Simulating many-engine spacecraft: Exceeding 1 quadrillion degrees of freedom via information geometric regularization

Benjamin Wilfong
School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, GA, USA
bwilfong3@gatech.edu

Anand Radhakrishnan
School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, GA, USA
aradhakr34@gatech.edu

Henry Le Berre
School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, GA, USA
henryleberre@gatech.edu

Daniel Vickers
School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, GA, USA
dvickers6@gatech.edu

Tanush Prathi
School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, GA, USA
tprathi3@gatech.edu

Nikolaos Tselepidis
NVIDIA
Zurich, Switzerland
ntselepidis@nvidia.com

Benedikt Dorschner
NVIDIA
Zurich, Switzerland
bdorschner@nvidia.com

Reuben Budiardja*
Oak Ridge National Laboratory
Oak Ridge, TN, USA
budiardjard@ornl.gov

Brian Cornille
AMD
Naperville, IL, USA
brian.cornille@amd.com

Stephen Abbott
Hewlett Packard Enterprise (HPE)
St. Paul, MN, USA
stephen.abbott@hpe.com

Florian Schäfer†
Courant Institute of Mathematical Sciences
New York University
New York, NY, USA
florian.schaefer@nyu.edu

Spencer Bryngelson†
School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, GA, USA
shb@gatech.edu

Abstract

We present an optimized implementation of the recently proposed information geometric regularization (IGR) for unprecedented scale simulation of compressible fluid flows applied to multi-engine spacecraft boosters. We improve upon state-of-the-art computational fluid dynamics (CFD) techniques in terms of computational cost, memory footprint, and energy-to-solution metrics. Unified memory on coupled CPU–GPU or APU platforms increases problem size with negligible overhead. Mixed half/single-precision storage and computation are used on well-conditioned numerics. We simulate flow at 200 trillion grid points and 1 quadrillion degrees of freedom, exceeding the current record by a factor of 20. A factor of 4 wall-time speedup is achieved over optimized baselines. Ideal weak scaling is observed on OLCF Frontier, LLNL El Capitan, and CSCS Alps using the full systems. Strong scaling is near ideal at extreme conditions, including 80% efficiency on CSCS Alps with an 8 node baseline and stretching to the full system.

CCS Concepts

• **Applied computing** → **Physics**; • **Computing methodologies** → **Massively parallel and high-performance simulations**.

Keywords

CFD, regularization, exascale, unified memory

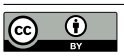
ACM Reference Format:
Benjamin Wilfong, Anand Radhakrishnan, Henry Le Berre, Daniel Vickers, Tanush Prathi, Nikolaos Tselepidis, Benedikt Dorschner, Reuben Budiardja, Brian Cornille, Stephen Abbott, Florian Schäfer, and Spencer Bryngelson. 2025. Simulating many-engine spacecraft: Exceeding 1 quadrillion degrees of freedom via information geometric regularization. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3712285.3771783

1 Justification for ACM Gordon Bell Prize

Largest compressible computational fluid dynamics simulation, exceeding 1 quadrillion degrees of freedom and a factor of 20 beyond

*This manuscript has been authored in part by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan.

†Equal contribution



SC '25, St Louis, MO, USA

ACM ISBN 979-8-4007-1466-5/25/11
https://doi.org/10.1145/3712285.3771783

state-of-the-art. Enabled by a large-scale optimized use of inviscid (information geometric) regularization. Time-to-solution and energy-to-solution decrease by up to a factor of 4 and 5.4 in double precision, respectively, with greater decreases in single and mixed precision.

2 Performance Attributes

Table 1: Summary of Performance Attributes

Performance attribute	This submission
Category of Achievement	Scalability, problem size, time-to-solution
Type of Method Used	Finite volume w/ information regularization
Results Reported Based On	Whole application including I/O
Precision Reported	Mixed (FP16/32), Single, Double
System Scale	Full system
Measurement Mechanism	Timers, FLOPs, Power management counters

3 Overview of the Problem

The 21st century witnesses a "new space race" [19] driven by private companies replacing government agencies in providing launch services. The resulting drop in launch cost enabled many business models, ranging from satellites to space manufacturing [10]. The new emphasis on cost efficiency motivated innovations in rocket design, such as reusing rocket stages and leveraging economies of scale.

An example of such innovations is the use of many-engine rockets. Five large F-1 engines powered the first stage of the Saturn V rocket. Their size was not constrained by road width or transportation logistics but dictated by engineering requirements. Instead, SpaceX Super Heavy, the first stage of Starship, is powered by 33 smaller Raptor engines. This has multiple advantages. The economy of scale benefits the production of a larger number of smaller engines, and their relatively compact size allows their transport via standard road infrastructure. The multitude of engines also provides a degree of redundancy, and a small number of engine failures can be compensated for without risking mission success. Upon reuse, the defective engines can be replaced. Further, when landing the Super Heavy after flight, thrust can be reduced by turning off individual engines.

Large arrays of small engines create new design challenges. The exhaust plumes of densely packed engines can interact, propelling hot gas toward the rocket base and heating it. This so-called base heating can cause mission failure [18], mandating heat shields on the rocket base that increase weight and cost.

Mitigating base heating most cost-effectively, in terms of both dollars and weight, requires understanding the mechanism by which engine exhaust is reflected towards the rocket and identifying which parts are most affected. A key difficulty in experimental approaches to understanding plume recirculation is that it depends on numerous parameters, including varying ambient pressure as the rocket traverses the atmosphere and engine thrust vectoring for steering. A detailed flow field characterization under a broad range of conditions is only feasible with numerical simulations. They even allow probing the impacts of changes to the rocket design. Prior work on the numerical simulation of interacting rocket plumes was limited to small numbers (up to 7) of rocket engines and limited resolution (up to ≈ 10 million grid points) [18, 21, 27]. Our work addresses this shortcoming by utilizing the latest flagship exascale systems, leveraging their hardware design and coupling novel algorithms, computational methods, and the optimizations they enable. With this, we can simulate the interaction of rocket engine plumes at unprecedented scales.

As a model, we represent the exhaust via the compressible Navier–Stokes equations

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0, \tag{1}$$

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u} + p\mathbf{I} - \boldsymbol{\tau}) = \mathbf{0}, \tag{2}$$

$$\frac{\partial E}{\partial t} + \nabla \cdot [(E + p)\mathbf{u} - \mathbf{u} \cdot \boldsymbol{\tau}] = 0, \tag{3}$$

with the ideal gas law equation of state

$$p = (\gamma - 1)\rho e, \quad \text{for} \quad e \coloneqq E/\rho - \|\mathbf{u}\|^2/2 \tag{4}$$

and, for μ, ζ denoting shear, bulk viscosity, constitutive law

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) + (\zeta - 2\mu/3)\, \delta_{ij} \frac{\partial u_i}{\partial x_j}. \tag{5}$$

Equations (1) to (3) describe the conservation of mass, momentum, and energy, and $\boldsymbol{\tau}$ is the viscous stress tensor and p the pressure. Tracking the mixture ratios of different gases and fluids, as well as their chemical reactions, is a natural extension of the demonstration. For this work and its implications, we focus on eqs. (1) to (5).

Summary of Contributions

- Information geometric regularization foregoes nonlinear viscous shock capturing, enabling linear off-the-shelf numerical schemes and sequential summation of right-hand side contributions.
- Unified addressing on tightly coupled CPU–GPU and APU platforms increases total problem size with negligible performance hit.
- FP32 compute and FP16 storage further reduce memory use while remaining numerically stable, enabled by the algorithm's well-conditioned numerics.
- Reduce memory footprint 25-fold over state-of-the-art.
- Improve time and energy-to-solution factors of 4 and 5.4, compared to an optimized implementation of state-of-the-art methods.
- First CFD simulation exceeding 200T grid points and 1 quadrillion degrees of freedom, improving on previous largest simulations by a factor of 20.

