ICCFD13-2026-338

A GPU-accelerated high-order modal Discontinuous Galerkin scheme with structure-preserving properties

L. Alberti*, E. Carnevali*, A. Crivellini*, and A. Colombo**
Corresponding author: e.carnevali@staff.univpm.it
* Marche Polytechnic University, Department of Industrial Engineering and Mathematical Sciences, Ancona, Italy.
**University of Bergamo, Department of Engineering and Applied Sciences, Bergamo, Italy.

Abstract: This work presents the GPU implementation of a scale-resolving, explicit modal discontinuous Galerkin (DG) solver for the compressible Navier–Stokes equations. Unlike nodal formulations, modal DG implementations on GPUs are less common in the literature and entail distinct implementation challenges and design choices. To this end, the CUDA extension for Fortran programming language is used to rewrite the solution routines to be offloaded to the available device. Notably, achieving substantial performance gains requires a convective flux formulation well-suited for the underlying multithreaded architecture. In particular, avoiding conditional branching and register pressure minimization are fundamental ingredients to optimally exploit the available hardware concurrency, also for CPU implementation,. As such, we adopt an entropy-conservative central flux which, when combined with the entropy projection-correction framework, further enforces the desired physical admissibility constraints on the underlying spatial discretization. We show that even in single precision computations this framework is sufficiently robust to allow entropy-conserving discretizations at low Mach numbers. Moreover, we highlight that the computational overhead introduced by the L_2 projection and residual correction account for about 20% of the baseline simulation time, making the strategy a feasible option to tackle scale-resolving problems.

Keywords: Discontinuous Galerkin, high-order, GPU, CUDA, high-performance computation.

1. Introduction

The arbitrary degree of polynomial approximation proper of Finite element methods, coupled with their strong geometrical flexibility, makes them a very well-suited numerical framework to tackle Computational Fluid Dynamics (CFD) simulations. Particularly, within Galerkin methods, the adoption of a broken finite-element test space produces a compact, local discretization that is highly suitable for the massive level of parallelization available in High Performance Computing (HPC) clusters. Nonetheless, the use of high-order polynomials in the solution approximation substantially increases the computational complexity of the method, as it requires the use of quadrature rules with high-degree of exactness in the integrals evaluation. In this optic, the adoption of hybrid hardware architectures poses a significant opportunity to exploit the massive parallelization offered by general-purpose Graphical Processing Units (GPGPUs) to tackle the large number of operations to be performed in high-order computations. Since the surge of interest in hybrid architectures, the CFD solver environments has been enriched by a significant number of GPU-aware schemes, such as [1, 2, 3, 4]. What we present here is the results of a GPU-porting of an explicit modal DG solver producing a high-order discretization of the compressible Navier-Stokes equations. In its baseline formulation the scheme discretizes the Navier-Stokes equations using the conservative set of unknowns as prognostic variables. As strongly compressible phenomena are considered, however, the approach starts suffering of severe robustness issues. To obviate this, an entropy-aware strategy based on a projection-correction approach may be efficiently embedded in the solver, at a moderate computational cost. The porting of the software in a GPU-accelerated version is

realized using the extension of the Fortran programming language provided by NVIDIA, i.e. CUDA Fortran. Within this context only the preprocessing activities remain on CPU, while the computationally intensive routines are offloaded to the device to take advantage of its high throughput. It is worth underlining that while the current scientific landscape mainly proposes GPU implementations of high-order *nodal* DG methods, what we present here is a *modal* DG solver which requires significantly different implementation strategies to fully and efficiently exploit the computational resources of modern GPU architectures. The paper present the following structure: Sec. 2 outlines the employed numerical method, Sec. 3 collects an overview of the designed CPU and GPU kernels, Sec. 4 reports on the derived results in terms of achieved speed-up and result accuracy, finally conclusions are briefly reported in Sec. 5.

