POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Boosting performance of compressible CFD on Exascale platforms

Laurea Magistrale in High-Performance Computing Engineering - Ingegneria del Calcolo ad Alte Prestazioni

Author: Tommaso Trabacchin

Advisor: Prof. Fabrizio Ferrandi

Co-advisors: Prof. Diego Rossinelli, Dr. Patrick Zulian

Academic year: 2025-2026

1. Introduction

Computational fluid dynamics (CFD) has grown enormously in impact and applicability and today encompasses engineering domains including chemically reacting flows , nuclear engineering, rarefied gas dynamic, combustion, jet design, gas turbine analysis, detonation modeling, high-speed train design , and supersonic land-based vehicle design.

Its wider adoption likely reflects how it complements experimental fluid mechanics by providing access to spatiotemporal scales beyond what can be measured in the laboratory, albeit within a limited range. The drive to expand that range has kept CFD at the core of supercomputing since the earliest mainframes. With exascale alleviating the limitation, we expect an even more transformative role of CFD in engineering design to emerge, especially where ground testing is impractical for reasons of safety or facility availability.

The range of scales accessible via CFD are bounded by how effectively the software leverages the computing infrastructure. We measure achieved performance as the attained fraction of peak:

$$\% \text{ of Peak} = \frac{\text{FLOP count}}{\text{TTS}} \cdot \frac{100}{\text{Peak}} \qquad (1)$$

where 'FLOP count' is the overall FP64 arithmetic operations exhibited by the algorithm [1], TTS is the time-to-solution, and 'Peak' is the nominal FP64 peak performance of the underlying computing system. is measured as the total relevant algorithmic operations divided by the *time-to-solution* (TTS) and becomes especially meaningful when compared to the theoretical nominal peak performance. Its reciprocal sets a hard upper bound on the spatiotemporal scales that a given algorithm implementation can resolve within a given TTS and compute allocation.

Exascale Challenges.

The computational power of exascale systems promises to transform scientific simulation, yet poses major challenges. Software must simultaneously achieve 1)high fractions of peak performance, 2) performance portability, and 3) software productivity.

[1] even though the implementation FLOP count could be substantially higher.

Reaching high peak efficiency requires deep microarchitectural awareness and domain expertise. Reported fractions of peak for compressible CFD are scarce, but state-of-the-art solvers typically achieve around 5%, with values near 13% considered exceptional. Moreover, the exascale ecosystem is fragile and heterogeneous: small instruction changes can trigger severe penalties, and architectural diversity often demands source-level specialization.

No established *technical path* addresses all three goals at once. Current approaches fall into three categories. Hand-written kernels maximize performance but sacrifice portability and productivity. Compiler directives improve productivity but rarely match hand-tuned performance. Domain-specific languages (DSLs) offer a compromise, enabling algorithmic and performance tuning at a higher abstraction level.

Challenging conventional practice, Rossinelli et al. [1] propose high-throughput–oriented HPC guidelines, including microbenchmark-based system characterization, a symbolic layer to enhance productivity and portability, and techniques to hide network communication latency.

Contributions

We propose a *technical path* for addressing the exascale challenges related to compressible flow solvers and beyond. Such path includes a performance modelling framework for bottom-up supercomputing software development. We validate this methodology by realizing a new compressible flow solver *LVR* which stands for *Lean Vertical Rewrite*. LVR

- achieves 35% of the nominal peak performance of the underlying platforms,
- measures at 35% of peak across diverse computing systems, including the Aurora exascale supercomputer,

and shows that the challenges discussed above are addressable in practice. The baseline implementation of our flow solver has been developed in three months by four researchers by strictly following the proposed modelling framework.

2. Numerical Schemes and Solver Structure

LVR solves the compressible flow problem, modelled by the following hyperbolic system of equations:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0 \tag{2a}$$

$$\frac{\partial (\rho \boldsymbol{u})}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}\boldsymbol{u}^T + p\mathbb{I}) = \boldsymbol{0} \tag{2b}$$

$$\frac{\partial (\rho E)}{\partial t} + \nabla \cdot ((\rho E + p)\boldsymbol{u}) = 0 \tag{2c}$$

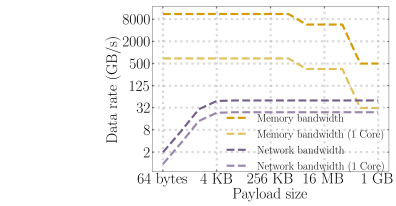
where $\rho, \boldsymbol{u}, p, E, T$ are the density, speed vector, pressure, energy and absolute temperature of the fluid. In the present work, the above system of equations is closed by the ideal gases equation of state.

