Z, and

higher-order reconstructions provides methods that are well-established to reduce the numerical dissipation errors inherent in conventional, lower-order schemes [80–82,107]. The Strang-split particle solver with the adaptive time stepping algorithm was implemented to improve numerical efficiency for stiff sub-grid dynamics while ensuring second-order temporal accuracy [83,84]. The known inaccuracy of Riemann solvers in the low-Mach number limit [86] is addressed with two implemented treatments that reduce numerical dissipation, following the approaches of Thornber et al. [87] and Chen et al. [88]. A ghost-cell immersed boundary method is now available. This method supports complex 2D and 3D geometries that can be imported as level sets or STL files.

Software infrastructure improvements deliver performance portability across modern architectures. Continuous integration pipelines enforce rigorous quality control through over 300 regression tests, including reproducibility checks across hardware platforms and compilers.

Exascale readiness is demonstrated through multiple performance tests. OpenACC implementations and GPU-aware MPI throughout the codebase enable state-of-the-art strong scaling efficiency on AMD MI250X platforms. Metaprogramming enables a speedup of nearly 10 times over the baseline. Ideal weak scaling behavior is observed for current exascale machines, OLCF Frontier and LLNL El Capitan, which use AMD MI250X GPUs and MI300A APUs, and large-scale NVIDIA-based GPU machines like OLCF Summit and LLNL Lassen.

CRediT authorship contribution statement

Benjamin Wilfong: Writing – original draft, Visualization, Software, Investigation, Data curation, Conceptualization; **Henry A. Le Berre:** Software, Methodology, Formal analysis; **Anand Radhakrishnan:** Software, Methodology, Investigation, Conceptualization; **Ansh Gupta:** Writing – original draft, Visualization, Software; **Daniel J. Vickers:** Writing – review & editing, Methodology, Investigation, Formal analysis; **Diego Vaca-Revelo:** Writing – original draft, Software, Methodology; **Dimitrios Adam:** Writing – original draft, Software, Formal analysis, Data curation; **Haocheng Yu:** Writing – original draft, Visualization, Methodology, Formal analysis; **Hyeoksu Lee:** Writing – original draft, Software, Methodology; **Jose Rodolfo Chreim:** Writing – original draft, Software, Formal analysis; **Mirelys Carcana Barbosa:** Writing – original draft, Methodology; **Yanjun Zhang:** Writing – original draft, Validation, Software, Methodology, Investigation; **Esteban Cisneros-Garibay:** Writing – review & editing, Writing – original draft, Supervision, Software, Project administration, Methodology, Investigation; **Aswin Gnanaskandan:** Writing – review & editing, Writing – original draft, Supervision; **Mauro Rodriguez Jr:** Writing – original draft, Supervision; **Reuben D. Budiardja:** Supervision, Software, Resources, Conceptualization; **Stephen Abbott:** Supervision, Software, Project administration, Investigation; **Tim Colonius:** Supervision; **Spencer H. Bryngelson:** Writing – review & editing, Writing – original draft, Supervision, Software, Resources, Project administration, Funding acquisition, Formal analysis, Conceptualization.

Statement of data availability

MFC is available in perpetuity under the MIT license at github.com/MFlowCode/MFC.

Data availability

Data links are provided in the manuscript itself.

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.

Acknowledgments

The authors gratefully acknowledge OLCF Frontier Hackathons and Open Hackathons, the fruitful support of numerous MFC contributors, hackathon mentors, and vendor experts, of whom there are too many to name individually. SHB acknowledges support from the U.S. Department of Defense, Office of Naval Research under grant numbers N00014-22-1-2519 and N00014-24-1-2094, the Army Research Office under grant number W911NF-23-10324, the Department of Energy under DOE DE-NA0003525 (Sandia National Labs, subcontract), the Oak Ridge Associated Universities (ORAU) Ralph E. Powe Junior Faculty Enhancement Award, and hardware gifts from NVIDIA and AMD. MRJ acknowledges support from the U.S. Department of Defense under the DEPSCoR program Award No. FA9550-23-1-0485 (PM Dr. Timothy Bentley) and the U.S. National Science Foundation (NSF) under Grant No. 2232427. AG acknowledges support from the U.S. National Science Foundation (NSF) under Grants CBET 2,301,721 and CBET 2301709. Some computations were also performed on the Tioga, Tuolumne, and El Capitan computers at Lawrence Livermore National Laboratory's Livermore Computing facility. This research also used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725 (allocation CFD154, PI Bryngelson). This work used Delta and DeltaAI at the National Center for Supercomputing Applications and Bridges2 at the Pittsburgh Supercomputing Center through allocations PHY210084 and PHY240200 (PI Bryngelson) from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.cpc.2026.110055