2. Numerical Method

The finite-element discretization of the Navier-Stokes equations (NSE), to be solved within an arbitrary physical domain $\Omega \in \mathbb{R}^d$, starts from the weighted-residual formulation

$$\int_\Omega \boldsymbol{\Phi}^\top \frac{\partial \mathbf{q}}{\partial t} d\Omega \quad - \quad \int_\Omega \left(\frac{\partial \boldsymbol{\Phi}}{\partial x_i} \right)^\top \mathbf{F}_i d\Omega + \int_{\partial\Omega} \boldsymbol{\Phi}^\top \mathbf{F}_i n_i d\sigma = 0 \,, \tag{1}$$

where $i = 1, \ldots, d$, $\boldsymbol{\Phi} \in \mathbb{R}^{2+d}$ denotes a suitable smooth test function and $\mathbf{F}_i = \mathbf{F}_{i,c}(\mathbf{q}) + \mathbf{F}_{i,v}(\mathbf{q}, \nabla\mathbf{q})$ is the global flux in the $i-$th Cartesian direction. A semi-discretized form is then obtained upon substituting the physical domain with the triangulation $\mathcal{K}_h = \{K\}$, made of of mesh elements K and faces F, with $\mathcal{F}_h = \{F\}$. Particularly, to derive a DG discretization the continuous functions $\mathbf{q}$ and $\boldsymbol{\Phi}$ are replaced with the discontinuous finite-element approximations $\mathbf{q}_h, \boldsymbol{\Phi}_h \in [\mathcal{V}_h]^{d+2}$, where

$$\mathcal{V}_h = \left\{ \phi \in L_2(\Omega) : \phi|_K \in \mathbb{P}_d^p(K), \forall K \in \mathcal{K}_h \right\} \tag{2}$$

identifies the space spanned by the chosen set of piecewise-continuous, $p-$th degree $d-$dimensional polynomial functions, defined according to [5]. By doing so, the following element-wise semi-discrete system

$$\int_K \boldsymbol{\Phi}_h^\top \frac{d\mathbf{q}_h}{dt} d\Omega - \int_K \left(\frac{\partial \boldsymbol{\Phi}_h}{\partial x_i} \right)^\top \mathbf{F}_{h,i} d\Omega + \oint_{\partial K} \boldsymbol{\Phi}_h^\top \hat{\mathbf{F}}_{h,i} n_i d\sigma = 0, \tag{3}$$

is derived. The locality of the DG approximation effectively decouples the degrees of freedom (DoF) of adjacent elements, since the basis functions used to approximate the solution have compact support on each element K. As a consequence, the solution continuity is enforced only in a weak sense by replacing, at each mesh face $F \in \partial K$, the physical flux $\mathbf{F}_{h,i}$ with a numerical counterpart $\hat{\mathbf{F}}_{h,i}$, evaluated along the face-normal direction and depending on the solutions of the elements sharing the considered face. As far as the convective flux discretization is concerned, it is possible to demonstrate, see [6], that upon computing the discrete space operator as a function of a set of entropy variables $\mathbf{v}_h$ [7], obtained as L_2 projection of the conservative set $\mathbf{q}_h$, and introducing an explicit residual correction for the entropy [8],

$$\alpha_K = \int_K \left(\frac{\partial \mathbf{v}_h}{\partial x_i} \right)^\top \mathbf{F}_{h,i,c}(\mathbf{v}_h) d\Omega - \oint_{\partial K} \psi_i n_i d\sigma \tag{4}$$

distributed across the solution DoF as

$$\alpha_K \frac{\int_K \boldsymbol{\Phi}_0^\top \mathbf{v}_{0,h} d\Omega}{\int_K \mathbf{v}_{0,h}^\top \mathbf{v}_{0,h} d\Omega} \tag{5}$$

results in a DG space discretization that ensures entropy conservation/stability (EC/ES) upon suitable choice for the convective part of the numerical flux, i.e. $\widehat{\mathbf{F}}_{h,i,c}(\mathbf{v}_h^\pm) n_i$. Particularly, in this contribution we will consider both the ES Godunov flux of [9] and the KEP-EC flux of Ranocha [10]. Notably, as the

chosen modal approximation makes use of orthonormal base functions, the distribution of Eq. (5) equates to altering each out-of-mean DoF of a quantity proportional to its value, normalized by the squared L_2 norm of the perturbed solution, multiplied by α_K. It is worth noting that while in a CPU implementation the element-wise computation of α_K is straightforward, when moving to GPU systems the multi-threaded nature of the computation forces threads synchronization. Particularly, atomic operations are required for the parallel assembling of the elemental value of Eq. (4).