Example simulation result

Figure 1 shows a visualization for a simulation run on CSCS Alps. The visualization shows the interactions between an array of 33 Mach 10 rocket engines organized in an array inspired by that of the SpaceX Super Heavy, each resolved by 600 grid cells across its outlet. The high resolution allows fine-scale details to be represented despite the high Mach number. The simulation of fig. 1 uses

Code available at https://github.com/MFlowCode/MFC

Figure 1: Simulation results showing the interacting plumes from an array of 33 thrusters in a configuration inspired by the SpaceX Super Heavy with 16.5 trillion degrees of freedom. (The thrusters themselves are for visualization purposes. We model them through inflow boundary conditions.)

a rectilinear grid of 3.3T cells and ran for 16 hours on 9.2K GH200s (2300 Alps nodes).

4 Current State of the Art

4.1 Shock capturing

Shock waves. Shock waves are the primary concern in high-speed compressible fluid dynamics simulations. They arise in numerous natural and man-made phenomena, including supernovae, air, and spacecraft. The velocity and density fields of compressible high-speed flows sharpen over time, eventually forming macroscopic discontinuities or shock waves. However, on the microscopic scale, the gas viscosity balances the steepening of shock waves, resulting in smooth profiles. In practice, this happens on the scale of the mean free path of gas particles, orders of magnitude smaller than the quantities of interest.

Computation with shocks: A multiscale problem. Higher-order numerical methods exploit the target solution's regularity to approximate it with smooth functions, most frequently polynomials. The mean-free path is many orders of magnitude shorter than a realistic computational grid spacing. Thus, shocks appear as discontinuities on the grid scale, and the direct application of higher-order methods leads to Gibbs–Runge oscillations and subsequent simulation failure. *Shock capturing* modifies either the equation or its discretization to obtain a well-behaved object on the grid scale. It amounts to coarse-graining the microscopic shock and correctly representing its macroscopic effects without resolution at the grid level. The resulting coarse-scale model should preserve smooth grid-scale oscillations due to turbulence, reactions, or acoustic effects.

Existing approaches. Artificial viscosity mitigates Gibbs–Runge oscillations at the cost of excessive dissipation of fine-scale features. To remedy the latter, numerous approaches apply artificial diffusivity *adaptively*, in the vicinity of the shock [9, 13, 17].

Realizing a viscous regularization is challenging in practice. A lack of viscosity creates spurious oscillations, while excessive viscosity dissipates the solution. Common choices, such as the localized artificial diffusivity (LAD) of [9], attempt to strike this balance. Viscous methods spread the shock over multiple grid points, but the resulting shock profile is not high-order smooth (fig. 2 (a,i)). This can still lead to Gibbs–Runge oscillations overcoming the regularization and destabilizing solutions. Increasing the shock width requires increasing the strength of the artificial viscosity, which smears true physical features critical to representing the flow (fig. 2, (b,i)). In the presence of sufficiently strong shocks, the required artificial viscosity affects the CFL numbers of the explicit time steppers considered state-of-the-art for such hyperbolic problems.

Limiters are an alternative to artificial viscosity. They adaptively lower the order of the numerical method at shocks [16, 26]. They are more robust but also risk dissipating fine-scale features. Riemann solvers aim to mitigate this problem but add computational cost [14, 22, 25].

4.2 GPU memory

The evolution of GPU memory over the past decades, in terms of capacity and bandwidth, has been highly beneficial for CFD applications, which exhibit low arithmetic intensity. Thus, performance is limited by memory bandwidth. The spatio-temporal scale separation in fluid flows requires a high resolution, necessitating large amounts of memory to be accessed at high speeds. Both NVIDIA and AMD have blurred the lines between GPU and CPU memory by introducing coherent CPU-to-GPU interfaces: 900 GB/s bidirectional bandwidth for the NVIDIA Grace Hopper and InfinityFabric at 8×72 GB/s for the AMD Trento+4 MI250X configuration of Frontier. These fast interconnects, along with technologies such as unified virtual memory (UVM), allow CPU memory for GPU computations and even (in the case of Grace Hopper) allow the GPU to saturate the host memory bandwidth. Furthermore, AMD introduced the MI300A in LLNL's El Capitan and Tuolumne, with a single physical HBM pool accessed by both CPU and GPU devices. This strategy eliminates concerns about device and host pointers or memory storage locations.

4.3 Floating point computation

The advent of artificial intelligence has fueled the development of algorithms with reduced and mixed precision, although their adoption in HPC applications remains limited. Most traditional scientific applications rely on FP64 computation and storage. However, recent advances in software and numerical methods demonstrate the viability of sub-FP64 precision for well-conditioned numerical methods and a range of traditional fluid flow problems [15]. Still, FP64 is the de facto standard for compressible, shock-laden flow simulation and traditional shock-capturing techniques. WENO-based reconstruction methods and approximate Riemann solves,

considered state-of-the-art and used as the baseline in this work, involve poorly conditioned operations and thus are not well suited to reduced precision [1]. We elide such issues herein by avoiding numerical shock capturing entirely via IGR.

4.4 Large CFD simulations

Previous Gordon Bell winner [23] performed a 10T grid point CFD simulation on IBM BlueGene/Q. The system's substantial CPU memory enabled this simulation. However, the wall time required to process such a large simulation makes achievable physical time scales short. More recently, [24] solved a compressible CFD problem of 10T grid points when extrapolating to 100% of Frontier.

The current largest accelerator-based flow simulations are for incompressible flows, which have fewer degrees of freedom. The largest simulation represented turbulence on a 30T point grid using OLCF Frontier [29]. Large time step costs arise from all-to-all communications, which dominate the time-to-solution.

5 Innovations Realized

5.1 The need for scale

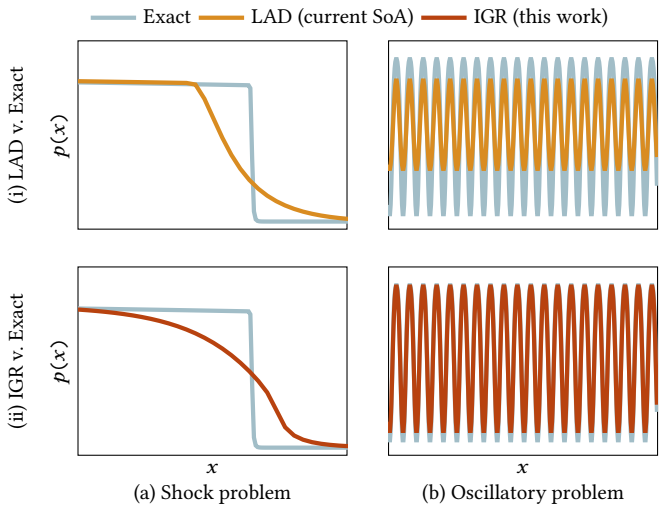
Current CFD simulations struggle to resolve phenomena across strongly separated space and time scales. The fluid dynamics of engineering interest involve these multiscale interactions; in the most elementary case, large coherent flow structures and small turbulent eddies.