References

[1] S.H. Bryngelson, K. Schmidmayer, V. Coralic, K. Maeda, J. Meng, T. Colonius, MFC: An open-source high-order multi-component, multi-phase, and multi-scale compressible flow solver, Comput. Phys. Commun. 266 (2021) 107396.
[2] A.K. Kapila, R. Menikoff, J.B. Bdzil, S.F. Son, D.S. Stewart, Two-phase modeling of deflagration-to-detonation transition in granular materials: reduced equations, Phys. Fluids 13 (10) (2001) 3002–3024.
[3] G. Allaire, S. Clerc, S. Kokh, A five-equation model for the simulation of interfaces between compressible fluids, J. Comput. Phys. 181 (2) (2002) 577–616.
[4] R. Saurel, F. Petitpas, R.A. Berry, Simple and efficient relaxation methods for interfaces separating compressible fluids, cavitating flows and shocks in multiphase mixtures, J. Comput. Phys. 228 (5) (2009) 1678–1712.
[5] E. Johnsen, Numerical Simulations of Non-Spherical Bubble Collapse with Applications to Shockwave Lithotripsy, Ph.D. thesis, California Institute of Technology, 2008.
[6] E. Johnsen, T. Colonius, Implementation of WENO schemes in compressible multicomponent flow problems, J. Comput. Phys. 219 (2) (2006) 715–732.
[7] V. Coralic, T. Colonius, Finite-volume WENO scheme for viscous compressible multicomponent flows, J. Comput. Phys. 274 (2014) 95–121.
[8] J.C. Meng, Numerical Simulations of Droplet Aerobreakup, Ph.D. thesis, California Institute of Technology, 2016.
[9] K. Maeda, T. Colonius, Eulerian–Lagrangian method for simulation of cloud cavitation, J. Comput. Phys. 371 (2018) 994–1017.
[10] K. Maeda, T. Colonius, A source term approach for generation of one-way acoustic waves in the Euler and Navier–Stokes equations, Wave Motion 75 (2017) 36–49.
[11] B. Wilfong, A. Radhakrishnan, H. Le Berre, D. Vickers, T. Prathi, N. Tselepidis, B. Dorschner, R. Budiardja, B. Cornille, S. Abbott, F. Schäfer, S. Bryngelson, Simulating many-engine spacecraft: exceeding 1 quadrillion degrees of freedom via information geometric regularization, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '25, Association for Computing Machinery, New York, NY, USA, 2025, pp. 14–24.
[12] A. Services, ACCESS: advanced cyberinfrastructure coordination ecosystem, 2025, (https://access-ci.org). Accessed: 2025-02-14.
[13] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G.D. Peterson, R. Roskies, J.R. Scott, N. Wilkins-Diehr, XSEDE: Accelerating scientific discovery, Comput. Sci. Eng. 16 (5) (2014) 62–74.

[14] Pittsburgh Supercomputing Center (PSC), Bridges: a HPC resource for accelerating computational research, 2015, (https://www.psc.edu/bridges). Funded by the National Science Foundation under Grant No. OCI-1445606. Retired in 2021.

[15] S.T. Brown, P. Buitrago, E. Hanna, S. Sanielevici, R. Scibek, Bridges-2: transforming research with advanced computational capabilities, Pittsburgh Supercomput. Center 35 (2021) 1–4.

[16] San Diego Supercomputer Center (SDSC), Comet: a petascale supercomputer for the long tail of science, 2015, (https://www.sdsc.edu/News%20Items/PR20170201_Comet_10k.html). Accessed: 2025-02-14.

