HyperCODA – Towards high-performing time-resolving flow simulations

Johannes Wendler[1], Immo Huismann[1], Ronny Tschüter[1], and Stefan Fechter[2]

[1] German Aerospace Center (DLR),
Institute of Software Methods for Product Virtualization,
Dresden
Johannes.Wendler@DLR.de, Immo.Huismann@DLR.de, Ronny.Tschueter@DLR.de
[2] German Aerospace Center (DLR),
Institute of Aerodynamics and Flow Technology,
Göttingen
Stefan.Fechter@DLR.de

Abstract. The present work focuses on the performance analysis of the DG-SEM implementation of the CFD solver CODA. The turbulent Taylor-Green vortex is employed as a simple testcase for scaling behavior, while for a more detailed node-level performance analysis more granular kernel benchmarks are used. Bottlenecks in the implementation are highlighted and possible solutions proposed.

Keywords: DG-SEM, CFD, performance analysis, optimization

1 Introduction

While steady-state RANS simulations are the state of the art for flow simulation in industry, they often mispredict the resulting flow patterns [1][2]. Scale-resolving and time-resolving simulations offer a way out of the predicament at the expense of resources. Not only does the flow have to be resolved on a per-timestep basis, but low diffusivity is also required. This is where high-order methods can shine.

From the available high-order methods, the nodal discontinuous Galerkin spectral-element method (DG-SEM) offers speed and scalability while allowing for shock capturing and coupling with finite-volume methods [3, 4]. The speed results from two factors: On the one hand, tensor-product operators can be well optimized such that the code stays compute-bound, not memory bound [5]. On the other hand, well-crafted DG-SEM methods can scale up to one element per core [6]. In order for these methods to jump the gap from academia to industry, CODA implements a DG-SEM method. CODA is the computational fluid dynamics (CFD) software being developed as part of a collaboration between the French Aerospace Lab ONERA, the German Aerospace Center (DLR), Airbus, and their European research partners. CODA is jointly owned by ONERA, DLR and Airbus.

While the DG-SEM method allows for reaching peak performance, it is often not reached in practice. Meticulous performance engineering is essential here [8].

For this purpose we analyze the current DG-SEM implementation of CODA and HyperCODA [9] with the turbulent Taylor-Green vortex as a simple testcase for scaling behavior and perform more granular benchmarks to evaluate the node level performance of the most relevant kernels using Score-P [12] and LIKWID [13]. This analysis reveals some inefficiencies and we propose possible optimization strategies.

2 Test case

The focus of this paper is the performance evaluation of CODA to enable efficient simulations of complex use cases with the DG-SEM method. As a simple test case we chose the turbulent Taylor-Green vortex. It is a benchmark from the 1st International Workshop on High-Order CFD methods, test problem 3.5 [14]. The time-resolved simulation of the Taylor-Green vortex transitions to fully turbulent at the chosen Reynolds number of $Re = 1600$ [10, 11].

The Taylor-Green vortex test case is simulated with the flow solver CODA, using the DG-SEM discretization implemented using the methods for shock capturing and multispecies extension by Renac [17] and Marmignon et. al. [18]. As discretization, we chose DG-SEM of order $n = 4$, meaning each element contains four points per direction. The computation is performed on a cubical domain of 64^3 elements until $20T$ was reached. $T = 1/u_0$ represents the characteristic time scale of the flow, where u_0 is the amplitude of the periodic initial velocity. For time stepping a fourth-order, time-explicit Runge-Kutta scheme with constant time step size during the whole simulation is employed. As typical with DG-SEM, an "implicit LES" is used, i.e. the inherent dissipation of the discretization serves as turbulence model [6]. Furthermore, the Roe scheme in conjunction with the second scheme of Bassi and Rebay (BR2) [15, 16] were employed.

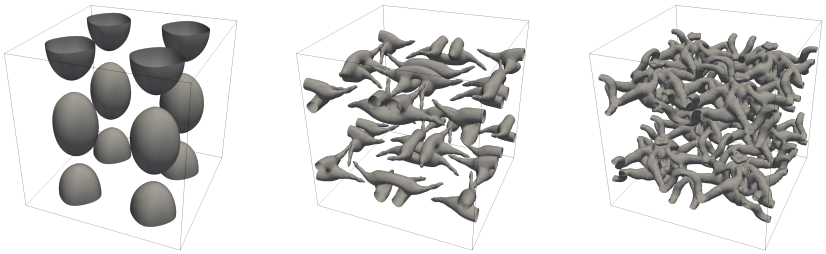

Fig. 1. Vortex structures in the Taylor-Green vortex at different times, visualized via isoplanes of negative relative pressure of $p - p_0 = -0.002$. Left: $t = 0 \cdot T$, middle: $t = 7 \cdot T$, right: $t = 11 \cdot T$.