For external aerodynamics, such as the jets we focus on in this study, high-fidelity predictions require faithful representation of the interaction between large-scale wake structures and small vortices. Current methods force compromises. Fine grids are required to represent shock waves, acoustic phenomena, and their turbulent interactions. Current state-of-the-art methods suffer from numerical dissipation, dissipating important flow features. By combining high resolution with low wall time cost and an *inviscid* regularization (IGR), we reduce cell sizes relative to flow feature scales, overcoming these challenges.

5.2 Shock treatment with information geometric regularization

We avoid the limitations of standard shock-capturing approaches via the first inviscid regularization of the equations, called *information geometric regularization* (IGR) as recently proposed by Cao and Schäfer [5]. IGR was first derived in the pressureless (infinite Mach number) case, where shocks amount to the loss of injectivity of the flow map $x \mapsto \phi_t(x)$ that maps gas particles from their initial position to their position at time t. IGR modifies the geometry according to which the flow map evolves, such that particle trajectories $t \mapsto \phi_t(x_0)$ do not cross and instead asymptotically approach each other (see fig. 3). This preserves the long-time post-shock behavior, prevents the formation of grid-level singularities, and recovers the nominal vanishing viscosity solutions in the limit. This result has been proven rigorously in the unidimensional, pressureless case [6].

Applying IGR to the compressible Euler equations amounts to adding a so-called *entropic pressure* Σ to p, resulting in the modified



(a) Shock problem (b) Oscillatory problem

Figure 2: Inviscid regularization: Localized artificial diffusion (LAD) spreads shocks over a user-defined width (a,i). The resulting curve is not high-order smooth. This can cause methods with a high order of accuracy to develop oscillations and, ultimately, fail. Increasing the width for coarser discretizations yields unphysical and significant dissipation of oscillatory solution profiles (b,i). Information geometric regularization replaces shocks with smooth profiles (a,ii) at the grid scale and preserves oscillatory features (b,ii).

conservation law:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0, \tag{6}$$

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u} + (p + \Sigma)\mathbf{I} - \boldsymbol{\tau}) = \mathbf{0}, \tag{7}$$

$$\frac{\partial E}{\partial t} + \nabla \cdot \left[(E + p + \Sigma)\mathbf{u} - \mathbf{u} \cdot \boldsymbol{\tau} \right] = 0, \tag{8}$$

$$\alpha \left[\mathrm{tr}\,(\nabla \mathbf{u})^2 + \mathrm{tr}^2\,(\nabla \mathbf{u}) \right] = \frac{\Sigma}{\rho} - \alpha \nabla \cdot \left(\frac{\nabla \Sigma}{\rho} \right). \tag{9}$$

Shown in fig. 2, this *inviscid* regularization yields smooth solutions without diffusion of fine-scale features. The parameter $\alpha \propto \Delta x^2$, where Δx is the nominal mesh spacing, determines the width of the smoothly expanded shocks. Computing the flux requires solving the auxiliary equation eq. (9), but $\sqrt{\alpha}$ is proportional to the mesh size. So, the resulting discrete system is a uniformly well-conditioned grid-point-local problem. Using the previous solution as a warm start, $\lesssim$ 5 Jacobi or Gauss–Seidel sweeps per flux computation suffice at negligible computational cost.

Discretization. IGR allows one to bypass shock capturing, instead using a third- or fifth-order accurate finite volume method directly. Lax–Friedrichs numerical fluxes treat the hyperbolic part of the equation. Due to the high Reynolds number of the problem under consideration, we find that a second-order accurate approximation of the derivatives of $\mathbf{u}$ suffices to compute the viscous stress tensor. We reuse these derivatives to compute the left side of eq. (9) and

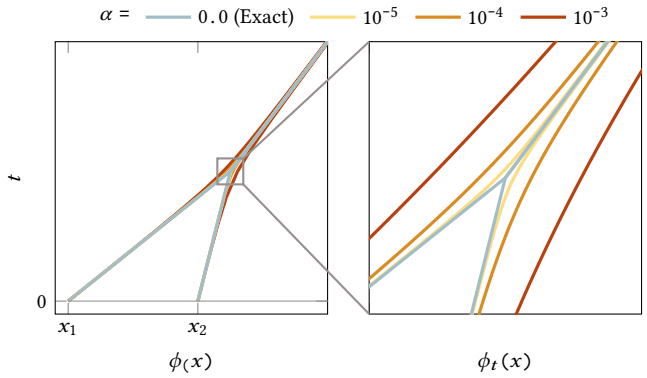


Figure 3: Information geometric regularization modifies shocks by changing the geometry according to which the flow map ϕ_t evolves in time. In the modified geometry, the trajectories of two tracer particles $t \mapsto \phi_t(x_1), \phi_t(x_2)$ converge in t rather than cross. The regularization strength α determines the rate of convergence. The vanishing viscosity solution is recovered in the $\alpha \to 0$ limit. Figure adapted from Cao and Schäfer [5] with author permission.

discretize the elliptic operator on the right using a standard 7-point stencil.

For each computation of the hyperbolic flux, we solve eq. (9) using up to 5 sweeps of Jacobi or Gauss–Siedel iteration, with the previously computed Σ as an initial guess. We use a third-order accurate Runge–Kutta time stepper [12]. For a single species (advected fluid) case, the total number of floating point numbers stored by our scheme is $17N + o(N)$, where $o(N)$ is the number of grid points. This includes 5 state variable arrays (density, energy, and three momenta), which are the *degrees of freedom* of the solution per grid cell. We also hold 5 arrays for the Runge–Kutta sub-step, 5 arrays for the right-hand side of the discretized PDE system, one array for Σ, and one for the right-hand side of eq. (9). An additional copy of Σ is required if Jacobi sweeps are used for the iterative solve.

5.3 The algorithm

The key algorithmic kernel of our method computes the right-hand side of the ordinary differential equation obtained from the spatial discretization. It is presented in algorithm 1. We advance the conservative variables line (1) at the cell centers using a 3rd-order accurate Runge–Kutta time stepper, requiring 2 copies of the state variables. The flux calculations at the cell boundaries are split dimensionally across the three coordinate directions line (12). The conservative variables are reconstructed at the cell boundaries using a 5th-order accurate polynomial interpolation scheme (lines 23 to 28).

The viscous fluxes require the calculation and reconstruction of velocity gradients (lines 16 to 18). A conversion of the reconstructed conservative variables to their primitive form is performed at the cell boundaries (lines 25 and 29). The Riemann problem at the interface is then solved using a Lax–Friedrichs approximate Riemann solver (lines 26 and 34). The net flux at the cell center is