[17] S. Strande, H. Cai, M. Tatineni, W. Pfeiffer, C. Irving, A. Majumdar, D. Mishin, R. Sinkovits, M. Norman, N. Wolter, T. Cooper, I. Altintas, M. Kandes, I. Perez, M. Shantharam, M. Thomas, S. Sivagnanam, T. Hutton, Expanse: computing without boundaries: architecture, deployment, and early operations experiences of a supercomputer designed for the rapid evolution in science and engineering, Pract. Exper. Adv. Res. Comput.(PEARC '21) (2021) 1–8. https://doi.org/10.1145/3437359.3465588

[18] X.C. Song, P. Smith, others, Anvil – system architecture and experiences from deployment and early user operations, in: Practice and Experience in Advanced Research Computing (PEARC '22), Association for Computing Machinery, New York, NY, USA, 2022, p. 23. https://doi.org/10.1145/3491418.3530766

[19] National Center for Supercomputing Applications (NCSA), Delta: advanced computing resource for science and engineering, 2022, (https://delta.ncsa.illinois.edu). Accessed: 2025-02-14.

[20] National Center for Supercomputing Applications (NCSA), DeltaAI: expanding AI-focused computing capacity at NCSA, 2023, (https://deltaai.ncsa.illinois.edu). Accessed: 2025-02-14.

[21] Texas Advanced Computing Center (TACC), Stampede: high-performance computing resource at TACC, 2013, (????). Supported by the National Science Foundation under Grant No. OCI-1134872.

[22] N.A. Nystrom, et al., Stampede2: the evolution of an XSEDE supercomputer, Proc. Pract. Exper. Adv. Res. Comput. (PEARC '17) (2017). https://doi.org/10.1145/3093338.3093385

[23] Texas Advanced Computing Center (TACC), Stampede3: advanced supercomputing for open science research, 2024, (https://www.tacc.utexas.edu). Funded by the National Science Foundation under Award No. 2320757.

[24] Intel Corporation, Intel developer cloud: AI-optimized supercomputing platform, 2024, (https://console.cloud.intel.com/docs/index.html). Accessed: 2025-02-14.

[25] SPEC, SPEChpc, 2025, (https://www.spec.org/products). Accessed: 2025-02-14.

[26] B. Wilfong, A. Radhakrishnan, H.A. Le Berre, T. Prathi, S. Abbott, S.H. Bryngelson, Testing and benchmarking emerging supercomputers via the MFC flow solver, in: Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '25, Association for Computing Machinery, New York, NY, USA, 2025.

[27] N. Andrianov, G. Warnecke, The Riemann problem for the Baer–Nunziato two-phase flow model, J. Comput. Phys. 195 (2) (2004) 434–464.

[28] K. Schmidmayer, S.H. Bryngelson, T. Colonius, An assessment of multicomponent flow models and interface capturing schemes for spherical bubble dynamics, J. Comput. Phys. 402 (2020) 109080. https://doi.org/10.1016/j.jcp.2019.109080

[29] R. Menikoff, B.J. Plohr, The Riemann problem for fluid flow of real materials, Rev. Mod. Phys. 61 (1989) 75–130.

[30] K. Mohseni, T. Colonius, Numerical treatment of polar coordinate singularities, J. Comput. Phys. 157 (2) (2000) 787–795.

[31] A.K. Henrick, T.D. Aslam, J.M. Powers, Mapped weighted essentially non-oscillatory schemes: achieving optimal order near critical points, J. Comput. Phys. 207 (2) (2005) 542–567.

[32] E.F. Toro, The HLL and HLLC Riemann Solvers, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 293–311.

[33] K.W. Thompson, Time-dependent boundary conditions for hyperbolic systems, II, J Comput Phys 89 (2) (1990) 439–461.

[34] S. Gottlieb, C.-W. Shu, Total variation diminishing Runge–Kutta schemes, Math. Comput. 67 (221) (1998) 73–85.

[35] Y.-H. Tseng, J.H. Ferziger, A ghost-cell immersed boundary method for flow in complex geometry, J. Comput. Phys. 192 (2) (2003) 593–623.

