Cache-Driven Super-Linear Scaling in a Python Fluid Solver

Harshit Tiwari
NYU Tandon School of Engineering, USA
IIT Kanpur, India
harshit.tiwari@nyu.edu

Samar Aseeri
KAUST, Saudi Arabia
samar.aseeri@kaust.edu.sa

Katepalli R. Sreenivasan
NYU Tandon School of Engineering, USA
katepalli.sreenivasan@nyu.edu

David E. Keyes
KAUST, Saudi Arabia
david.keyes@kaust.edu.sa

Mahendra K. Verma
IIT Kanpur, India
mkv@iitk.ac.in

***Abstract*—Turbulent convection transports heat in stars, planets, and the earth's atmosphere, and simulating it demands extreme-scale computing. We present DHARA, a modular fluid solver written in Python (NumPy, CuPy, mpi4py) with compiled stencil kernels for CPUs and GPUs. On the CPU-based Shaheen III (KAUST), DHARA scales to 786,432 cores for compressible convection, and its strong scaling is *super-linear*: adding cores speeds up each step by more than the ratio of the core counts. Profiling explains why. As the domain is split across more cores, each subdomain's stencil working set drops into the fast L1/L2 caches, so cache misses collapse, leading to a better than doubling of the per-core throughput. The same code carries over to the Frontier (OLCF) and Polaris (ALCF) GPU systems with near-ideal strong scaling.**

***Index Terms*—Python HPC, super-linear scaling, cache-fit effect, GPU acceleration, thermal convection.**

I. Introduction

Turbulent thermal convection drives planetary cores, stellar interiors, and the earth's atmosphere; resolving its thin boundary layers and bulk turbulent cascade demands massive grids and extreme-scale computing. Python is prized for productivity but rarely trusted at this scale; DHARA shows that Python can compete. DHARA is a general-purpose Python solver (NumPy, CuPy, mpi4py); here we study one representative problem, compressible convection, on which it achieves super-linear strong scaling on CPUs and near-ideal scaling on GPUs.

II. Numerical Method and Parallelization

DHARA solves the fully compressible Navier-Stokes equations in a Cartesian box:

$$\partial_t \rho + \partial_i(\rho u_i) = 0 \tag{1}$$

$$\partial_t(\rho u_i) + \partial_j(\rho u_i u_j + p\delta_{ij} - \tau_{ij}) = -\rho g\, \delta_{i3} \tag{2}$$

$$\partial_t E + \partial_i\left(u_i(E+p) - u_j\tau_{ji} - \Phi_i\right) = 0, \tag{3}$$

with density ρ, velocity u_i, pressure p, energy E, and gravity g; the non-dimensional stress τ_{ij} and heat flux Φ_i carry the Rayleigh, Prandtl, dissipation, and compressibility parameters.

We recast these as conservation laws for $\mathbf{q} = [\rho,\, \rho u_x,\, \rho u_y,\, \rho u_z,\, E]^T$ and integrate with a second-order MacCormack-TVD scheme on a collocated grid. Strang splitting factors each step into a symmetric sequence of 1D sweeps,

$$\mathbf{q}^{n+1} = \mathbf{L}_x\mathbf{L}_y\mathbf{L}_z\mathbf{L}_z\mathbf{L}_y\mathbf{L}_x\,\mathbf{q}^n, \tag{4}$$

where $\mathbf{L}_\alpha(\delta t/2)$, $\alpha \in \{x, y, z\}$, advances $\mathbf{q}$ along one axis; the palindromic order gives second-order accuracy and the milder 1D CFL limit. Each $\mathbf{L}_\alpha$ combines a backward predictor and forward corrector, with flux $\mathbf{F}_\alpha$ and source $\mathbf{S}_\alpha$:

$$\mathbf{q}^p = \mathbf{q}^n - \tfrac{\delta t}{2}\,\Delta_-\mathbf{F}^n_\alpha + \tfrac{\delta t}{2}\mathbf{S}^n_\alpha \tag{5}$$

$$\mathbf{q}^c = \mathbf{q}^n - \tfrac{\delta t}{2}\,\Delta_+\mathbf{F}^p_\alpha + \tfrac{\delta t}{2}\mathbf{S}^p_\alpha \tag{6}$$

$$\mathbf{L}_\alpha\mathbf{q}^n = \tfrac{1}{2}(\mathbf{q}^p + \mathbf{q}^c). \tag{7}$$

A separate TVD-correction step adds a limited dissipative term $\mathbf{\Psi}^n$ near steep gradients. These compact, single-axis $\mathbf{L}$ operators are the solver's computational kernel, letting the working set fit in cache (Section III).