Algorithm 1: Compute right-hand side (RHS)

```
1  (ρ, ρu, E, α) ← Conservative variables
2  (u, p) ← Primitive variables
3  Σ ← Entropic pressure
4  rhs ← Time stepper RHS
5  vflux ← Temp. array for viscous flux
6  coeff ← Reconstruction coefficients
7  igr_rhs ← RHS for elliptic solve
8  igr_func() ← Routine to compute igr_rhs
9  viscous() ← Routine for viscous flux
10 flux() ← Routine for inviscid flux
11 for t = 1 to T do                          // Loop over time steps
12     for dir ← (x, y, z) do                 // Loop over domain
13         foreach (i, j, k) in cells do
14             for q ← -2, 3 do               // Reconstruction
15                 for n ← (x, y, z) do
16                     compute d_n u
17                     vflux_L ← vflux_L + coeff_L(q) d_n u
18                     vflux_R ← vflux_R + coeff_R(q) d_n u
19                     if dir = x ∧ q = 0 then
20                         store d_n u
21                 if dir = x ∧ q = 0 then
22                     igr_rhs ← igr_func(d_x u, d_y u, d_z u)
                   // Recon. density, velocity
23                 ρ_L, ρ_R ← ρ(-2:3) along dir
24                 ρu_L, ρu_R ← ρu(-2:3) along dir
                   // Convert to primitive
25                 u_L, u_R ← ρu_L, ρu_R
                   // Viscous fluxes
26                 rhs ← viscous(vflux_L, vflux_R, u_L, u_R)
                   // Recon. remaining variables
27                 E_L, E_R ← E(-2:3) along dir
28                 α_L, α_R ← α(-2:3) along dir
                   // Convert to primitive
29                 p_L, p_R ← E_L, E_R
30                 Σ_L, Σ_R ← 0
                   // IGR contribution in y,z
31                 if dir = y ∨ dir = z then
32                     Σ_L, Σ_R ← Σ(-2:3) along dir
33                 for d ← L, R do             // Inviscid fluxes
34                     rhs ← flux(ρ_d, u_d, E_d, p_d, α_d, σ_d)
35         if dir = x then
               // IGR elliptic solve
36             Σ ← igr_rhs
37             for d ← L, R do                 // IGR x contrib.
38                 ρu_L, ρu_R ← ρu(-2:3) along dir
39                 u_L, u_R ← ρu_L, ρu_R
40                 Σ_L, Σ_R ← Σ(-2:3) along dir
41                 rhs ← flux(u_d, σ_d)
42     (ρ, ρu, E, α) ← (ρ, ρu, E, α) + dt · rhs
```

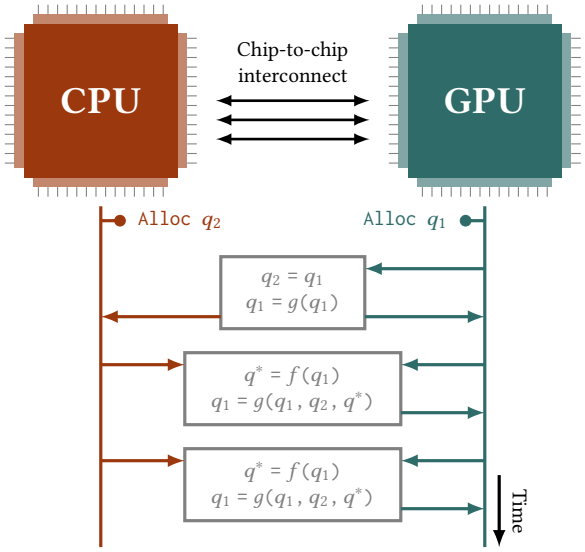


Figure 4: Schematic of the chip-to-chip (C2C) transfers of intermediate time-step variables between the on-node CPU and GPU devices. The time sub-steps are $q_{1,2}$ and the full step integration is stored in q_1.

an input to the time stepper (line 4) via the right-hand side. The entropic pressure Σ is calculated at the cell centers by solving the elliptic PDE (line 36) in eq. (9) and incorporated into the right-hand side (lines 34 and 41). The left side of eq. (9) also requires velocity gradients, which are reused from the viscous flux calculations (lines 20 to 22).

5.4 Optimizations

Our implementation eliminates the storage of the reconstructed states (lines 23 to 29), velocity gradients (lines 16 to 20), and fluxes (lines 26, 34 and 41) and keeps all operations in a single kernel (algorithm 1). The memory footprint is reduced by storing the intermediate variables as thread-local temporary arrays within this kernel. The algorithm only requires storing 2 copies of the conservative variables, the net flux at each grid point, and the solution and left side of the elliptic PDE in eq. (9).

Each thread solves an approximate Riemann problem at the grid cell–cell interface and accumulates its contribution to the right-hand side at overlapping cells via atomic operations to prevent race conditions. During the reconstruction along the 1st coordinate dimension (x, here), the contributions to the left-hand side of equation eq. (9) are computed using the velocity gradients for the viscous fluxes (line 22). The elliptic PDE in eq. (9) is solved after its left side is computed (line 36). The entropic pressure added directly to the flux in the 2nd and 3rd dimensions (lines 31 to 32). The flux contribution of the entropic pressure in the first dimension is completed separately after the elliptic solve (lines 36 to 41). The above decreases memory use 25-fold compared to an optimized 5th-order accurate WENO and Riemann solver implementation in the same codebase [28].

5.5 Unified Memory

The unified memory approach uses unified-shared-memory (USM) mode on the MI300A (El Capitan), AMD InfinityFabric for CPU–GPU connections on Frontier, and the NVLink C2C connection on Grace Hopper (Alps), Figure 4 shows this strategy, using the full capacity of the compute node while expanding beyond the GPU memory without incurring a meaningful performance penalty. This allows us to store the intermediate Runge-Kutta stage on the host, reducing GPU memory use by up to a factor of 12/17.

5.5.1 AMD approach for El Capitan. On the AMD MI300A, the CPU and GPU share a single physical HBM pool. We compile MFC with OpenMP offloading directives using AMD's next-generation Flang compiler and OpenMP's USM mode. All variables have a single copy in memory and are accessible from CPUs and accelerators with no data movement. CCE's OpenACC implementation is also used for speed comparisons.

5.5.2 AMD approach for Frontier. Our approach to Frontier involves using OpenACC or OpenMP, as well as HPE's CCE or the AMD Flang next-generation compiler, for performance comparisons between compilers and unified memory implementations.

None of the vendor compilers available on Frontier support an equivalent of USM for OpenACC, though both CCE and AMD's ROCm compiler support OpenMP's USM mode. CCE allows users to allocate device-accessible memory and request OpenACC to omit a separate device copy, effectively eliding the need for USM via UVM. The AMD heterogeneous system architecture (HSA) requires that GPU memory be accessible by the host. To reduce HBM use on Frontier, we allocate a single time-step stage as device-resident memory (via `hipMalloc`) and the second as pinned host memory (via `hipMallocManaged` and `hipMemAdvise`). With CCE and OpenACC, we set `CRAY_ACC_USE_UNIFIED_MEM=1`. Hence, the CCE OpenACC runtime detects that these arrays do not need to be mapped and instead uses zero-copy across the AMD Trento–MI250X InfinityFabric.

With OpenMP, the mapping of the Fortran pointers associated with the device-resident memory within the MFC data structures is omitted. No redundant memory copy is created due to MP's treatment of the derived type mapping with Fortran pointer components. MFC can also run on Frontier in USM mode, but was found to be less performant on Frontier than a UVM strategy discussed in section 5.5.3 below, and is not used in this work. In summary, we reduce memory usage on both the host and device without compromising performance.

With CCE, the buffers passed to MPI are mapped to device memory by the OpenACC runtime. We use GPU-aware MPI to communicate GPU buffers directly. This work also tests a development build of AMD's new Flang compiler that requires a beta version of the ROCm 7 runtime. API changes in this runtime compared to ROCm 6 require us to disable GPU-aware MPI for AMD Flang builds.

5.5.3 NVIDIA approach for Alps. The Alps nodes have a hardware-coherent system that couples a Grace CPU and a Hopper GPU using a 900 GB/s NVLink-C2C connection. This allows both processors to access all system memory at high speeds coherently and consistently. In addition to the usual CUDA allocators, GPU memory can be allocated using system allocators such as `malloc`. The unified

Table 2: Node and full system properties of the supercomputers tested in this work. TOP500 rankings from June 2025.