[36] K.J. Weiler, Topological structures for geometric modeling (Boundary representation, manifold, radial edge structure), Ph.D. thesis, Rensselaer Polytechnic Institute, 1986.

[37] G. Strang, On the construction and comparison of difference schemes, SIAM J. Numer. Anal. 5 (1968) 506–517. https://doi.org/10.1137/0705041

[38] A. Zein, M. Hantke, G. Warnecke, Modeling phase transition for compressible two-phase flows applied to metastable liquids, J. Comput. Phys. 229 (2010) 2964–2998. https://doi.org/10.1016/j.jcp.2009.12.026

[39] T. Flåtten, A. Morin, S.T. Munkejord, On solutions to equilibrium problems for systems of stiffened gases, SIAM J. Appl. Math. 71 (1) (2011) 41–67.

[40] R. Saurel, F. Petitpas, R. Abgrall, Modelling phase transition in metastable liquids: application to cavitating and flashing flows, J. Fluid Mech. 607 (2008) 313–350. https://doi.org/10.1017/S0022112008002061

[41] J.R. Simões-Moreira, J.E. Shepherd, Evaporation waves in superheated dodecane, J. Fluid Mech. 382 (1999) 63–86. https://doi.org/10.1017/S0022112098003796

[42] O.L. Métayer, J. Massoni, R. Saurel, Modelling evaporation fronts with reactive Riemann solvers, J. Comput. Phys. 205 (2005) 567–610. https://doi.org/10.1016/j.jcp.2004.11.021

[43] D.Z. Zhang, A. Prosperetti, Ensemble phase-averaged equations for bubbly flows, Phys. Fluids 6 (9) (1994) 2956–2970. https://doi.org/10.1063/1.868122

[44] S.H. Bryngelson, K. Schmidmayer, T. Colonius, A quantitative comparison of phase-averaged models for bubbly, cavitating flows, Int. J. Multiphase Flow 115 (2019) 137–143. https://doi.org/10.1016/j.ijmultiphaseflow.2019.03.028

[45] S.H. Bryngelson, R.O. Fox, T. Colonius, Conditional moment methods for polydisperse cavitating flows, J. Comput. Phys. 477 (2023) 111917.

[46] S.H. Bryngelson, A. Charalampopoulos, T.P. Sapsis, T. Colonius, A Gaussian moment method and its augmentation via LSTM recurrent neural networks for the statistics of cavitating bubble populations, Int. J. Multiphase Flow 127 (2020) 103262.

[47] S.H. Bryngelson, T. Colonius, Simulation of humpback whale bubble-net feeding models, J. Acoust. Soc. Am. 147 (2) (2020) 1126–1135. https://doi.org/10.1121/10.0000746

[48] R. Menikoff, B.J. Plohr, The Riemann problem for fluid-flow of real materials, Rev. Mod. Phys. 61 (1) (1989) 75–130.

[49] A. Charalampopoulos, S.H. Bryngelson, T. Colonius, T.P. Sapsis, Hybrid quadrature moment method for accurate and stable representation of non-Gaussian processes and their dynamics, Philosoph. Trans. R. Soc. A 380 (2229) (2022).

[50] S.H. Bryngelson, Fast integration method for averaging polydisperse bubble population dynamics, Comput. Fluids 304 (2026) 106877.

[51] M.S. Plesset, A. Prosperetti, Bubble dynamics and cavitation, Ann. Rev. Fluid Mech. 9 (1) (1977) 145–185.

[52] R.O. Fox, F. Laurent, A. Vié, Conditional hyperbolic quadrature method of moments for kinetic equations, J. Comput. Phys. 365 (2018) 269–293.

[53] K. Ando, T. Colonius, C.E. Brennen, Improvement of acoustic theory of ultrasonic waves in dilute bubbly liquids, J. Acoust. Soc. Am. 126 (3) (2009) El69–El74.

[54] D. Fuster, T. Colonius, Modelling bubble clusters in compressible liquids, J. Fluid Mech. 688 (2011) 352–389.