The decomposition adapts to the hardware: a 2D *pencil* on CPUs, keeping subdomains from growing too thin across hundreds of thousands of ranks, and a 1D *slab* on GPUs, where far fewer ranks keep communication simple. Non-blocking MPI hides communication: each step posts `Isend`/`Irecv` for the halos and applies the $\mathbf{L}$ operators over the interior while the halos are in transit. On GPUs, `cupy.ElementwiseKernel` fuses each operator's flux and update into one pass.

III. Results and Discussion

On Shaheen III (KAUST), an HPE Cray EX system of 4,608 dual-socket AMD EPYC 9654 nodes (192 cores, 384 GB each), DHARA scaled compressible convection to 4,096 nodes — 786,432 of the machine's 884,736 cores (≈89%) — with *super-linear* strong scaling at every resolution (Fig. 1): each core doubling cut the step time by more than half.

We profiled the 1152^3 case with the Tuning and Analysis Utilities (TAU) from 9 to 288 nodes: computation time fell from 14.21 to 0.08 s per step, outpacing the $32\times$ core growth (Fig. 2), while per-core memory held near 100 MB, far below the 384 GB per node, ruling out DRAM relief.

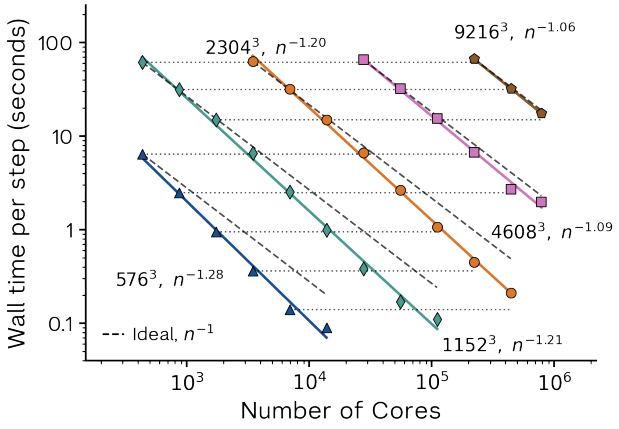


Fig. 1: Strong/weak scaling on Shaheen III; exponents ($n^{-1.06}$–$n^{-1.28}$) beat the ideal n^{-1} (dashed).

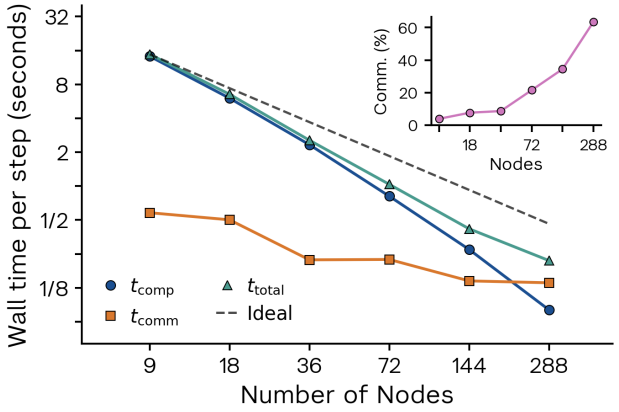


Fig. 2: Step time for the 1152^3 case, split into computation and communication; the latter grows to 64% by 288 nodes (inset).

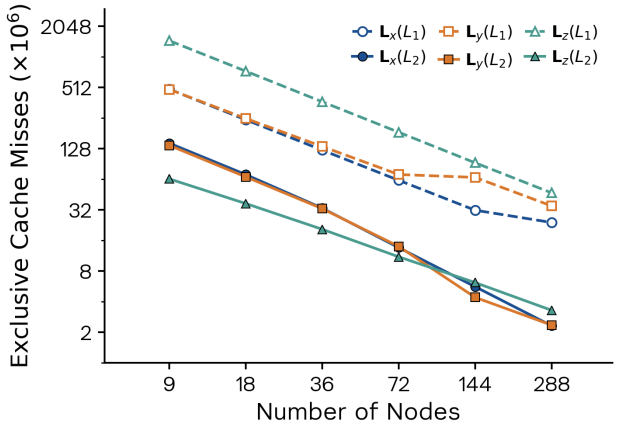


Fig. 3: Exclusive L1/L2 misses for the three $\mathbf{L}$ operators; misses drop up to two orders of magnitude as the subdomain shrinks.