Figure 1 depicts the resulting vortex structures, their decay and generation of smaller structures. Here, isoplanes of negative pressure visualize the vortex structures. Initially, the periodic, analytic pressure distribution can be observed,

until $t = 5T$, where vortices form and join. At $t = 10T$, the flow becomes fully turbulent with the large-scale vortices decaying into ever-smaller ones.

3 Performance analysis

To investigate the performance of the DG-SEM implementation in CODA we performed benchmarks on the DLR cluster CARO. The machine has two sockets per node, each with an AMD EPYC 7702, running at a fixed clock speed of 1.8 GHz for all test runs. All used binaries were compiled with GCC 10.3.0, using

```
-O3 -march=native --param inline-min-speedup=1 --param
    inline-unit-growth=200 -DNDEBUG
```

as additional compile flags and with disabled asserts in the build system for CODA version `2022.07.0-137`.

3.1 Scaling behavior

As a baseline, we performed a weak scaling analysis with 10 timesteps of the turbulent Taylor-Green vortex test case from 1 to 32 nodes. The number of elements per node is kept constant and the size is roughly 2.5 times the size of the L3 caches to stay in memory.

Figure 2 depicts the runtime per element and the parallel efficiency for the DG-SEM orders $n = 4$, $n = 8$ and $n = 16$. Each node having the same amount of data should, in the best case, keep the runtime constant. The linear increase of total elements w.r.t. the number of nodes then leads to a linear decrease in runtime per element. This exact behavior is represented in the plot of runtime over the number of nodes. Also the parallel efficiency stays at the ideal 100 % mark. From these two metrics of this benchmarks, we conclude that the parallelization and communication parts of the implementation are no bottleneck to consider for optimization.

3.2 Identification of critical Kernels

For the identification of critical kernels, we use a version of CODA instrumented by Score-P [12]. The turbulent Taylor-Green vortex benchmark was rerun with a DG-SEM of order $n = 4$ on a cubic domain of size 64^3. Profiling data (aggregated runtime of functions) and a trace (time resolved data about communication and computation patterns) were extracted. The runtime overhead compared to the original simulation was below 2 %, meaning the data depicts an accurate representation of the simulation.

Figure 3 depicts a time-resolved stack of the computation, where the lowermost bar is the currently running function. It can be seen, that the most time is spent in the mesh loopers (light blue), which are all called by the residual calculation (turquoise), which is in turn called four times by the Runge-Kutta scheme.

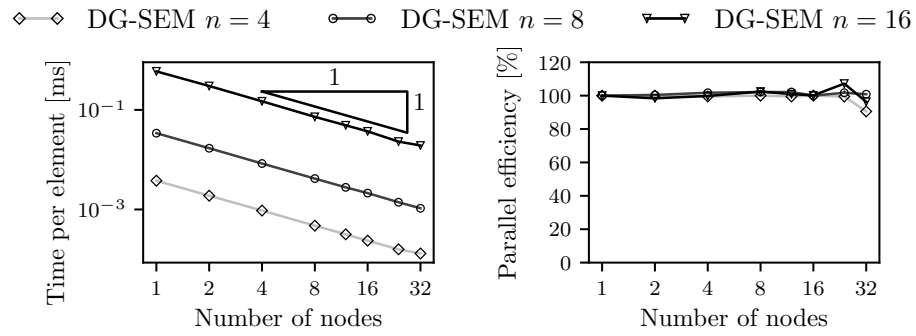

Fig. 2. Left: Runtime per element and right: parallel efficiency for different numbers of nodes for the Taylor-Green vortex, with roughly the same amount of data per node.