As far as the viscous contribution of the numerical flux is concerned, this is computed on the base of the elliptic operator discretization provided by the BR2 approach of Bassi and Rebay [11, 12]. Within our entropy-projection framework, the BR2 scheme adopts a centred numerical flux

$$\hat{\mathbf{F}}_{h,i,v}, n_i = \{\mathbf{F}_{h,i,v}\left(\mathbf{v}_h, \nabla_h \mathbf{v}_h + \eta_F \mathbf{r}_F(\llbracket \mathbf{v}_h \rrbracket)\right)\} n_i, \tag{6}$$

based on the definition of a local $\mathbf{r}_F \in [\mathcal{V}_h]^d$

$$\sum_{K \in \mathcal{K}_h} \int_K \boldsymbol{\tau}_h^\top \mathbf{r}_F(w)\, d\Omega = \int_F w \left\{\boldsymbol{\tau}_h\right\}^\top \mathbf{n}_F\, d\sigma \quad \forall \boldsymbol{\tau}_h \in [\mathcal{V}_h]^d, \forall w \in L^2(F) \tag{7}$$

and global $\mathbf{r} \in [\mathcal{V}_h]^d$ lifting operators

$$\mathbf{r}(w) = \sum_{F \in \mathcal{F}_h} \mathbf{r}_F(w), \tag{8}$$

used to discretize the volume and face solution gradient. The adopted scheme, hence, compute the volume viscous flux considering the *global* lift of the element discontinuities in Eq. (8), while adopting a *local* symmetric jump penalization approach in the computation of the interface normal fluxes. Similarly to what observed for the α_K construction, we note that the assembling of the global lift operator requires the adoption of atomic operations when being assembled in CUDA multithread parallelism, as different thread-blocks may concurrently update the global lift of a given elements.

It is worth pointing out that although the convective and viscous discrete operators in the above relations are written in terms of the L_2-projected entropy state $\mathbf{v}_h$, the baseline conservative implementation computes the space discrete operators using the variables set $\mathbf{q}_h$.

Finally, concerning the time integration algorithm, we employ the explicit Strong Stability Preserving third-order three-stage Runge-Kutta scheme of [13], herein referred to as SSPERK33. With this, the semi-discrete non-linear system resulting from the DG discretization (3), compactly represented as

$$\frac{d\mathbf{Q}}{dt} + \mathbf{R}(\mathbf{Q}) = \mathbf{0}, \tag{9}$$

is advanced in time as

$$\begin{aligned} \mathbf{Y}_i &= \mathbf{Q}^n + \Delta t \sum_{j=1}^{i-1} a_{ij} \mathbf{R}\left(\mathbf{Y}_j\right), \\ \mathbf{Q}^{n+1} &= \mathbf{Q}^n + \Delta t \sum_{j=1}^{s} b_j \mathbf{R}\left(\mathbf{Y}_j\right), \end{aligned} \tag{10}$$

where s is the number of stages and a_{ij}, b_j are the real coefficients reported in [13].

3. Algorithms Description

The optimization strategies adopted for CPU and GPU architectures are shaped by the parallel execution models exposed by the two different platforms. Modern CPUs exploit data-level parallelism through

SIMD (Single Instruction Multiple Data) vector instructions, where a single instruction operates simultaneously on multiple data elements packed into vector registers, effectively multiplying the number of instructions issued per clock cycle. To date, Intel hardware architectures implements SIMD instruction sets which spans register widths from 128 (IntelSSE) to 512 (IntelAVX-512) bits. As such, the highest of these level of vectorization allows to simultaneously process eight 64-bit operands or sixteen 32-bit operands depending on the adopted floating point precision. As modern multicore architectures further allocate more than one FMA (Fused Multiply-Add) unit in each core, the full exploitation of SIMD data parallelism theoretically ensures to perform $\#\text{FMA} \times 2 \times 8$ 64−bit floating point operations at each clock cycle, effectively multiplying the concurrent amount of operations and hence decreasing, at least theoretically, the computational time required by HPC calculations. However, to ensure a vectorized code to achieve high computational efficiency, careful attention has to be directed towards the organization of both computational kernels and memory layouts. Particularly, to ensure effective vectorization requires structuring the memory layout to maximize data alignment, so that contiguous data may be packed into SIMD registers, on top of ensuring absence of data dependencies between lanes.