	Node Configuration	# Nodes	Memory [Node, System]	Peak Power	Rmax	TOP500
LLNL El Capitan	4 AMD MI300A APU	11136	[512 GB, 5.6 PB] APU	34.8 MW	1742 PFLOPs	1
OLCF Frontier	4 AMD MI250X GPU 1 AMD Trento CPU	9472	[512 GB, 4.8 PB] GPU [512 GB, 4.8 PB] CPU	24.6 MW	1353 PFLOPs	2
CSCS Alps	4 NVIDIA GH200 (4 Grace CPU, 4 Hopper GPU)	2688	[384 GB, 1.0 PB] GPU [480 GB, 1.3 PB] CPU	7.1 MW	435 PFLOPs	8

memory concept maps to the Grace Hopper superchip and is used herein.

To realize out-of-core GPU computations, we compile and link via `-gpu=mem:unified`, instructing the compiler to use CUDA Unified Memory. This provides a single unified address space for the CPU and GPU. Our optimizations leverage this infrastructure via a *zero-copy* strategy, where the most frequently accessed data are hosted in GPU memory and the least frequently accessed are in CPU memory. We avoid data movement during the simulation and only perform local or remote direct accesses. This eliminates the duplication of host and device buffers, thereby maximizing the simulation size. To fine-tune data placement, we provide memory hints to the CUDA driver via `cudaMemAdvise` and `cudaMemPrefetchAsync`. For the buffers that stay in CPU memory for the full lifetime of the simulation, we use `pinned` allocations. This results in a minimally intrusive out-of-core implementation that does not sacrifice GPU performance.

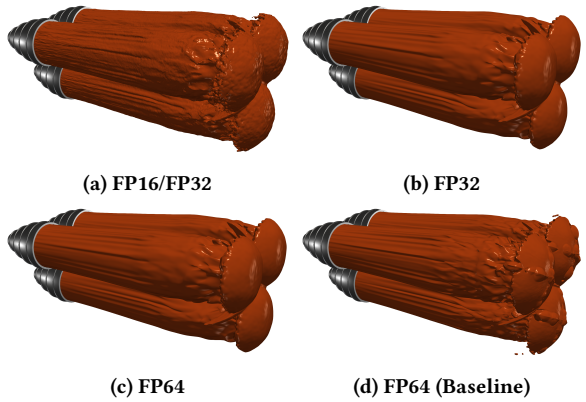
The time step updates in the Runge–Kutta scheme are rearranged so that only the current sub-step is passed to the right-hand side routine. The buffer holding the previous state is used to update the current Runge–Kutta state. With this rearrangement, we only store one sub-step and the right-hand side buffer in GPU memory. Thus, the intermediary sub-step is always in CPU memory, freeing GPU memory and increasing simulation size without sacrificing performance. The time step updates access Runge–Kutta sub-steps from their physical locations via zero-copy, allowing simultaneous access to data from both GPU and CPU memory through the C2C NVLink. We include flexibility in hosting the IGR temporary variables in either GPU or CPU memory, further reducing the memory footprint from 12/17 to 10/17 and scaling to larger problems with little performance impact.

We communicate via CUDA-aware MPI and GPUDirect, avoiding the need for staging in host memory. We allocate the send and receive buffers on the GPU using OpenACC capture to guide MPI in selecting the GPU path. This strategy enables separate memory for the send/receive halo buffers.

5.6 Mixed Precision

On all systems, we implement a FP16/32 mixed-precision strategy; computations are performed in FP32 and stored in FP16. On Alps we use NVHPC and on AMD systems the AMD Flang compiler, both of which support FP16 natively.

This strategy is numerically viable for various compressible flow simulations, as discussed in section 4.3. This strategy further doubles the maximum simulation size compared to state-of-the-art shock capturing methods we are comparing against, which are numerically unstable even at single precision.



(a) FP16/FP32 **(b) FP32**

(c) FP64 **(d) FP64 (Baseline)**

Figure 5: Visualization of a three-engine configuration and its plumes using (a) FP16, (b) FP32, (c) FP64 storage, and (d) the baseline numerics. The contours indicate where the velocity exceeds the free stream flow, and the initial state is seeded with smooth, random noise in all cases. FP32 and FP64 yield visually indistinguishable results. Visual differences in the FP16 case are solely due to the earlier onset of physical flow instabilities, yet they remain faithfully representative of the flow features. The grid-dependent nature of the baseline shock-capturing approach results in spurious grid-alignment artifacts.

While most directive constructs are readily applicable to half-precision, support for atomic updates in the NVHPC compiler was just added to its SDK and is used here. Here, atomic updates are more performant in their vectorized version, which we implement in a batched manner. Using FP16/32, we enable GH200 simulations *exceeding 100T grid points* by extrapolating our results from CSCS Alps to JSC JUPITER, which has the same primary architecture.

Figure 5 compares simulation results for engine plumes performed with FP16, FP32, and FP64 storage with the FP64 baseline numerics. Double and single precision results are visually indistinguishable. Half-precision storage yields visually different results. They arise from the more rapid onset of the hydrodynamic instabilities due to their seeding with numerical noise. At longer simulation times, we expect all precision results to be equivalent.

Grid-aligned artifacts appear in the baseline numerics result due to the grid-dependent nature of the shock-capturing approach.

6 How Performance was Measured

6.1 HPC platforms

We performed calculations on LLNL El Capitan, OLCF Frontier, and CSCS Alps. Table 2 shows the performance attributes of these systems.

6.1.1 LLNL El Capitan. The El Capitan nodes have four AMD MI300A APUs. Each APU combines 24 EPYC Zen 4 CPU cores and 228 CDNA 3 compute units with a single 128 GB layer of HBM3 memory for both processor types. Nodes are interconnected via HPE Cray Slingshot-11 Ethernet in a dragonfly topology with four 200 GB/s NICs per node. The GPUs can be programmed with unified shared memory (USM) to avoid duplicate host–device memory addresses. We use this strategy via OpenMP target offload (see section 5.5).

6.1.2 OLCF Frontier. Each Frontier node has one 64-core AMD EPYC Trento CPU and four AMD MI250X GPUs. Each MI250X GPU contains two Graphics Compute Dies (GCDs) with 64 GB of HBM2E each (or 128 GB per GPU). The total system memory is 9.6 PB, equally split between HBM2E GPU and DDR4 CPU memory. Frontier uses HPE Slingshot Ethernet for interconnects with a 3-hop Dragonfly topology and four 200 GB/s NICs per node, which are attached to the GPUs.

6.1.3 CSCS Alps. Alps nodes have 4 NVIDIA Grace Hopper GH200 superchips. Each GH200 has a Hopper GPU with 4 TB/s of memory bandwidth for its 96 GB of HBM3 memory and a Grace CPU with 72 Arm-v9 cores with 500 GB/s bandwidth for its 120 GB of LPDDR5 memory. NVLink-C2C connects the CPU and GPU, enabling efficient data movement. We use this new capability herein (see section 5.5). Alps's nodes interconnect through HPE Slingshot with 200 GB/s injection bandwidth per superchip.

6.2 Software environment and performance baseline

We base our implementation on MFC [4, 28][1], a compressible flow solver that implements state-of-the-art numerical shock capturing, as well as the IGR implementation of this work. All of MFC's schemes scale ideally to 100% of LLNL El Capitan, OLCF Frontier, and CSCS Alps, among other previous and existing flagship supercomputers [11, 20].

MFC offloads computation onto GPU and superchip/APU devices via either OpenMP or OpenACC, and uses metaprogramming to abstract away and automate vendor-specific or otherwise burdensome optimizations. MFC has a history of being used to simulate compressible multi-species, phase, and chemically reacting fluid flows [2, 3, 7, 8]. Performance results are measured using a representative three-dimensional simulation of the exhaust plume of a single Mach 10 jet.