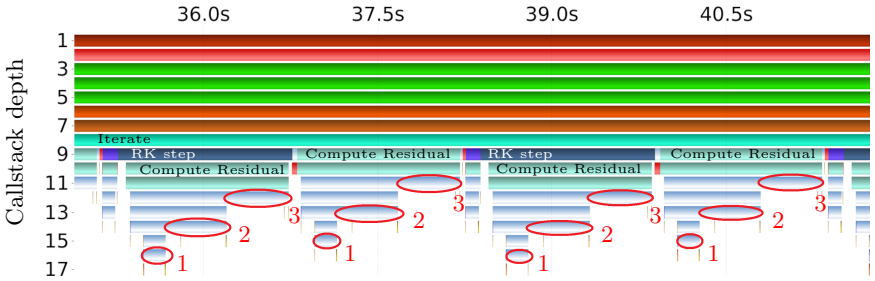

Fig. 3. Trace of one timestep, showing the callstack. The red ellipses highlight the critical kernels (1: Lifting Operator, 2: Convection and Diffusion face flux, 3: Chandrashekar [7] element contribution), which are called by the residual calculation and loop over the mesh.

Table 1 summarizes the aggregated runtime data of the most relevant kernels. Three of the mesh loopers stand out in particular: the convection and diffusion face flux functor, the element contribution functor and the lifting operator. These three plus a few smaller functor loopers make up the residual evaluation, which accounts for more than 87 % of the total runtime.

3.3 Analysis of critical kernels

With the relevant parts of the code identified, their behavior for different DG-SEM orders and how these kernels scale with it is the focus of this next section. The next step then is the analysis of node level performance of the residual calculation to grasp how well the implementation utilizes the capabilities of the hardware.

To have an estimation of how the kernels should behave, we took a look at the loops and operations performed in the residual calculation. It gets triggered four times during one time step, once for each Runge-Kutta stage. Inside, one loop over the faces and two loops over the elements of the mesh. All the functors

Table 1. The most relevant function calls with their respective percentage of total runtime on a cubic domain with a width of 64 elements with DG-SEM of order $n = 4$.

Kernel	runtime	percentage
Residual evaluation	246.6	87.7
· Convection and Diffusion face flux	81.7	29.1
· Chandrashekar element contribution [7]	74.1	26.4
· Lifting Operator	65.6	23.3
· Zero Setter Functor (gradient)	18.1	6.4
· Inverse mass matrix	7.1	2.5
Explicit Runge Kutta	18.6	6.6
Other	16.0	5.7
Total	281.2	100

of these loops use the BR2 scheme with a lifting operator when accessing the state of the simulation. The lifting operator is triggered when the state gradient is accessed for the first time by any operator. The lifting operator itself sets the gradient inside the element to zero (order n^3), then computes a face integral contribution to the gradients for each point on the element faces, which is of order n^2. Lastly it applies the inverse mass matrix on the gradients for the element, again order n^3.

The face loop of the residual calculation, which computes the convection and diffusion flux, is called first and triggers the lifting operator of order n^3. For the flux computation on the faces (order n^2), the gradient is needed for each point, which is of order n per point, resulting in order n^3 for the faces.

Of the two element loops, the first called functor calculates the volume term contributions to the residual. First the entropy conservative and kinetic energy preserving numerical convection flux is calculated using the scheme of Chandrashekar [7], which is of order n per point, leading to order n^4 for the numerical flux. Then the convection and diffusion flux is evaluated for each point inside the element (order n^3), which triggers the gradient computation for each point, exactly as in the face loop, resulting in order n^4. Lastly the source terms get evaluated in this element functor, but as there are none in the Taylor-Green vortex test case, we leave this one out.

The last step for the residual calculation is the application of the inverse mass matrix on the residual on all points inside an element, simply of order n^3.

Figure 4 shows the scaling behavior w.r.t DG-SEM order of the critical kernels. Due to the fact that the lifting operator is triggered by the convection and diffusion functor, they are measured together here. As can be seen in the runtime per element plot, the convection and diffusion face flux plus lifting operator scales nearly with order n^3 and the element contribution scales almost perfectly with order n^4, just as expected. Also, the whole residual calculation gets dominated by the Chandrashekar contribution for higher orders, which is apparent in both the time per element and throughput diagram.

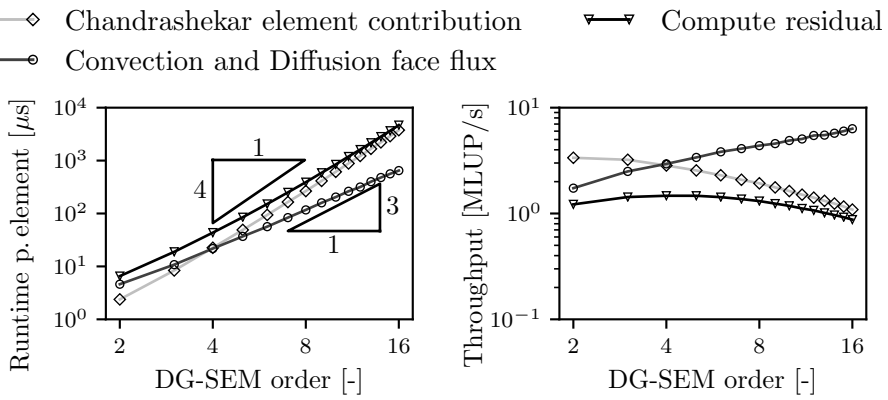

Fig. 4. Left: Runtime per element and right: throughput as computed degrees of freedom per second in Mega Lattice Updates per Second (MLUP/s) for the different operators.