As far as GPUs are concerned, these rely on a Single Instruction Multiple Threads (SIMT) execution model, in which a large number of threads execute concurrently in groups, *warps* within the CUDA environment, following a strict lockstep execution paradigm. Briefly, NVIDIA devices are constituted by arrays of fundamental hardware units, called Streaming Multiprocessors (SM), containing the architectural resources necessary to schedule, issue and execute the kernel instructions. When offloading a kernel, the workload is distributed across multiple threads hierarchically organized in blocks. These, in turn, are distributed across the hardware SMs which further partition them into 32-thread groups, the *warps*, constituting the fundamental execution units. At each clock cycle the warp schedulers on the SM issue one among the eligible warps, which gets mapped to the SM execution partitions, simultaneously executing the same instruction on data points mapped, at software level, by each individual thread in the warp. Hence, one may think of SMs execution partitions as SIMD lanes, each executing the issued instruction on memory positions and data operands associated to the thread identification. As such, rather than increasing performance through wide vector units, as in CPUs, GPUs achieve high throughput by distributing computations across a hierarchy of thread blocks, while hiding memory latency through massive concurrency and the rapid context-switching between warps.

3.1 CPU algorithm

In this section we briefly mention the structure of the main solution routines, leaving a detailed report for future publication. The numerical integration is performed processing padded block data, with padding size ng_block depending on the adopted precision in the computation. Since SIMD execution operates on vector register of given width, 512−bit for AVX-512, the block size is chosen to allocate exactly $512/s_{\text{FP}}$ data element per block, where s_{FP} is the bit size of the single floating-point datum at chosen precision. Similarly, a padded approach is also adopted for the solution DoFs, e.g. N_{DoF}. Particularly, given the number of base functions to be used in representing the solution using complete Pascal polynomial basis, i,e. $N_{DoF} = \binom{p+d}{d}$, the latter is padded as $((N_{DoF} + s_{\text{FP}} - 1)/s_{\text{FP}}) \cdot s_{\text{FP}}$, so that the reduction cycle over N_{DoF} may be exactly distributed over an integer number of subsequent SIMD execution. Moreover, to avoid overheads associated to the horizontal reduction that would be triggered in SIMD execution during the computation of solution and fluxes at the block integration points, a choice we made was to duplicate the base functions array defined at element faces to adopt, together with a standard DoF-based indexing, also a block-indexed structure. The latter data layout relies on an intra-element Structure of Arrays (SoA) approach, tailored to the number of points that can be concurrently handled by the SIMD hardware. The Array of Structures (AoS) organization is utilized instead when vectorizing operations over the degrees of freedom proves more efficient—for instance, during the assembly of the residual after the solutions and fluxes have been evaluated at the face and volume quadrature points. This enforces data-independence in the reduction operations of the modal contraction, used to recover values at integrations points, guaranteeing the vectorization of the instruction.

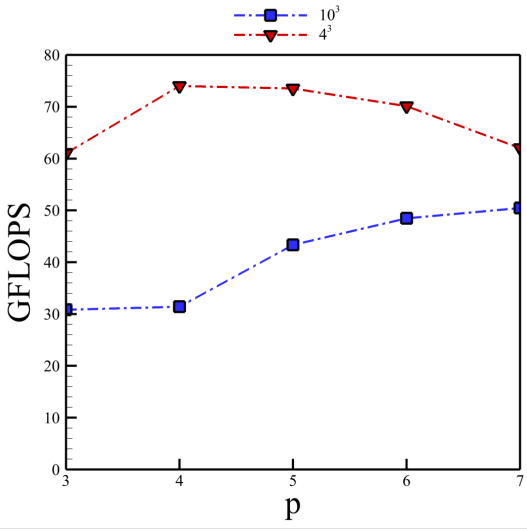

Figure 1: Roofline performance. The results are obtained via the Advisor profiler provided by Intel oneAPI.

In this sense, it is worth noting that for the flux computation to be correctly vectorized requires the same instruction to be issued on all the integration points being processed simultaneously by the vector register. This, however, introduces a constraint on the choice of the numerical scheme to be used. Particularly, for the convective part it is clear that the implementation of a Riemann-like solver would break SIMD execution due to the presence of multiple diverging branches during selection of the solution connecting the two neighbouring states. As a consequence, for the convective flux we adopt the KEP-EC flux of Ranocha [10]. Its combination with our entropy-aware framework, indeed, result in a EC discretization of the convective operator, which has been demonstrated to yield considerably accurate results, at least for low-Mach configurations, in condition of severely under-resolved space discretization [14].