We use MFC's optimized implementation of WENO nonlinear reconstructions and HLLC approximate Riemann solves as a baseline for performance comparisons [28]. This implementation matches or outperforms other codebases with the same and similar reconstruction schemes. On Frontier and El Capitan we test HPE's CCE 19.0.0 compiler and a pre-release of AMD's Next Generation Flang compiler with HPE Cray MPICH-8.1.31 for messaging. For both compilers, we specify `-O3` for optimization. With CCE, we also apply the `-haggress` flag to improve performance in large kernels. On Alps, we use a pre-release of the NVIDIA HPC SDK 25.9 and HPE Cray MPICH 8.1.30 for messaging. We also use the compile-time flag `-gpu=fastmath`, which yields a performance improvement for single-precision computation on NVIDIA devices. The latest NVIDIA HPC SDK exhibits an FP64 performance regression; we use NVHPC SDK 24.3 instead.

[1] Openly available at https://github.com/MFlowCode/MFC

Table 3: Wall time for representative simulations, quantified via nanoseconds per grid cell per time step. The baseline method is compared to the current work. For the AMD MI300A, USM mode is used, so no in-core quantities are presented; other unified cases use UVM. AMD device cases were run using Cray CCE 19.0.0 and AMD Flang Preview 7.0.5 compilers, using OpenACC and OpenMP offloading, with the best results presented. AMD Flang is the only current compiler option for native FP16/32 on all AMD GPU/APU devices. NVHPC is used for the GH200 cases.

Device	Baseline (in-core)	IGR (in-core)	IGR (unified)	
GH200	16.89	3.83	4.18	**FP64**
MI250X GCD	69.72	13.01	19.81	
MI300A	29.50	†—	7.21	
GH200	*N/A	2.70	2.81	**FP32**
MI250X GCD	*N/A	9.12	13.03	
MI300A	*N/A	†—	4.19	
GH200	*N/A	3.06	3.07	**FP16/32**
MI250X GCD	*N/A	22.63	24.71	
MI300A	*N/A	†—	17.39	

*Numerically unstable; †MI300A is always unified

6.3 Measurement tools

We measure execution time using application internal timers, including the standard `cpu_time` and `system_clock` procedures. On Frontier and El Capitan, we periodically sample the instantaneous GPU or APU power draw by reading the virtual file system for the AMD driver. The power reading is the same as reported by `rocm-smi`, but reading from the virtual filesystem has lower overhead. On Frontier, `rocm-smi` collects GPU and HBM power only. On El Capitan `rocm-smi` includes CPU, GPU, and memory power. On Alps, `nvidia-smi` records the module (CPU, GPU, and memory) power draw.

On all platforms, we post-process the results to account only for power draw during time-stepping, which is then averaged and multiplied by the average time per time step. The number of grid points further normalizes energy use.

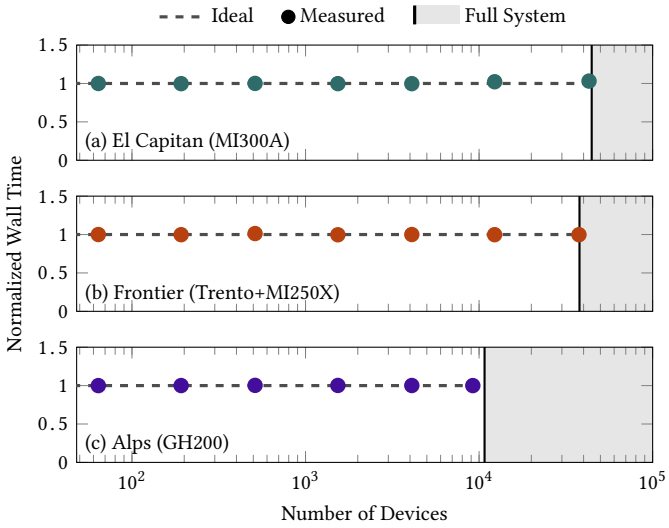


Figure 6: Weak scaling performance for a representative thruster problem on LLNL El Capitan (number of MI300As), OLCF Frontier (number of MI250Xs), and CSCS Alps (number of GH200s). The full system is indicated for each case. All runs use unified memory, FP16/32 mixed precision, and a 16 node configuration for scaling comparison, which ensures that all MPI communication directions are touched. On El Capitan, we see 97% efficiency out to 10750 MI300As. On Alps, ideal (≈ 100%) scaling is observed at 9.2K GH200s. On Frontier, we also achieve ideal (≈ 100%) efficiency at 37.6K MI250X GPUs, with a maximum problem size of 200T grid points.

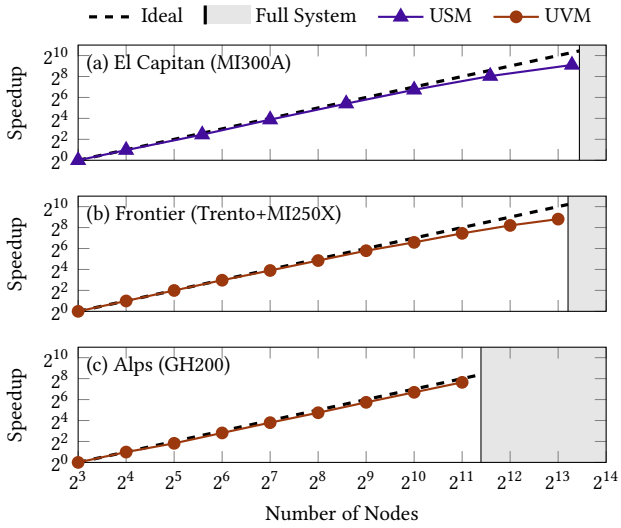


Figure 7: Strong scaling on all systems for the same problem configuration as fig. 6. All results are for FP16/32 mixed precision, though FP32 and FP64 show similar results (FP32 shown in fig. 8). The full system is shown for each case. Following fig. 6, we base all speedups on an 8 node configuration, such that all communication directions are touched.

7 Performance Results

7.1 Time step cost

Table 3 shows the normalized grind times for a problem solved using WENO reconstructions and HLLC approximate Riemann solves with in-core computation (current state of the art, optimized implementation in MFC [28]), IGR with in-core computation, and IGR with unified memory on one GH200 on Alps, one MI250X GCD on Frontier, and one MI300A APU on El Capitan. The grind time is defined as nanoseconds per grid cell per time step, used to normalize against the different problem sizes that fit within device memory. Smaller grind times indicate shorter time to solution.

GH200 metrics are collected using separate memory mode for in-core cases and unified memory mode for out-of-core/unified cases. Performance impacts of less than 5% are observed when moving from in-core to unified computation on the GH200 architecture. We used the NVHPC 24.3 compiler for the GH200 FP64 cases due to performance regressions in FP64 performance in newer versions. However, an unrelated regression in NVHPC 24.3 was observed when switching from separate memory to unified memory in-core computation, resulting in a 9% performance hit in that case (see table 3). The performance difference between in- and out-of-core unified memory builds is less than 5%.

When transitioning from in-core computation to unified memory computation on Frontier, performance degradation of 42% and 51% is observed for FP32 and FP64, owing to the slower xGMI links between the Trento CPU and MI250X GCDs (72 GB/s each) compared to the C2C bandwidth between the Grace CPU and Hopper GPU (900 GB/s). On El Capitan, there is no difference between unified/in-core as the device shares a unified HBM pool.