[55] G. Altmeyer, E. Rouhaud, B. Panicaud, A. Roos, R. Kerner, M. Wang, Viscoelastic models with consistent hypoelasticity for fluids undergoing finite deformations, Mech. Time-Dependent Mater. 19 (3) (2015) 375–395. https://doi.org/10.1007/s11043-015-9269-5

[56] G. Altmeyer, B. Panicaud, E. Rouhaud, M. Wang, A. Roos, R. Kerner, Viscoelasticity behavior for finite deformations, using a consistent hypoelastic model based on Rivlin materials, Continuum Mech. Thermodyn. 28 (6) (2016) 1741–1758.

[57] M. Rodriguez, E. Johnsen, A high-order accurate five-equations compressible multiphase approach for viscoelastic fluids and solids with relaxation and elasticity, J. Comput. Phys. 379 (2019) 70–90.

[58] J.-S. Spratt, Numerical Simulations of Cavitating Bubbles in Elastic and Viscoelastic Materials for Biomedical Applications, Ph.D. thesis, California Institute of Technology, 2024.

[59] K. Kamrin, C.H. Rycroft, J.-C. Nave, Reference map technique for finite-strain elasticity and fluid–solid interaction, J. Mech. Phys. Solids 60 (11) (2012) 1952–1969.

[60] M.C. Barbosa, M. Rodriguez, Jr., J. Yang, High-fidelity numerical simulations of inertial microbubble collapse at a gel-water interface with finite elasticity and phase change, in: Center for Turbulence Research, Proceedings of the Summer Program, 2024, pp. 349–359.

[61] A. Aiyer, C. Meneveau, Coupled population balance and large eddy simulation model for polydisperse droplet evolution in a turbulent round jet, Phys. Rev. Fluids 5 (2020) 114305.

[62] Y. Ren, J. Cai, H. Pitsch, Theoretical single-droplet model for particle formation in flame spray pyrolysis, Energy Fuels 35 (2021) 1750–1759.

[63] W.R. Zeng, S.F. Li, W.K. Chow, Review on chemical reactions of burning poly(methyl methacrylate) PMMA, J. Fire Sci. 20 (2002) 401–433.

[64] E. Cisneros-Garibay, C. Pantano, J.B. Freund, Flow and combustion in a supersonic cavity flameholder, AIAA J. 60 (2022) 1–12.

[65] A.N. Al-Khateeb, J.M. Powers, S. Paolucci, A.J. Sommese, J.A. Diller, J.D. Hauenstein, J.D. Mengers, One-dimensional slow invariant manifolds for spatially homogenous reactive systems, J. Chem. Phys. 131 (2009).

[66] B.J. McBride, NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species, National Aeronautics and Space Administration, John H. Glenn Research Center, 2002.

[67] W. Gardiner, Combustion Chemistry, Springer New York, 1984.

[68] P.J. Martínez-Ferrer, R. Buttay, G. Lehnasch, A. Mura, A detailed verification procedure for compressible reactive multicomponent Navier–Stokes solvers, Comput.Fluids 89 (2014) 88–110.

[69] P. Saxena, F.A. Williams, Testing a small detailed chemical-kinetic mechanism for the combustion of hydrogen and carbon monoxide, Combust. Flame 145 (2006) 316–323.

[70] K. Schmidmayer, F. Petitpas, E. Daniel, N. Favrie, S. Gavrilyuk, A model and numerical method for compressible flows with capillary effects, J. Comput. Phys. 334 (2017) 468–496. https://doi.org/https://doi.org/10.1016/j.jcp.2017.01.001

[71] W. Dai, P.R. Woodward, Extension of the piecewise parabolic method to multidimensional ideal magnetohydrodynamics, J. Comput. Phys. 115 (2) (1994) 485–514.

[72] T. Miyoshi, K. Kusano, A multi-state HLL approximate Riemann solver for ideal magnetohydrodynamics, J. Comput. Phys. 208 (1) (2005) 315–344.

[73] G. Tóth, The $\nabla \cdot B = 0$ constraint in shock-capturing magnetohydrodynamics codes, J. Comput. Phys. 161 (2) (2000) 605–652.