3.4 Roofline model

As the parts of the residual evaluation scale as expected, we analyze how well the hardware is actually utilized. The roofline model is a well established analysis tool for node level performance. It uses the concept of arithmetic intensity

$$I = \frac{\text{flops performed}}{\text{bytes loaded or stored}}, \tag{1}$$

which is a measure of the code balance between computation and memory transfer.

Table 2 shows the ideal number of flops performed and bytes transferred through the memory interface of the CPU for the three most relevant kernels. To obtain these values we made a few assumptions: All constant values and local buffer data structures reside in the caches, special functions (i.e. log, sqrt) are counted as a single flop and index calculations are ignored. This theoretical code balance of the implementation proves unattainable in practice, but provides an upper limit, where it could be, when no unnecessary data is loaded or stored.

Table 2. Flops performed and bytes loaded/stored by each kernel.

	Flops			Bytes	
Scaling order	n^2	n^3	n^4	n^2	n^3
Convection and Diffusion face flux	853	12	-	640	-
Lifting Operator	24	5	-	88	80
Chandrashekar Element contribution	-	459	414	-	320

The roofline itself is then defined as

$$P_{\text{Roofline}} = \min(\beta * I, P_{\text{Peak}}), \tag{2}$$

where β is the memory bandwidth of the CPU and P_{Peak} the peak performance of the CPU. The peak performance is given by

$$P_{\text{Peak}} = \text{clock} \cdot \text{n}_{\text{AVX}} \cdot \text{n}_{\text{FMA}} \cdot \text{n}_{\text{ports}} \cdot \text{n}_{\text{cores}}. \tag{3}$$

The AMD EPYC 7702 CPU is running at a clock speed of 1.8 GHz, has AVX registers that fit 4 double precision float values, has two execution ports that can perform FMA instructions (multiply and addition in one cycle) and CARO has 128 cores per node. This results in a peak performance of 3.69 TFlop/s per node and 0.92 TFlop/s without AVX vectorization. The memory bandwidth was determined with a sum stream benchmark to be 360 GB/s.

To measure the achieved flops and memory throughput, we use LIKWID [13], a lightweight tool for accessing hardware counters on CPUs during runtime of an application. The AMD Zen2 architecture provides counters for flops and for cachelines transferred through the memory interface, which together with the runtime provides measured data for arithmetic intensity and computation performance.

Figure 5 depicts the roofline analysis, once with the measured and once with the computed arithmetic intensity (Table 2). On the left, in the classic roofline plots, it can be seen that the measured intensity is much lower than the computed one. On the right are the percentages of the roofline performance that the kernels reach. Assuming our computed arithmetic intensity, the element contribution functor, which is the more relevant one for higher orders, achieves less than 10 % for orders $n > 4$, with order $n = 16$ reaching 9.0 %. Even for the measured intensity of the current implementation it is only around 25 % of the hardware limit.

The most conspicuous thing to note is the unexpected behavior of the element contribution kernel: Its arithmetic intensity should be higher, which suggests unnecessary data transfer happening. Also, the low performance hints at a low utilization of vectorized instructions, especially towards higher orders, where it should be compute bound.

Lastly, to get a different angle of the element contribution kernel, its generated assembly code was analyzed. Almost no vectorized instructions were present and many functions were not inlined by the compiler, even with the compile flags allowing more aggressive inlining. Instead of AVX instructions, the assembly code shows many mov and push instructions, i.e. moving and handling of data. In general it seems the compiler is not able to optimize much from the very complex DG-SEM implementation in CODA, at least for the configuration chosen here.

4 Optimization potential

The above performance analysis leads to a few possible optimizations that can be considered down the line.