The above model is spatially discretized with the Finite Volume method. Values at inter-cell boundaries are first projected to the characteristic space, reconstructed using fifth-order WENO, projected back to the original space, and then fed to a HLLC approximate Reimann solver, which computes numerical fluxes. The latter are provided as input to a third-order low-storage Runge-Kutta scheme, which advances the solution in time. At each time-step, its duration is adaptively computed, so that the CFL constraint is respected. More details can be found in [2]

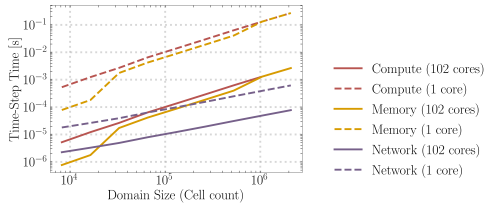
Following these numerical schemes, LVR is composed of three main kernels: `DT`, which computes the adaptive time-step length; `FLUX`, which computes the fluxes; `DUC`, which computes fluxes divergence, applies the Runge-Kutta scheme and then converts conserved values $(\rho, \rho\boldsymbol{u}, \rho E)$ to their primitive counterparts. This last step is required as fluxes are defined in terms of conserved variables, while LVR solves for the primitive ones.

3. Time Scales Model

In the development of LVR, we aimed at taking performance-oriented design decisions, without iterating through several tentative versions before satisfactory performance was achieved. Instead, we wanted to model our solver's performance ahead of time, even before an instruction stream existed. In order to do so, we developed the *Time Scales Model* performance model. Its main objective is not to offer accurate performance predictions, but rather to provide informative insights regarding the LVR's bottlenecks and performance upper-bounds. The model considers the main contributors to LVR's total execution time:



(a) Rates



(b) Time scales

Figure 1: Synthetic micro-benchmarks are used to profile data rates (left). The curves are absorbed in the symbolic framework. By accessing the coarse-grained view of the abstraction spectrum, the symbolic framework allows us to have accurate performance modelling in terms of the timescales involved in the simulation (right).

- floating-point operations,
- memory transfers,
- network transfers,
- CPU-GPU transfers.

We estimate the time spent by the solver on each of them, as follows:

$$t_{\text{compute}} = \frac{\text{FLOP count}}{\text{Peak Compute Performance}}, \quad (3)$$

$$t_{\text{transfer}} = \frac{\text{Payload size}}{\text{Rate(Payload size)}} \quad (4)$$

Here, t_{compute} represents the time spent on floating-point computations, while t_{transfer} corresponds to the time required for a given data transfer type. We compute one transfer time per transfer class listed above.

The FLOP count denotes the total number of floating-point operations required by the algorithm. *Peak Compute Performance* is the FP64 theoretical peak of the target platform. Payload size is the total volume of data moved through memory, the network, or the CPU–GPU interconnect (e.g., PCIe). Finally, Rate refers to the effective bandwidth for that transfer, expressed as a function of the payload size.

Estimating each of these quantities requires its own methodology, which we detail next.

Peak Compute Performance This quantity can be computed theoretically by accounting for the hardware characteristics of the target platform.

Under ideal conditions, that is, instruction streams rich in dependency-free FMA instructions, performance close to their theoretical peak can be reached. Consequently, our model relies on theoretical peak values and does not require empirical benchmarking to obtain them.

Rates Estimating data transfer rates is challenging for two main reasons. First, theoretical bandwidth values are rarely attained in practice, even under ideal conditions. For example, sustained memory bandwidth often reaches no more than about 80% of the theoretical peak. Second, as noted earlier, effective bandwidth strongly depends on the payload size, as illustrated in Figure 1a.

To account for these effects, we determine data rates through micro-benchmarks—small programs designed to measure specific transfer behaviors in isolation. We rely on standard well-established microbemchmarks when possible, and custom ones otherwise. By varying the payload sizes used in these benchmarks, we obtain rate estimates across a broad range of transfer sizes.