[74] A. Mignone, G. Bodo, An HLLC Riemann solver for relativistic flows–II. magnetohydrodynamics, Mon. Not. R. Astron. Soc. 368 (3) (2006) 1040–1054.

[75] S. Pirozzoli, T. Colonius, Generalized characteristic relaxation boundary conditions for unsteady compressible flow simulations, J. Comput. Phys. 248 (2013) 109–126.

[76] M.B. Giles, Nonreflecting boundary conditions for Euler equation calculations, AIAA J. 28 (12) (1990) 2050–2058. https://doi.org/10.2514/3.10430

[77] G. Lodato, P. Domingo, L. Vervisch, Three-dimensional non-reflecting boundary conditions for direct and large-eddy simulation of compressible flows, J. Comput. Phys. 227 (10) (2008) 5105–5143. https://doi.org/10.1016/j.jcp.2008.01.027

[78] C.K.W. Tam, Z. Dong, Radiation and outflow boundary conditions for direct computation of acoustic and flow disturbances in a nonuniform mean flow, J. Comput. Phys. 129 (1) (1996) 164–180. https://doi.org/10.1006/jcph.1996.0237
[79] S. Jaensch, C. Sovardi, W. Polifke, Time-domain impedance boundary conditions for compressible viscous flow, J. Comput. Phys. 314 (2016) 145–159. https://doi.org/10.1016/j.jcp.2016.03.026
[80] R. Borges, M. Carmona, B. Costa, W.S. Don, An improved weighted essentially non-oscillatory scheme for hyperbolic conservation laws, J. Comput. Phys. 227 (6) (2008) 3191–3211.
[81] L. Fu, X.Y. Hu, N.A. Adams, A family of high-order targeted ENO schemes for compressible-fluid simulations, J. Comput. Phys. 305 (2016) 333–359.
[82] C.-W. Shu, Essentially Non-Oscillatory and Weighted Essentially Non-Oscillatory Schemes for Hyperbolic Conservation Laws, ICASE Report No. 97-65 NASA/CR-97-206253, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA, 1997.
[83] G. Strang, On the construction and comparison of difference schemes, SIAM J. Numer. Anal. 5 (3) (1968) 506–517.
[84] E. Hairer, S.P. Nørsett, G. Wanner, Solving Ordinary Differential Equations I. Non-stiff Problems, Springer, 1993.
[85] C.E. Brennen, Cavitation and Bubble Dynamics, Oxford University Press, 1995.
[86] H. Guillard, C. Viozat, On the behaviour of upwind schemes in the low Mach number limit, Comput. Fluids 28 (1) (1999) 63–86.
[87] B. Thornber, A. Mosedale, D. Drikakis, D. Youngs, R.J.R. Williams, An improved reconstruction method for compressible flows with low Mach number features, J. Comput. Phys. 227 (10) (2008) 4873–4894.
[88] S.-S. Chen, J.-P. Li, Z. Li, W. Yuan, Z.-H. Gao, Anti-dissipation pressure correction under low Mach numbers for Godunov-type schemes, J. Comput. Phys. 456 (2022) 111027.
[89] P.M. Gresho, S.T. Chan, On the theory of semi-implicit projection methods for viscous incompressible flow and its implementation via a finite element method that also introduces a nearly consistent mass matrix. part 2: implementation, Int. J. Numer. Methods Fluids 11 (5) (1990) 621–659.
[90] S. Wienke, P. Springer, C. Terboven, D. an Mey, OpenACC — first experiences with real-World applications, in: C. Kaklamanis, T. Papatheodorou, P.G. Spirakis (Eds.), Euro-Par 2012 Parallel Processing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 859–870.
[91] OpenMP Architecture Review Board, OpenMP Application Programming Interface Version 5.2, OpenMP Architecture Review Board, 2021. https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf.