In Fig. 3.1 we provide, for the CPU code, a roofline model performance evaluation, performed on the LEONARDO Datacentric partition [15]. The portrayed data refers to single-core performance, at single precision arithmetic, evaluated on simulations over a fixed number of time steps of a chosen benchmark problem, namely the Taylor-Green vortex case. Polynomial orders spanning $p = 3, \dots, 7$, on a coarse and fine mesh having respectively 4^3 and 10^3 elements, are reported. We note that on coarse mesh the algorithm achieve its peak performance at polynomial order $p = 4$, reaching the 31% of the theoretical machine roofline of 240 GFLOP/s. From this point, we note a steady decrease of efficiency as the polynomial order increases. It is reasonable to interpret this as an effect of increased memory traffic to higher latency memory levels due to larger working sets, not fitting entirely on low-level cache. On the contrary, for finer mesh, we note an opposite behaviour, coherent with the memory-bound behaviour of higher-order methods, namely the increase of computational performance at increasing polynomial order. We wish to emphasize that the performance result reported in Fig. 3.1 refers to the whole algorithm rather than single kernels; as such, it is worth noting that achieving over 30% of the theoretical peak FLOP/s, counting two floating-point operations per Fused Multiply-Add (FMA) instruction, across the entire application execution, rather than an isolated kernel, demonstrates an excellent level of optimization. Furthermore, on older CPUs such as the Intel Core i7-6700, which are equipped only with AVX2 instructions (yielding half the vector width of AVX-512), performance can reach up to 40% of the theoretical peak FLOP/s. These values represent approximately a twofold or greater increase over the performance recorded prior to the optimizations implemented in this work. This optimized CPU performance will serve as the baseline for evaluating the GPU performance discussed later. Ultimately, to ensure a fair and accurate assessment of the speedup provided by the GPU architecture, it was deemed necessary to fully optimize the CPU performance first. The compiler employed is Intel's new ifx compiler, natively targeting the Intel CPUs tested here; this unified approach mirrors that of NVIDIA, which similarly provides both the hardware

and the software stack for its HPC and consumer GPUs. Additionally, specific Intel-proprietary directives were utilized to assist the vectorization process. Evaluating performance with alternative compilers and across different CPU vendors remains a subject for future work. Finally, it should be noted that the excellent parallel performance of the solver, leveraging non-blocking MPI communications, has already been demonstrated in previous works (see, for example, [16]). Consequently, the analysis presented in this section has focused solely on single-core performance.

3.2 GPU algorithm

Concerning the GPU algorithm, while an in-depth discussion is again left for future work, we note that the algorithm structure is designed to exploit the execution model of CUDA architectures. Particularly, the solution kernels distribute faces and volumes throughout multiple thread blocks, each responsible for the processing of one or more of the mapped geometrical elements. Within each block, CUDA threads are first mapped to integration points for the population of a shared array storing local solution values. Once the shared arrays are populated, the CUDA threads are mapped, within the block of integration points being processed, to one or more solution DoFs. In this phase each thread is responsible to perform the numerical integration of the term $\int \phi_j \mathbf{R}_j$, with j being the one-to-one mapped DoF to each active, eligible thread. Similarly to the CPU kernels, also in the GPU porting we make use of duplicated shape functions arrays for each mesh face, with, however, slight differences in the data organization. Particularly, for the first phase in each integration block, when all threads in the warp recover the values of the solution at local integration points, the modal reduction employs shape function arrays with ordering (N_{DoF}, ng, nf). In the subsequent phase, when the multithread parallelism is distributed across the residual DoFs, the more suitable organization of shape function array (ng, N_{DoF}, nf) is employed. Clearly, this particular organization of operations is made necessary by the underlying modal approximation, requiring at each integration point a modal reduction to recover the solution value. In conclusion, also for the GPU implementation achieving high computational performances relies on an adequate choice for the interface convective numerical flux, with Riemann-like solvers having severe impact on the throughput performance due to the compound effect of branch divergence and large memory register pressure. As such, also for the GPU algorithm the fastest achieved implementation relies on the adoption of the central KEP-EC flux of [10].

Table 1: Simulation times (s) for single device performance, baseline scheme implementation.