The increased grind time in calculations that use unified memory to increase problem size per device on Frontier and Alps results from the exchange of conservative variable buffers between the CPU and GPU at each Runge–Kutta update. On all devices, the time to solution is reduced by a factor of approximately 4 when comparing WENO to IGR in FP64. FP64 is broadly recommended for ENO-type shock-capturing due to catastrophic cancellation [1]. Our approach can even handle even mixed FP16/FP32 precision. This reduces the time to solution by a factor of at least 6 compared to the baseline.

For FP16/32, we observe a performance regression on all devices compared to FP32, but we expect to exceed or match FP32 performance upon further code and compiler releases and optimizations. We use the AMD Flang beta compiler and a pre-release version of the NVHPC SDK 25.9. Further compiler improvements are staged for non-beta AMD Flang releases, and the performance hit can be expected to be similar to the NVHPC strategy.

7.2 Scaling

Figure 6 shows the weak scaling performance on LLNL El Capitan, OLCF Frontier, and CSCS Alps. Unified memory and FP16/32 mixed precision are used on all systems. A weak scaling efficiency of 97%

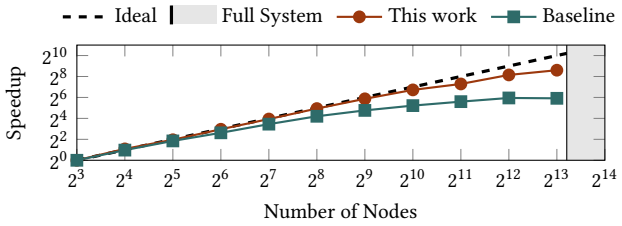


Figure 8: Strong scaling following fig. 7, here for FP32 on Frontier. We compare the performance of the current work to the optimized baseline method as labeled. The current work uses and can accommodate 10.5B grid points per node, while the baseline accommodates 421M grid points, both of which are used for the 8-node speedup reference.

is observed when scaling from 64 to 43K MI300As on El Capitan. On Frontier, we achieve perfect (100%, up to measurement variance) weak scaling when scaling from 64 to 37.6K MI250X GPUs (128 to 75.2K GCDs). Likewise, on CSCS Alps, we achieve perfect weak scaling when scaling from 64 to 9.2K GH200s. Better than 95% efficiencies are also seen for FP32 and FP64 precision (not shown).

We exceed *200T grid cells*, or 1 quadrillion degrees of freedom (5 state variables) on Frontier using 37.6K MI250X GPUs (9408 nodes), accommodating 1386^3 grid points per GCD with UVM, and FP16/32 mixed precision. On Alps, a problem size of 1611^3 is used per GH200 and UVM with FP16/32 mixed precision, amounting to 45T grid point simulation on the full system (2688 nodes). On the recently deployed JSC JUPITER, this amounts to 100.3T grid points or 501T degrees of freedom, given its size and matching architecture to Alps. On El Capitan, we use 1380^3 grid points per MI300A GPU and reach 113T grid points using 10750 nodes (11.1K nodes full system). The lower grid count on Alps and El Capitan is due purely to system size.

Strong scaling results on Alps, Frontier, and El Capitan are shown in fig. 7. The base case uses 8 nodes (32 GPUs), resulting in 32 ranks on El Capitan and Alps and 64 ranks on Frontier, arranged in a rectilinear configuration. We use FP16/32 mixed precision for storage/computation; unified memory is enabled via UVM on Frontier and Alps and USM on El Capitan; GPU-aware MPI is used on Alps. For a 32-fold increase in device count, we achieve strong scaling efficiencies of 90%, 90%, and 86% on El Capitan, Frontier, and Alps. When scaling to the *full systems*, we achieve 44% (El Capitan), 44% (Frontier), and 80% (Alps) strong scaling efficiencies. Thus, one can execute an 8 node computation on the full system, decreasing time to solution by a factor of about *500*.

Figure 8 shows optimized baseline numerics in FP32 for the same problem, which do not strong-scale well compared to the current work. We observe 6% strong scaling efficiency for the baseline and 38% for the current work when scaling from 8 nodes to the full Frontier system.

7.3 Energy efficiency

Table 4 shows the measured energy consumption per grid cell per time step, comparing our IGR implementation with a prior state-of-the-art WENO implementation. The problem size is adjusted to exhaust the total GPU/APU memory on a single node in double precision. On the NIVIDA GH200 we measured the in-core implementation. On the AMD MI250X and MI300A, we use the CCE 19.0.0 compiler with GPU-only (in-core) and unified memory builds.

Table 4: Energy in μJ per grid cell per time step for the baseline implementation compared to the current work.

Energy (μJ)	El Capitan (MI300A)	Frontier (MI250X)	Alps (GH200)
Baseline	15.24	10.67	9.349
IGR	3.493	1.982	2.466

The IGR method significantly reduces energy consumed compared to the current state of the art on all platforms. The most significant reduction comes from the improved time to solution, and to lowest order, the Frontier and El Capitan energy improvements match the grind time speedups in table 3. Due to the higher power draw of the WENO scheme on Alps, we observe energy savings beyond those resulting from grind time speedup.

The largest improvement is realized on Frontier with a 5.38-times improvement in energy consumed by the GPU. As described in section 6.3, the Frontier measurements include only GPU and GPU HBM energy and do not represent changes that may have impacted energy consumption of the CPU or CPU memory.

8 Implications

This work presents a highly scalable technique for predictive simulation of compressible fluid flows. We demonstrated the method's capability by simulating high-Mach many-rocket engine thrusters and their plume–plume interaction. This demonstrates that fully resolved simulations of these systems are within reach of current supercomputers, exceeding the scales demanded of state-of-the-art current and prototype spacecraft. This advance paves the way for the computation-driven design of critical components for space exploration.

The combination of IGR, the resulting simplified algorithm, and its optimized implementation improves the efficiency of simulating compressible flow, as measured by key metrics: time to solution, memory footprint, and energy to solution. We leverage the closely coupled architectures of modern supercomputing platforms, including the MI250X, MI300A, and GH200, which serve as proof of concept for the efficient use of unified memory addressing.

Furthermore, our simplified algorithm is amenable to mixed-precision computation. The resulting computational savings enable the first CFD simulations with more than *200 trillion* grid points and *1 quadrillion degrees* of freedom, exceeding the previous largest such simulations by a factor of 20. At the same time, compared to the baseline, the improved grind times and near-ideal strong scaling of our approach enable flagship supercomputers to achieve orders of magnitude reductions of time to solution on smaller problems. This unlocks new opportunities for integrating simulation into design optimization. Beyond flagship supercomputing, the drastic increase in grid points per device extends the capabilities of resource-constrained users.

This work focuses on a set of thruster plumes. However, the methodology shown is suited to general multiphase and multicomponent flows, spanning fields ranging from biomedical treatments to marine and aviation applications. Our work thus has the potential to aid the understanding of a broad range of physical phenomena. IGR is *agnostic* to the numerical discretization, as it regularizes the momentum balance equation of the associated PDE. Thus, one could apply the same strategy to other techniques, such as discontinuous Galerkin or finite difference methods.

This work demonstrates how users can avoid expensive and complex viscous, nonlinear numerics via IGR. This achieves markedly better scalability, speed, problem sizes, and time- and energy-to-solution than current state-of-the-art methods. The spatial discretizations used in this work adhere to traditional algorithmic design patterns commonly used in computational fluid dynamics. Removing the need for numerical shock capturing enables a vast and largely unexplored design space for numerical methods for fluid dynamics problems.