[92] B. Aradi, Fypp: python-powered fortran metaprogramming, 2025. GitHub Repository, https://github.com/aradi/fypp.
[93] NVIDIA Corporation, cuTENSOR: a high-performance CUDA library for tensor primitives, 2019. https://developer.nvidia.com/cutensor.
[94] I. Advanced Micro Devices, hipBLAS: high-performance basic linear algebra subprograms library, 2020. Github Repository, https://github.com/ROCmSoftwarePlatform/hipBLAS.
[95] NVIDIA Corporation, cuFFT: high-performance CUDA FFT library, 2007. https://developer.nvidia.com/cufft.
[96] I. Advanced Micro Devices, hipFFT: high-performance fast fourier transform library, 2020. Github Repository, https://github.com/ROCmSoftwarePlatform/hipFFT.
[97] M. Frigo, S.G. Johnson, The design and implementation of FFTW3, Proc. IEEE 93 (2) (2005) 216–231.
[98] W. Elwasif, S. Bastrakov, S.H. Bryngelson, M. Bussmann, S. Chandrasekaran, F. Ciorba, M.A. Clark, A. Debus, W. Godoy, N. Hagerty, J. Hammond, D. Hardy, J.A. Harris, O. Hernandez, B. Joo, S. Keller, P. Kent, H. Le Berre, D. Lebrun-Grandie, E. MacCarthy, V.G.M. Vergara, B. Messer, R. Miller, S. Oral, J.-G. Piccinali, A. Radhakrishnan, O. Simsek, F. Spiga, K. Steiniger, J. Stephan, J.E. Stone, C. Trott, R. Widera, J. Young, Early application experiences on a modern GPU-accelerated arm-based HPC platform, in: HPC Asia '23, International Workshop on Arm-based HPC: Practice and Experience (IWAHPCE), Singapore, 2023, pp. 35–49. https://doi.org/10.1145/3581576.3581621
[99] A. Radhakrishnan, H. Le Berre, B. Wilfong, J.-S. Spratt, M. Rodriguez Jr., T. Colonius, S.H. Bryngelson, Method for portable, scalable, and performant GPU-accelerated simulation of multiphase compressible flow, Comput. Phys. Commun. 302 (2024) 109238. https://doi.org/10.1016/j.cpc.2024.109238
[100] E. Cisneros-Garibay, H. Le Berre, S.H. Bryngelson, J.B. Freund, Pyrometheus: symbolic abstractions for XPU and automatically differentiated computation of combustion kinetics and thermodynamics,(2025), arXiv preprint arXiv:2503.24286.
[101] G.P. Smith, D.M. Golden, M. Frenklach, N.W. Moriarty, B. Eiteneer, M. Goldenberg, C.T. Bowman, R.K. Hanson, S. Song, W.C. Gardiner, V.V. Lissianski, Z. Qin, GRI-Mech 3.0, 1999, (http://www.me.berkley.edu/gri_mech/).
[102] K. Hillewaert, TestCase C3.5 - DNS of the transition of the Taylor–Green vortex, in: 1st International Workshop on High Order CFD Methods, 2013, pp. 1–5.
[103] F. De Vanna, G. Baldan, URANOS-2.0: Improved performance, enhanced portability, and model extension towards exascale computing of high-speed engineering flows, Comput. Phys. Commun. 303 (2024) 109285.
[104] S. Sathyanarayana, M. Bernardini, D. Modesti, S. Pirozzoli, F. Salvadore, High-speed turbulent flows towards the exascale: STREAmS-2 porting and performance,(2023), arXiv preprint arXiv 2304.05494 .
[105] X. Zhu, E. Phillips, V. Spandan, J. Donners, G. Ruetsch, J. Romero, R. Ostilla-Mónico, Y. Yang, D. Lohse, R. Verzicco, M. Fatica, R.J.A.M. Stevens, AFiD-GPU: a versatile Navier–Stokes solver for wall-bounded turbulent flows on GPU clusters, Comput. Phys. Commun. 229 (2018) 199–210.
[106] N. Jansson, M. Karp, A. Podobas, S. Markidis, P. Schlatter, Neko: a modern, portable, and scalable framework for high-fidelity computational fluid dynamics, Comput. Fluids 275 (2024) 106243.
[107] R.J. LeVeque, Finite Volume Methods for Hyperbolic Problems, 31, Cambridge university press, 2002.