FLOP count For the numerical schemes used in LVR, for fluxes' computation, manually counting FLOPs is both tedious and prone to mistakes. Since we express kernels in a face-wise (for the FLUX kernel) and point-wise (for the DUC and DT kernels) fashion, we address this challenge through an ad-hoc symbolic framework built using SymPy. This allows us to express the numerical schemes directly as mathematical expressions. At this level of abstraction, FLOP counting can be fully automated, yielding accurate and efficient estimates.

Once the FLOP count of each kernel is computed, the solver's total FLOP count is derived from the problem size, i.e., the dimensions of the computational domain.

Payload sizes In LVR, payload sizes can be derived directly from the domain size, so no symbolic tool is required. For memory traffic, we assume a write-back, write-allocate

cache model. Under this model: 1) for each kernel, every input value is fetched from main memory only once, with all subsequent accesses served from cache; and 2) each write operation incurs an additional read, meaning that the number of written bytes must be counted twice when computing the payload size.

Given the complexity of this performance model—e.g., linking domain size to payload sizes and corresponding bandwidths—the computation of the resulting *Time Scales* is non-trivial and not reliably done by hand. Since these time scales depend on problem size, effective visualization is also required.
We therefore developed a lightweight framework to automate their computation and analysis, simplifying application of the model.
Once computed, interpretation is straightforward. The largest time scale ($t_{\max}$) identifies the dominant bottleneck, implying that the Time To Solution (TTS) cannot be smaller than this value.
This time scale can, in principle, reach peak performance. For any other time scale t_i, the maximum achievable fraction of peak performance f_i is:

$$f_i = \frac{t_i}{t_{\max}}, \tag{5}$$

By identifying the primary bottleneck and the maximum attainable peak fraction for each component, the Time Scales Model (TSM) generalizes the Roofline model. Unlike Roofline, TSM extends beyond compute and memory throughput and explicitly accounts for payload-dependent data rates.

3.1. Performance model-driven development

For the development of our compressible flow solver LVR, the TSM has been extensively used to take design decisions.
In the next subsections, we use our tools for calculating FLOP count and memory traffic from the numerical scheme, kernel variants and chosen layouts.

3.1.1 Domain Decomposition

In LVR, domain decomposition is leveraged for enabling simulations on multi-node systems. Here, the domain is subdivided among nodes to split the computational burden. At the interface between subdomains, data is shared to ensure a globally consistent procedure.
The choice of the numerical scheme determines how much data must be transferred. In the LVR case, given the 5^{th} order WENO scheme, for each boundary degree of freedom, 3 ghost degrees of freedom need to be retrieved from neighbor compute nodes.
The three domain decomposition schemes are: *Slabs*, *Pencil*, and *Cubic*. All of them, are based on splitting the global domain along either one, two or three axes. For each axis along which the domain is split, each node must exchange three layers of ghost degrees of freedom on each side with its neighboring nodes.
Slabs decompose the domain along the slowest-varying axis (z);
Pencil splits the domain along the two fastest-varying axes (y, z);
Cubic partitions the domain along all three axes. In Figure 3 and Figure 2 we compare LVR under these schemes using the TSM, on two different supercomputers: CSCS Alps (Figure 3) and ALCF Aurora (Figure 2), in both weak and strong scaling regime.

Weak scaling In the weak-scaling regime, network time is negligible relative to the other components, except on the two GPUs, where for subdomains of $\sim 10^7$ cells the compute time exceeds the network time under cubic decomposition. In all cases, slab decomposition delivers the best performance, while cubic remains the worst.
For sufficiently large subdomains, slab decomposition exchanges less data over the network. At these sizes, network payloads are large enough to saturate the available bandwidth in all three schemes, and compute and memory times coincide across decompositions.
These results indicate that slab decomposition is preferable for subdomains larger than $\sim 10^7$ cells, corresponding to domains of about $215 \times 215 \times 215$.