Acknowledgments

SHB acknowledges the use of 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 and allocation CFD154 (PI Bryngelson). This work was also supported by a grant from the Swiss National Supercomputing Centre (CSCS) for Alps.

FS acknowledges support from the Air Force Office of Scientific Research under award number FA9550-23-1-0668 (Information Geometric Regularization for Simulation and Optimization of Supersonic Flow).

The authors gratefully acknowledge contributions from Scott Futral (LLNL), Rob Noska (HPE), Michael Sandoval (OLCF), and Mat Colgrove (NVIDIA).

References

[1] Federico Brogi, Stefano Bnà, Giuseppe Boga, Giacomo Amati, Tomaso Esposti Ongaro, and Matteo Cerminara. 2024. On floating point precision in computational fluid dynamics using OpenFOAM. *Future Generation Computer Systems* 152 (2024), 1–16.

[2] S. H. Bryngelson and T. Colonius. 2020. Simulation of humpback whale bubble-net feeding models. *Journal of the Acoustical Society of America* 147, 2 (2020), 1126–1135.

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

[4] S. H. Bryngelson, K. Schmidmayer, V. Coralic, J. C. Meng, K. Maeda, and T. Colonius. 2021. MFC: An open-source high-order multi-component, multi-phase, and multi-scale compressible flow solver. *Computer Physics Communications* 266 (2021), 107396.

[5] Ruijia Cao and Florian Schäfer. 2023. Information geometric regularization of the barotropic Euler equation. *arXiv preprint arXiv:2308.14127* (2023).

[6] Ruijia Cao and Florian Schäfer. 2024. Information geometric regularization of unidimensional pressureless Euler equations yields global strong solutions. *arXiv preprint arXiv:2411.15121* (2024).

[7] A. Charalampopoulos, S. H. Bryngelson, T. Colonius, and T. P. Sapsis. 2022. Hybrid quadrature moment method for accurate and stable representation of non-Gaussian processes and their dynamics. *Philosophical Transactions of the Royal Society A* 380, 2229 (2022).

[8] Esteban Cisneros-Garibay, H. Le Berre, Spencer H. Bryngelson, and Jonathan B. Freund. 2025. Pyrometheus: Symbolic abstractions for XPU and automatically differentiated computation of combustion kinetics and thermodynamics. *arXiv preprint arXiv:2503.24286* (2025).

[9] Andrew W Cook and William H Cabot. 2004. A high-wavenumber viscosity for high-resolution numerical methods. *J. Comput. Phys.* 195, 2 (2004), 594–601.

[10] Gil Denis, Didier Alary, Xavier Pasco, Nathalie Pisot, Delphine Texier, and Sandrine Toulza. 2020. From new space to big space: How commercial space dream is becoming a reality. *Acta Astronautica* 166 (2020), 431–443.

[11] 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. Melesse 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, and J. Young. 2023. 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, 35–49.

[12] Sigal Gottlieb and Chi-Wang Shu. 1998. Total variation diminishing Runge–Kutta schemes. *Math. Comp.* 67, 221 (1998), 73–85.

[13] Jean-Luc Guermond, Richard Pasquetti, and Bojan Popov. 2011. Entropy viscosity method for nonlinear conservation laws. *J. Comput. Phys.* 230, 11 (2011), 4248–4267.

[14] Amiram Harten, Peter D Lax, and Bram van Leer. 1983. On upstream differencing and Godunov-type schemes for hyperbolic conservation laws. *SIAM review* 25, 1 (1983), 35–61.

[15] M. Karp, R. Stanly, H. Song, T. Mukha, L. Galimberti, S. Toosi, L. Dalcin, S. Rezaeiravesh, M. Münsch, N. Jansson, S. Markidis, M. Parsani, S. T. Bose, S. K. Lele, and P. Schlatter. 2024. Sensitivity of numerical simulations of turbulence to lower floating-point precision. In *Proceedings of the Summer Program, Center for Turbulence Research*. 405–414.

[16] Xu-Dong Liu, Stanley Osher, and Tony Chan. 1994. Weighted essentially non-oscillatory schemes. *J. Comput. Phys.* 115, 1 (1994), 200–212.

[17] Ali Mani, Johan Larsson, and Parviz Moin. 2009. Suitability of artificial bulk viscosity for large-eddy simulation of turbulent flows with shocks. *J. Comput. Phys.* 228, 19 (2009), 7368–7374.

[18] Manish Mehta, Francisco Canabal, Scott B Tashakkor, and Sheldon D Smith. 2013. Numerical base heating sensitivity study for a four-rocket engine core configuration. *Journal of Spacecraft and Rockets* 50, 3 (2013), 509–526.

[19] Saadia M. Pekkanen. 2019. Governing the New Space Race. *American Journal of International Law Unbound* 113 (2019), 92–97.

[20] A. Radhakrishnan, H. Le Berre, B. Wilfong, J.-S. Spratt, M. Rodriguez Jr., T. Colonius, and S. H. Bryngelson. 2024. Method for portable, scalable, and performant GPU-accelerated simulation of multiphase compressible flow. *Computer Physics Communications* 302 (2024), 109238.

[21] Fantao Ren. 2024. Numerical-Study of Large-Liquid Rocket Plume-Flow-Field. In *Journal of Physics: Conference Series*, Vol. 2755. IOP Publishing, 012038.

[22] Philip L Roe. 1981. Approximate Riemann solvers, parameter vectors, and difference schemes. *J. Comput. Phys.* 43, 2 (1981), 357–372.

[23] Diego Rossinelli, Babak Hejazialhosseini, Panagiotis Hadjidoukas, Costas Bekas, Alessandro Curioni, Adam Bertsch, Scott Futral, Steffen J Schmidt, Nikolaus A Adams, and Petros Koumoutsakos. 2013. 11 PFLOP/s simulations of cloud cavitation collapse. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 1–13.

[24] Srikanth Sathyanarayana, Matteo Bernardini, Davide Modesti, Sergio Pirozzoli, and Francesco Salvadore. 2025. High-speed turbulent flows towards the exascale: STREAmS-2 porting and performance. *J. Parallel and Distrib. Comput.* 196 (2025), 104993.

[25] Eleuterio F Toro. 2019. The HLLC Riemann solver. *Shock Waves* 29, 8 (2019), 1065–1082.

[26] Bram Van Leer. 1979. Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method. *J. Comput. Phys.* 32, 1 (1979), 101–136.

[27] Xu Wang, Xu Xu, Jiaqi Yu, and Qingchun Yang. 2024. Effect of Free-Stream Mach Number on the Base Thermal Environment of Launch Vehicle. *Journal of Thermal Science* 33, 6 (2024), 2426–2436.

[28] Benjamin Wilfong, Henry Le Berre, Anand Radhakrishnan, Ansh Gupta, Diego Vaca-Revelo, Dimitrios Adam, Haocheng Yu, Hyeoksu Lee, Jose Rodolfo Chreim, Mirelys Carcana Barbosa, Yanjun Zhang, Esteban Cisneros-Garibay, Aswin Gnanaskandan, Mauro Rodriguez Jr., Reuben D. Budiardja, Stephen Abbott, Tim Colonius, and Spencer H. Bryngelson. 2025. MFC 5.0: An exascale many-physics flow solver. *arXiv preprint arXiv:2503.07953* (2025).

[29] PK Yeung, Kiran Ravikumar, Stephen Nichols, and Rohini Uma-Vaideswaran. 2025. GPU-enabled extreme-scale turbulence simulations: Fourier pseudo-spectral algorithms at the exascale using OpenMP offloading. *Computer Physics Communications* 306 (2025), 109364.