Dealing with Non-linear Terms in a Modal High-Order Discontinuous Galerkin Method

Nikhil Anand, Harald Klimach and Sabine Roller

Abstract The Discontinuous Galerkin (DG) method utilizes a mesh of elements with local functions like traditional continuous finite element methods, together with a flux approximation between elements like finite volume methods. This combination yields a high locality of the overall scheme, especially for high-order representations within elements. Two local operations need to be mainly considered. One is the application of the mass matrix and the other is the stiffness matrix. With an appropriate orthogonal basis as choice for the local functions both operations can be computed with minimal complexity. In this contribution we are concerned with a DG implementation that makes use of a Legendre polynomial basis with an application to non-linear equation systems. For non-linear systems a complication is introduced by the scheme by the necessity to compute the non-linear flux operation, which generally can not be done in the optimal modal basis. Instead, a pointwise evaluation of the non-linear operations is usually performed. Combining the fast evaluation of the integrals in the modal scheme with the pointwise evaluation of the non-linear terms requires a transformation between these two. Many methods have been developed for a fast transformation from Legendre modes to nodal values [1]. However, most of those methods for fast polynomial transformations are designed for extremely high polynomial degrees in the range of several hundreds. In three-dimensional DG simulations the polynomial degree in each dimension is more limited, and we are looking for methods that are fast but suitable for polynomials in the range up to a maximal degree of one hundred. We discuss some approaches to the fast transformation, especially the method proposed by Alpert and Rokhlin [2], and compare our implementation of this method to a straight forward L_2 projection. The implementation specifically addresses also the hybrid parallelism with MPI and OpenMP for the three-dimensional DG elements.

N. Anand (✉) · H. Klimach · S. Roller
University of Siegen, 57076 Siegen, Germany
e-mail: nikhil.anand@uni-siegen.de

H. Klimach
e-mail: harald.klimach@uni-siegen.de

S. Roller
e-mail: sabine.roller@uni-siegen.de

M.M. Resch et al. (eds.), *Sustained Simulation Performance 2016*,
DOI 10.1007/978-3-319-46735-1_5

1 Introduction

High-order methods for CFD applications have gained popularity in the research community in the last decades. They have the potential to provide not only high accuracy but also efficient numerical solutions to the problems when compared to lower order methods, as the approximation error decreases exponentially with the order for smooth solutions. However, classical spectral methods suffer from limitations to simple periodic domains and a global support. Their deployment on parallel distributed systems for complex setups is usually limited in scalability.

The Discontinuous Galerkin (DG) method builds a class of schemes, that enable high-order discretizations of conservation laws. It shares many of the advantages of high-order spectral methods but overcomes its limitation of global ansatz functions by weakly coupled, element local functions. DG methods, due to this locality, provide a path towards massive parallelism with high-order on distributed systems. Within elements shared memory parallelism can be employed for the local operations, which allows the numerical scheme to match the typical hierarchy of modern computing systems.

Using an appropriate basis for the element local functions, linear operations can be efficiently computed with optimal computational complexity. For example, Legendre polynomials with their orthogonal basis and recursive definition results in a trivially invertable diagonal mass matrix and allows for a stiffness matrix that can be applied with optimal computational effort $\mathcal{O}(p^3)$ for three dimensional elements and a scheme order of p. Although an appropriate nodal basis also allows for an efficient computation of the mass matrix, the same can not be achieved for the stiffness matrix at the same time. With some restrictions the computational cost can be limited to $\mathcal{O}(p^4)$ operations in this case.

We also make use of cubical elements, which offer optimal properties for the DG scheme. The elements are organized in an Octree that enables together with a space-filling curve ordering an efficient partitioning and neighbor identification on distributed parallel systems. By restricting to hexahedral meshes in combination with orthogonal basis functions, we optimize the tensor-product nature in multiple dimensions. This enables us to use a dim-by-dim approach with minimal computational effort.

Though, linear equations can be efficiently computed with no added complexity when the appropriate basis is used, non-linear equations can generally not be treated so easily anymore. For example for nonlinear operations occurring in initial conditions, boundary conditions, source terms or non-linear fluxes, a transformation of the Legendre modes to pointwise representation needs to be performed. This forces us to look for algorithms that offer fast transformations. To allow an in-place transformation, we use the same number of points and modal coefficients. The naive evaluation of the polynomials at each of these points would result in $\mathcal{O}(p^6)$ operations in three dimensions, which clearly is not an option for high-order approximations. By employing the dim-by-dim method, the cost for this can be reduced to $\mathcal{O}(p^4)$. However, there are fast algorithms that achieve the transformation in $\mathcal{O}(p^3 log(p))$

operations theoretically. Unfortunately, many methods for fast polynomial transformations are designed for extremely high degrees and often do not exhibit their asymptotically fast behavior for low polynomial degrees like the ones used in three dimensional DG simulations. In the next paragraph, we briefly discuss some of these algorithms which we implemented in our highly parallel framework.