Strong scaling From Figure 3 and Figure 2, the slab decomposition outperforms the others by up to a factor of 3 up to ∼20 nodes on both

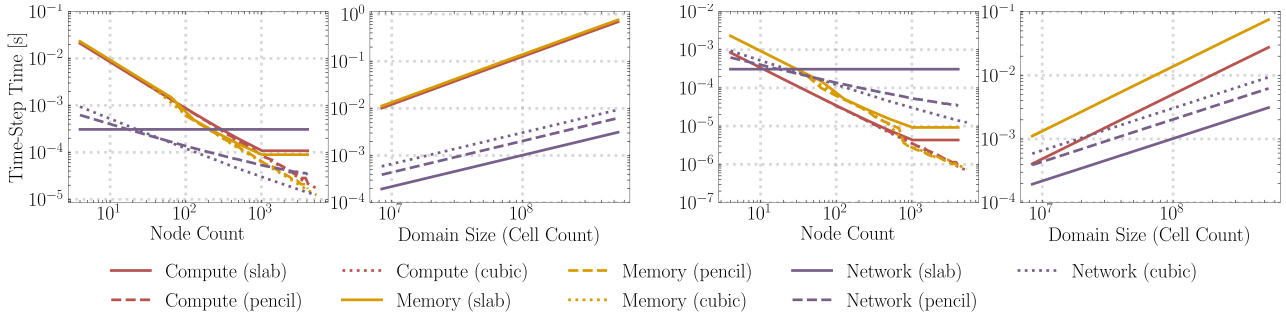


Figure 2: Plots (left → right): (1) Intel Sapphire Rapids CPU — strong scaling, (2) Intel Sapphire Rapids CPU — weak scaling, (3) Intel Max 1550 GPU — strong scaling, (4) Intel Max 1550 GPU — weak scaling. Each panel reports scaling performance for cubic, slab, and pencil domain decomposition schemes on a $256 \times 256 \times 1024$domain.

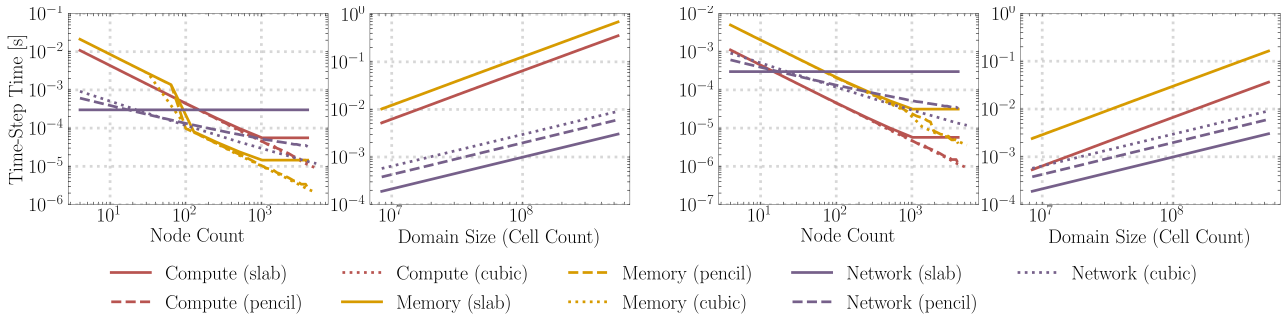


Figure 3: Plots (left → right): (1) NVIDIA Grace Superchip — strong scaling, (2) NVIDIA Grace Superchip — weak scaling, (3) NVIDIA H100 GPU — strong scaling, (4) NVIDIA H100 GPU — weak scaling. Each panel reports scaling performance for cubic, slab, and pencil domain decomposition schemes on a $256 \times 256 \times 1024$domain.

systems. Beyond this point, the cubic decomposition becomes superior. As node count increases and per-node subdomains shrink, cubic (and to a lesser extent pencil) benefits from reduced communication volume.

This indicates that for LVR, slab decomposition is optimal in weak-scaling regimes with large per-node workloads, whereas cubic performs best under strong scaling.

Given LVR's focus on high-throughput workloads, slab decomposition is adopted. It is also simpler to implement, requiring partitioning along a single dimension and limiting communication to at most two neighboring nodes.

3.1.2 Kernel fusion

Each substep executes two kernels: FLUX and DUC. FLUX consists of four stages—variable projection, WENO reconstruction, inverse projection, and the HLLC Riemann solver—while DUC performs divergence computation, the Runge–Kutta substep, and conversion from conserved to primitive variables.