		A100			RTX4070			RTX3090			CPU		
		P3	P4	P5	P3	P4	P5	P3	P4	P5	P3	P4	P5
Mesh	40^3	19.4	56.1	140.4	31.0	101.6	–	24.6	81.1	–	133.7	437.7	812.2
	60^3	64.5	187.3	–	–	–	–	83.4	–	–	460.9	1458.7	–
	80^3	150.0	–	–	–	–	–	–	–	–	1082.1	–	–
	100^3	296.5	–	–	–	–	–	–	–	–	2087.2	–	–

4. Numerical Results

The performance of the numerical scheme is assessed using the low-Mach Taylor-Green vortex benchmark [17] at $Re = 1600$. The numerical simulations are performed on three mesh levels discretizations of the triple-periodic domain $[-\pi, \pi]^3$, i.e. $40^3, 60^3, 80^3$ and 100^3, employing polynomial approximation orders of $p = 3, 4, 5$. On top of solution accuracy, in this section we further presents the speed-up achieved by the GPU implementation relative to the CPU optimized version. This metric is assessed through a side-by-side comparison across different GPU cards, covering both HPC-level and customer-grade hardware. As a reference, we use the host CPU system provided by LEONARDO Booster partition, i.e. the Intel Xeon Platinum 8358 32-cores processor. Concerning the GPU cards, alongside the NVIDIA A100 accelerators [18, 15] available on the LEONARDO cluster, we further consider two customer-grade

NVIDIA GPU cards provided by our in-house computing infrastructures, namely an RTX4070 (12GB) and an RTX3090 (24GB). To fairly evaluate the obtained speed-up we limit here the discussion to single device performance on single precision arithmetic, this in order to both fully exploit the computational capabilities of GPU devices and simultaneously avoid the strict memory limit of consumer-grade cards.

4.1 Speed-up results

The comparison between GPU and CPU algorithms is performed in their baseline conservative version, as this entails the lowest computational cost within the solution procedure and yields so the fastest algorithmic variant. The results, in terms of elapsed simulation time per 1000 time steps, are reported on Tab. 1. Due to the lesser robustness properties of the baseline scheme, however, the straightforward implementation of the KEP-EC flux is only possible within our projection-correction framework. As such, the numerical convective flux of the baseline scheme is additionally equipped with an upwind bias in the form of

$$-\max(\lambda^+, \lambda^-)\frac{\mathbf{q}_h^+ - \mathbf{q}_h^-}{2}\,, \tag{11}$$

with $\lambda^\pm$ identifying the largest convective eigenvalue for the left/right element solution.

Table 1 shows the GPU implementation to reach up to a 8× speed-up on the A100 card over the CPU hardware, with the RTX3090 realizing almost 6× peak acceleration. It is particularly revealing to see that on the smallest among the tested workloads, i.e. the 40^3 mesh at $p = 3$, the simulation times between the A100 and the RTX3090 are remarkably similar, underscoring the potential of exploitation of high-tier consumer-grade cards.

Since the analysis thus far has focused exclusively on single-card simulations, in Fig. 4.1 we supplement the reported data with the parallel scaling results obtained on the LEONARDO cluster [15] using a CUDA-aware MPI implementation. Here, it may be observed that efficiency remains above 60% up to 128 GPUs. The sudden fall is attributable to the excessive workload decrease; as for 256 GPUs each device receives less than 4000 mesh elements to process, making communication overhead significant. To prove this, we performed a simulation on a much finer mesh grid of 200^3 elements, which we ran on 128 and 256 GPUs, achieving a parallel efficiency of 93%.

To conclude the computational efficiency section, we quantify the overhead introduced by the projection-correction framework with respect to the baseline scheme. Table 4.1 reports, for the 40^3 and 80^3 meshes, the ratio of wall-clock times required to perform 1000 iterations, at polynomial orders $p = 3, 4, 5$, by the two algorithms. Overall, the compound effect of the L_2 projection and the construction of the

	$40 \times 40 \times 40$	$80 \times 80 \times 80$
P3	0.81	0.81
P4	0.83	0.83
P5	0.84	0.83

Table 2: Ratio $t_{\mathbf{q}_h}/t_{\mathbf{v}_h,\alpha_k}$ considering 1000 iterations of the third-order, three stage SSP explicit Runge-Kutta scheme [13]. The data refers to simulations performed on the A100 hardware.

entropy correction α_K only accounts for about a 20% increase in the computational cost over the baseline scheme. The trade-off, however, is a significant gain in robustness which enables to undertake challenging compressible flow simulations, with GPU results which we defer to future publications.