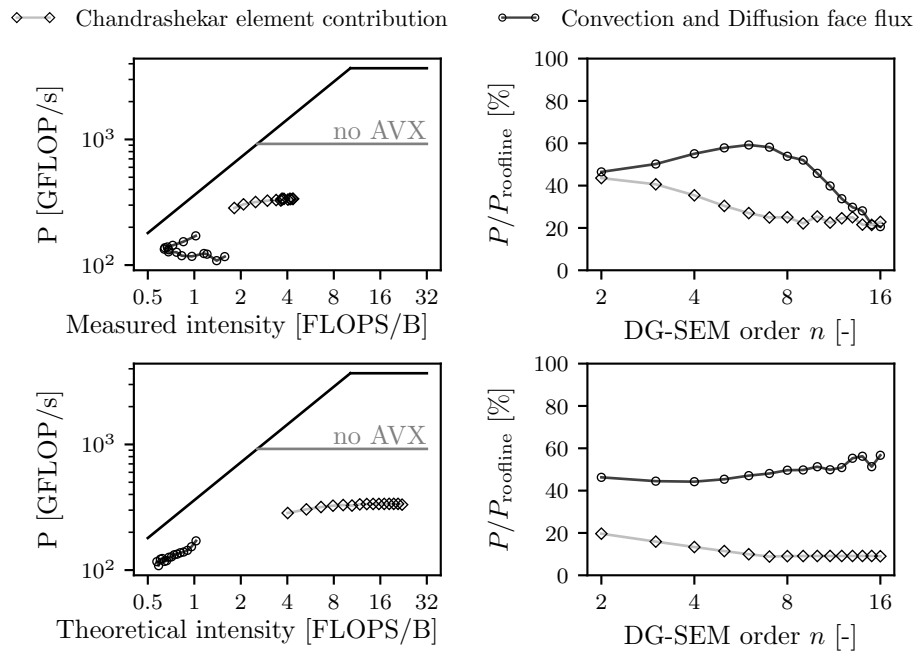

Fig. 5. Top left: Measured Roofline for both critical kernels, top right: percentage of roofline performance reached over the DG orders, bottom left: Roofline of the critical kernels with their respective performance at the theoretically optimal intensity (Table 2) and bottom right: percentage of roofline performance reached for optimal intensity.

First and foremost, the low utilization of vectorized instructions and function inlining is the most pressing concern. As long as the compiler is not capable of properly interpreting and optimizing the code, most optimizations will prove fruitless. The first focus here should be the Chandrashekar computation of the numerical flux, with its n^4 complexity. It is purely dependent on the element itself and does only require local restructuring of the data access patterns to enable the compiler to vectorize the code.

Secondly, the gradient gets computed multiple times, once for each functor requiring it. Also the convection and diffusion face flux kernel would be of order n^2 instead of n^3 without the gradient computation inside. With some reordering and the gradient being saved after being calculated when needed for the first time this can be implemented more efficiently.

Lastly, the lifting operator looping once over the mesh, setting the gradient to zero is unnecessary when the implementation of the actual computation is changed accordingly.

A long term consideration would be to introduce a different memory layout, with better cache locality and clearer access patterns for the compiler to vectorize with more ease. Figure 6 depicts the current memory layout and a reordered

variant. Placing the states of four points of four different elements, all with the same index and the same physical variable, next to each other can bring the compiler to calculate 4 elements in parallel during one loop, each instruction working on 4 points at the same time.

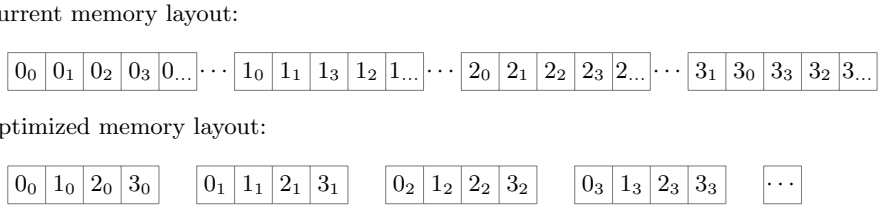

Fig. 6. Top: current memory layout with data of all integration points of an element locally grouped in memory and bottom: optimized memory layout with integration points of the same index of four different elements grouped together for better vectorization.

A second step can be to evaluate how the number of abstraction layers in the heavily templated code could be reduced. This can lead to more inlining, possibly also more vectorization and other optimization methods the compiler can utilize.

5 Conclusion

The performed benchmarks of CODA using the turbulent Taylor-Green vortex and a more granular kernel benchmark show, that the current DG-SEM implementation scales well, but the node level performance leaves much to be improved upon. Through the performance analysis using the roofline model we found some inefficiencies in the gradient computation and the lifting operator that can be removed. The assembly code additionally revealed optimization potential in more inlining and more vectorization. The next step consists of assessing the possible optimization strategies and choose the desired approach.