Fusion into these two kernels was guided by symbolic FLOP analysis. Individually, all FLUX stages exhibit low operational intensity and are memory-bound. Fusing them reduces per-point memory traffic by avoiding repeated loads and stores and keeping intermediates in registers or cache. In contrast, the unfused version writes each stage's output to main memory, increasing traffic by over an order of magnitude. As a result, the fused FLUX kernel reaches an operational intensity of 9.19 F/B and becomes compute-bound.

The same reasoning applies to DUC: its divergence computation, Runge–Kutta update, and variable conversion are each memory-bound in isolation. Although fusion does not make DUC compute-bound, it lowers memory traffic and increases operational intensity, achieving up to 0.88 F/B.

3.2. Program synthesis

The symbolic framework used for performance modelling in section 3 is also exploited for code synthesis. Kernels are expressed as SymPy tree expressions, as functions of free symbols, that represent the kernel parameters. Such expressions would, however, contain several repeated subexpressions, resulting in an inflated FLOP count. Therefore, expressions are then fed to the SymPy Common Subexpression Elimination (CSE) routine. There, repeated subexpressions are found, and assigned to temporary symbols, which are then replaced in the final expression. As temporary symbols are never reassigned, the CSE output structure resembles the Single Static Assignment (SSA) format.

4. Results

LVR was tested to evaluate its accuracy and performance. As test-case, the Triple-Point shock problem was chosen. As platforms, the ALCF Aurora and the CSCS Alps supercomputers were used.

4.1. CSCS Alps

Scaling benchmarks consider up to 256 nodes (1024 GPUs, 1024 cores), and a per-node grid consisting of more than 2 billion points and 1 billions points for CPUs and GPUs, respectively. On the NVIDIA Grace Superchip, LVR achieves a fraction of peak FP64 compute performance up to 35%, and an overall performance up to 25%. These results are consistently attained up to 512 nodes, with only a negligible reduction in the two metrics as the node number increases. On the NVIDIA H100, the flux kernel shows a floating-point performance up to 40% of the nominal value, and up to 35% for the overall performance. Also in this case, performance does not significantly vary with the used number of nodes.

4.2. ALCF Aurora

We assessed *LVR* on weak scaling up to hundreds of Aurora nodes. Per node, a grid size of more than 2 billion points, and 1.5 billion points was used, on CPUs and GPUs, respectively. On 512 nodes, a 20.2 PFLOP/s compute performance was achieved. Weak scaling efficiency remains above 92% and 87% for CPUs and GPUs, respectively, even when several hundreds of devices are employed. Performance penalty emerges when the node count further increases, with a sharp drop for the GPU version when going from 64 to 128 nodes. In the dragonfly network topology of Aurora, nodes are divided in fully-connected groups of 64 nodes, and any two groups are connected by 2 links. As the node count increases beyond 64, a performance deterioration is therefore expected.

5. Conclusions

We presented an unconventional approach to CFD software development. Built in three months by a four-person team, our solver achieves a high fraction of peak performance across diverse architectures, including exascale systems.

This was enabled by a novel performance modeling methodology combining microbenchmarks with symbolic performance analysis (e.g., floating-point operation counting). The model guided key design decisions from the outset, including domain decomposition and kernel fusion. By providing realistic upper bounds and accounting for varying data rates, it proves a valuable tool for HPC development beyond CFD.

We also demonstrated how symbolic-enabled program synthesis improves both performance portability and productivity. The proposed workflow simplifies implementation of complex kernels while enabling platform-specific optimizations.

Future work includes refining code generation to better exploit instruction-level parallelism and extending the solver with advanced turbulence models, chemical kinetics, large eddy simulations, and support for complex curvilinear grids.

References

[1] D Rossinelli, T Trabacchin, A Voci, L Brown, H Collis, M Khanwale, T Zahtila, D Brouzet, and G Iaccarino. Crafting software in hpc: a quantitative approach. 2025.

[2] Henry Collis, Deniz A. Bezgin, Shahab Mirjalili, and Ali Mani. A thermodynamically consistent and robust four-equation model for multi-phase multi-component compressible flows using eno-type schemes including interface regularization, 2025.