4.2 Accuracy results

In this section we compare the simulation results obtained by our GPU implementation, performed in single precision arithmetic, with the scale-resolved data of [17]. We start by noting, again, that the implementation of an entropy-conserving flux requires a sufficiently robust underlying space discretization to ensure the stability of the computation, with the baseline conservative formulation requiring the addition

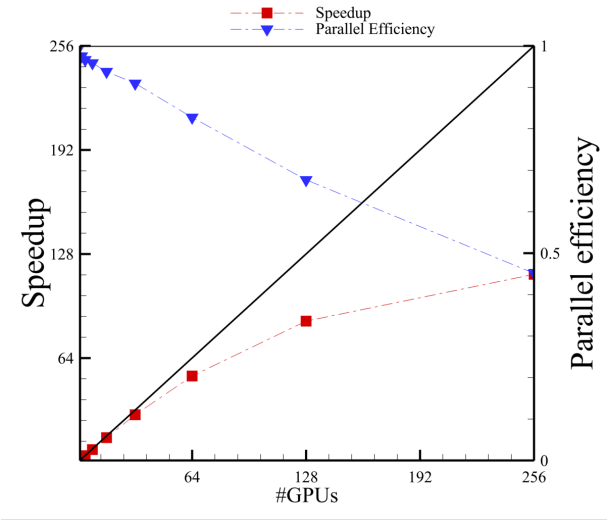

Figure 2: Parallel scalability performance. Simulations performed on a 100^3 mesh at polynomial order $p = 3$.

of an upwind-bias to avoid simulation crash. On the other hand, the projection-correction framework is able to reach solution end-time without introducing any additional dissipation. Concerning the quality of the derived solution, we note that our single precision result aligns quite accurately on the DNS data, both in terms of kinetic energy decay and total dissipation. Particularly, it appears that for coarser space discretization the straightforward implementation of the KEP-EC flux retains a significantly higher level of accuracy with respect to a stable formulation, a result also outlined in [14].

5. Conclusion

This contribution presented the GPU porting of a high-order modal DG solver featuring a structure-preserving formulation for entropy. Single-device performance was evaluated using the classic Taylor-Green vortex test case, yielding a substantial speed-up of up to 7× over a heavily vectorized CPU baseline. Additionally, parallel scalability tests conducted on the LEONARDO supercomputer [15] exhibited near-optimal strong scaling up to 256 GPUs. These findings confirm the suitability of the proposed DG methodology for large-scale, high-order CFD simulations on modern GPU-accelerated architectures.

Acknowledgements

We acknowledge ISCRA for awarding this project access to the LEONARDO supercomputer, owned by the EuroHPC Joint Undertaking, hosted by CINECA (Italy).

References

[1] Lucas Gasparino, Filippo Spiga, and Oriol Lehmkuhl. Sod2d: A gpu-enabled spectral finite elements method for compressible scale-resolving simulations. *Computer Physics Communications*, 297:109067, 2024.

[2] Nico Krais, Andrea Beck, Thomas Bolemann, Hannes Frank, David Flad, Gregor Gassner, Florian Hindenlang, Malte Hoffmann, Thomas Kuhn, Matthias Sonntag, et al. Flexi: A high order discontinuous galerkin framework for hyperbolic–parabolic conservation laws. *Computers & Mathematics with Applications*, 81:186–219, 2021.

[3] Srikanth Sathyanarayana, Matteo Bernardini, Davide Modesti, Sergio Pirozzoli, and Francesco

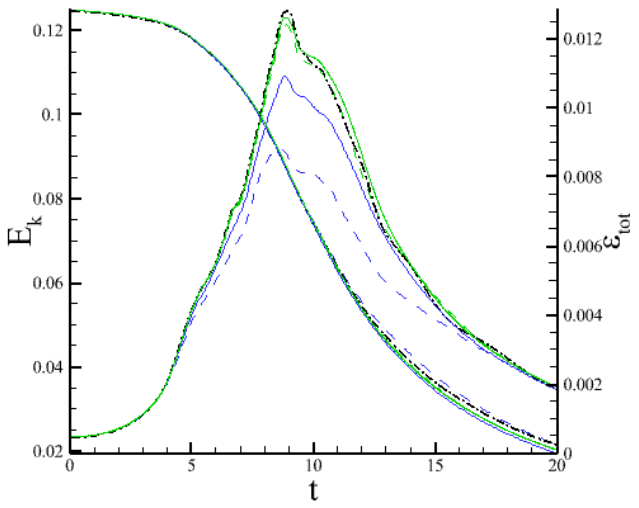

Figure 3: Simulations performed at polynomial order $n = 4$ on mesh 40^3 (blue) and mesh 100^3(green). Dashed lines refers to solution obtained with baseline conservative scheme and upwind-biased Ranocha flux, while contiguous line refers to the projection-correction framework implementing the standard Ranocha flux. DNS dataset of [17] (-·-·).