The reason is geometric: each EPYC 9654 core has 64 KB of L1 and 1 MB of L2. As the pencil decomposition spreads the domain over more cores, the local subdomain shrinks

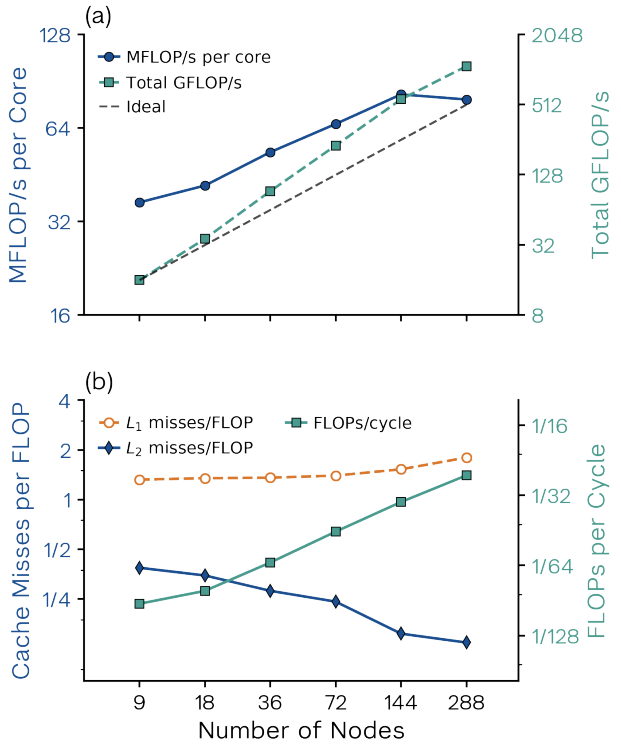


Fig. 4: Arithmetic efficiency, 1152^3 case ($\log_2$ axes, so ticks read as powers of two, e.g. 1/2, 1/4): (a) throughput, peaking at 144 nodes; (b) L2 misses/FLOP fall as FLOPs/cycle rise.

from 337.5 to 10.5 MB — still too large for L2, yet each $\mathbf{L}_\alpha$ sweep touches only a few neighboring planes, small enough to sit in cache. Cache-resident, the $\mathbf{L}$ operators shed exclusive L2 misses by up to two orders of magnitude (Fig. 3), and the DRAM-latency stalls that throttle a memory-bound solver vanish. Throughput follows (Fig. 4): 36.9 to 82.0 MFLOP/s at 144 nodes (2.2×), cache locality — not parallelism — driving the speedup. Beyond 144 nodes, communication overtakes computation.

The same solver runs on GPUs with minimal change; a single A100 delivers about 200× one AMD EPYC 7543 core. Multi-GPU strong scaling stays near-ideal ($t \propto n^{-0.9}$) on both vendors: 8–512 A100s on Polaris (ALCF, up to 4096^3) and 64–8,192 MI250X GCDs on Frontier (OLCF, up to 8192^3). GPUs rely on high-bandwidth memory rather than deep per-core caches, so they never see the CPU's cache-fit transition. Table I summarizes DHARA's applications to date. The optimized DHARA backends also compare favorably with established solvers on both the H100 GPU and Polaris CPU benchmarks (Table II).

TABLE I: Active Research Applications of DHARA

Problem Type	**Method**	**Key Parameters**	**Max Resolution**
Compressible Convection (2D)	Finite Difference	$Ra = 10^{16}$	32768^2
Compressible Convection (3D)	Finite Difference	$Ra = 10^{14}$	$1024^2 \times 8192$
4D Rayleigh-Bénard Convection	Finite Difference	$Ra = 2 \times 10^8$	256^4
Moist Convection	Finite Difference	$Ra = 2 \times 10^6$	$4096^2 \times 64$
Supersonic Turbulence	Finite Volume	$M_t = 3.0$	1024^3
Quantum Turbulence	Pseudo-Spectral	Gross-Pitaevskii	1024^3

TABLE II: Compressible Euler Taylor–Green vortex benchmark using WENO5-Z, HLLC, and SSPRK3.

Platform/grid	**Solver (backend)**	**Time/step (ms)**
H100, 256^3	DHARA (RawKernel)	29.1
	AthenaK (Kokkos)	33.9
	JAX-Fluids (JAX compiler)	99.3
	DHARA (ElementwiseKernel)	200.0
	DHARA (CuPy)	786.3
EPYC 7543P, 128^3	DHARA (Numba)	140.3
	AthenaK (Kokkos)	175.8

IV. Conclusion

A Python-based solver thus reaches extreme-scale, superlinear performance on CPUs while porting to exascale GPUs with minimal change. On many-core nodes, domain decomposition is as much a cache-hierarchy decision as a load-balancing one. We are now applying DHARA at higher Rayleigh numbers and to moist convection in the earth's troposphere.

Acknowledgment

We thank Dr. Lekha Sharma, Dhananjay Singh, Prof. Shashwat Bhattacharya, Shashwat Nirgudkar, and Prof. Rajesh Ranjan for their valuable discussions. We acknowledge computational resources from King Abdullah University of Science and Technology (KAUST), Oak Ridge Leadership Computing Facility (OLCF), Argonne Leadership Computing Facility (ALCF), and the Kotak School of Sustainability (KSS) at IIT Kanpur.