Acknowledgements This work is supported by Fabio Naddei (ONERA) and his work on DG-SEM methods in CODA.

References

1. Grabe, C., Probst, A., Horchler, T., Shishkin, A., Werner, M.: Potential, perspectives and HPC-requirements of scale-resolving simulation techniques. HPCN, DLR-AS (2022).
2. Slotnick, J. P., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E., Mavriplis, D. J.: CFD Vision 2030 study: A path to revolutionary computational aerosciences. Technical report NASA/CR-2014-218178, (2014).

3. Krais, N., Beck, A., Bolemann, T., Frank, H., Flad, D., Gassner, G., Hindenlang, F., Hoffmann, M., Kuhn, T., Sonntag, M., Munz, C.-D.: FLEXI: A high order discontinuous Galerkin framework for hyperbolic–parabolic conservation laws. Computers & Mathematics with Applications 81, 186-219 (2021).
4. Marmignon, C., Naddei, F., Renac, F.: Energy relaxation approximation for compressible multicomponent flows in thermal nonequilibrum. Numerische Mathematik 151, 151-184 (2022).
5. Cantwell, C. D., Sherwin, S. J., Kirby, R. M., Kelly, P. H. J.: From h to p efficiently: Strategy selection for operator evaluation on hexahedral and tetrahedral elements. Computers & Fluids 43(1), 23-28 (2011).
6. Beck, A. D., Gassner, G. J., Bolemann, T., Frank, H., Hindenlang, F., Munz, C.-D.: Underresolved turbulence simulations with stabilized high order discontinuous Galerkin methods. Direct and Large-Eddy Simulation IX. ERCOFTAC Series, vol 20, 103-108 (2015).
7. Chandrashekar, P.: Kinetic energy preserving and entropy stable finite volume schemes for compressible Euler and Navier-Stokes equations. Communications in Computational Physics 14, vol 5, 1252-1286 (2013).
8. Huismann, I., Stiller, J., Fröhlich, J.: Efficient high-order spectral element discretizations for building block operators of CFD. Computers & Fluids 197, 104386 (2020).
9. Huismann, I., Fechter, S., Leicht, T.: HyperCODA – Extension of Flow Solver CODA to Hypersonic Flows. STAB/DGLR Symposium, Springer, Cham (2020).
10. Gassner, G. J., Beck, A. D.: On the accuracy of high-order discretizations for underresolved turbulence simulations. Theoretical and Computational Fluid Dynamics 27(3), 221–237 (2013).
11. Fehn, N., Wall, W. A., Kronbichler, M.: Efficiency of high-performance discontinuous Galerkin spectral element methods for under-resolved turbulent incompressible flows. International Journal for Numerical Methods in Fluids 88(1), 32-54 (2018).
12. Knüpfer, A., Rössel, C., an Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., Nagel, W. E., Oleynik, Y., Philippen, P., Saviankou, P., Schmidl, D., Shende, S., Tschüter, R., Wagner, M., Wesarg, B., Wolf, F.: Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. Tools for High Performance Computing 2011, 79-91, Springer, Berlin, Heidelberg (2012).
13. Treibig, J., Hager, G., Wellein, G.: LIKWID: Lightweight performance tools. Competence in High Performance Computing 2010, 165-175 (2011).
14. 1st International Workshop on High-Order CFD Methods. https://cfd.ku.edu/hiocfd.html
15. Bassi, F., Rebay, S., Mariotti, G., Pedinotti, S., Savini, M.: A high-order accurate discontinuous finite element method for inviscid and viscous turbomachinery flows. In Proceedings of the 2nd European Conference on Turbomachinery Fluid Dynamics and Thermodynamics, 99-109 (1997).
16. Bassi, F., Grivellini, A., Rebay, S., Savini, M.: Discontinuous Galerkin solution of the Reynolds-averaged Navier–Stokes and k-ω turbulence model equations. Computers & Fluids 34(4-5), 507-540 (2005).
17. Renac, F.: Entropy stable, robust and high-order DGSEM for the compressible multicomponent Euler equations. Journal of Computational Physics 445, 110584 (2021).
18. Marmignon, C., Naddei, F., Renac, F.: Energy relaxation approximation for compressible multicomponent flows in thermal nonequilibrium. Numerische Mathematik, 151(1), 151-184 (2022).