Salvadore. High-speed turbulent flows towards the exascale: Streams-2 porting and performance. *Journal of Parallel and Distributed Computing*, 196:104993, 2025.

[4] Henry Waterhouse, Maciej Waruszewski, Lucas C Wilcox, and Francis X Giraldo. Gpu performance of an entropy-stable discontinuous galerkin euler solver with non-conservative terms. *arXiv preprint arXiv:2605.16684*, 2026.

[5] Francesco Bassi, Lorenzo Botti, Alessandro Colombo, Daniele A Di Pietro, and Pietro Tesini. On the flexibility of agglomeration based physical space discontinuous galerkin discretizations. *Journal of Computational Physics*, 231(1):45–65, 2012.

[6] Luca Alberti, Emanuele Carnevali, Alessandro Colombo, and Andrea Crivellini. An entropy conserving/stable discontinuous galerkin solver in entropy variables based on the direct enforcement of entropy balance. *Journal of Computational Physics*, 508:113007, 2024.

[7] Thomas JR Hughes, Leopaldo P Franca, and Michel Mallet. A new finite element formulation for computational fluid dynamics: I. symmetric forms of the compressible euler and navier-stokes equations and the second law of thermodynamics. *Computer methods in applied mechanics and engineering*, 54(2):223–234, 1986.

[8] Rémi Abgrall, Philipp Öffner, and Hendrik Ranocha. Reinterpretation and extension of entropy correction terms for residual distribution and discontinuous galerkin schemes: application to structure preserving discretization. *Journal of Computational Physics*, 453:110955, 2022.

[9] J J Gottlieb and C P T Groth. Assessment of Riemann Solvers for unsteady One-Dimensional Inviscid Flows of Perfect Gases. *Journal of Computational Physics*, 78:437–458, 1988.

[10] Hendrik Ranocha. Entropy conserving and kinetic energy preserving numerical methods for the euler equations using summation-by-parts operators. *Spectral and high order methods for partial differential equations ICOSAHOM 2018*, 134:525–535, 2020.

[11] Franco Brezzi, Gianmarco Manzini, Donatella Marini, Paola Pietra, and Alessandro Russo. Discontinuous galerkin approximations for elliptic problems. *Numerical Methods for Partial Differential Equations: An International Journal*, 16(4):365–378, 2000.

[12] Francesco Bassi and Stefano Rebay. High–order accurate discontinuous finite element solution of the 2D Euler equations. *Journal of Computational Physics*, 138(2):251–285, 1997.

[13] Raymond J Spiteri and Steven J Ruuth. A new class of optimal high-order strong-stability-preserving

time discretization methods. *SIAM Journal on Numerical Analysis*, 40(2):469–491, 2002.

[14] Luca Alberti, Emanuele Carnevali, Alessandro Colombo, and Andrea Crivellini. An entropy-aware discontinuous galerkin solver for implicit large eddy simulations of wall-bounded flows. *Physics of Fluids*, 37(7), 2025.

[15] Matteo Turisini, Giorgio Amati, and Mirko Cestari. Leonardo: A pan-european pre-exascale supercomputer for hpc and ai applications. *arXiv preprint arXiv:2307.16885*, 2023.

[16] Andrea Crivellini, Matteo Franciolini, Alessandro Colombo, and Francesco Bassi. Openmp parallelization strategies for a discontinuous galerkin solver. *International Journal of Parallel Programming*, 47(5):838–873, 2019.

[17] Thibault Dairay, Eric Lamballais, Sylvain Laizet, and John Christos Vassilicos. Numerical dissipation vs. subgrid-scale modelling for large eddy simulation. *Journal of Computational Physics*, 337:252–274, 2017.

[18] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro*, 41(2):29–35, 2021.