The Fast Polynomial Transformation (FPT) described by Alpert and Rhoklin is based on a fast approximative transformation of Legendre modes to Chebyshev polynomials [2], followed by a Fast Fourier Transformation. Another method involves the use of a Fast Multipole Method for a direct transformation of Legendre polynomials to Legendre nodes, developed by Suda [3]. This involves one algorithmic step less than the FPT. Both these method offer $\mathcal{O}(p^3 log(p))$ complexity for the transformation.

This paper is organized as follows: First, we briefly review the Discontinuous Galerkin discretization in Sect. 2. Then we highlight the choice of basis for ansatz and test function definition in Sect. 3. After that we introduce the projection algorithms and the underlying ideas in Sect. 4, followed by some strategies for hybrid parallelism using MPI and OpenMP in Sect. 5. Finally Sect. 6 presents the comparison and analysis of different projection algorithms used in this work.

2 The High Order Discontinuous Galerkin Method

In this section, we briefly introduce the semi-discrete form of the Discontinuous Galerkin Finite Element Method (DG) for compressible inviscid flows. The compressible Euler equations are the non-linear system of equations describing such flows with the conservation of mass, momentum and energy given by

$$\partial_t \mathbf{u} + \nabla \cdot \mathbf{F}(\mathbf{u}) = 0, \tag{1}$$

equipped with suitable initial and boundary conditions. Here $\mathbf{u}$ is a vector of conservative variables and the flux function $\mathbf{F}(\mathbf{u}) = (\mathbf{f}(\mathbf{u}), \mathbf{g}(\mathbf{u}))^T$ for two spatial dimensions is given by

$$\mathbf{u} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix}, \qquad \mathbf{f}(\mathbf{u}) = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (\rho E + p)u \end{bmatrix}, \qquad \mathbf{g}(\mathbf{u}) = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (\rho E + p)v \end{bmatrix},$$

where ρ, $\mathbf{v} = (u, v)^T$, E, p denotes the density, velocity vector, specific total energy and pressure respectively. The system is closed by the quation of state assuming the fluid obeys the ideal gas law with pressure defined as $p = (\gamma - 1)\rho \left(e - \frac{1}{2}(u^2 + v^2)\right)$. where $\gamma = \frac{c_p}{c_v}$ is the ratio of specific heat capacities and e is the total internal energy per unit mass.

The Discontinuous Galerkin formulation of the above equation is obtained by multiplying it with a test function ψ and integrating it over the domain Ω. Thereafter, integration by parts is used to obtain the following weak formulation

$$\int_{\Omega} \psi \frac{\partial \mathbf{u}}{\partial t} d\Omega + \oint_{\partial\Omega} \psi \mathbf{F}(\mathbf{u}) \cdot \mathbf{n} ds - \int_{\Omega} \nabla\psi \cdot \mathbf{F}(\mathbf{u}) d\Omega = 0, \qquad \forall\psi, \tag{2}$$

where ds denotes the surface integral. A discrete analogue of the above equation is obtained by considering a tessellation of the domain Ω into n closed, non-overlapping elements given by $T = \{\Omega_i | i = 1, 2, \ldots, n\}$, such that $\Omega = \cup_{i=1}^{n} \Omega_i$ and $\Omega_i \cap \Omega_j = \emptyset \forall i \neq j$. We define a finite element space consisting of discontinuous polynomial functions of degree $m \geq 0$ given by

$$P^m = \{f \in [L^2(\Omega)]^m\}. \tag{3}$$

With the above definition we can write the approximate solution $\mathbf{u}_h(\mathbf{x}, t)$ within each element using a polynomial function of degree m

$$\mathbf{u}_h(\mathbf{x}, t) = \sum_{i=1}^{m} \hat{u}_i \phi_i, \qquad \psi_h(\mathbf{x}) = \sum_{i=1}^{m} \hat{v}_i \phi_i, \tag{4}$$

where the expansion coefficients $\hat{u}_i$ and $\hat{v}_i$ denote the degrees of freedom of the approximation of solution and of test function respectively. Notice, that there is no global continuity requirement for $\mathbf{u}_h$ and ψ_h in the previous definition. Splitting the integrals in Eq. (2) into a sum of integrals over elements Ω_i, we obtain the space-discrete variational formulation

$$\sum_{i=1}^{n} \frac{\partial}{\partial t} \int_{\Omega_i} \psi_h \mathbf{u}_h d\Omega + \oint_{\partial\Omega_i} \psi_h \mathbf{F}(\mathbf{u}_h) \cdot \mathbf{n} ds - \int_{\Omega_i} \nabla\psi_h \cdot \mathbf{F}(\mathbf{u}_h) d\Omega = 0, \qquad \forall\psi_h, \tag{5}$$

Due to element local support of the numerical representation, the flux term is not uniquely defined at the element interfaces. The flux function is, therefore, replaced by a numerical flux function $\mathbf{F}^*(\mathbf{u}_h^-, \mathbf{u}_h^+, \mathbf{n})$ where $\mathbf{u}_h^-, \mathbf{u}_h^+$ are the interior and exterior traces at the element face in the direction $\mathbf{n}$ normal to the interface.

For simplicity we can re-write the equation above in matrix vector notation and obtain

$$\frac{\partial}{\partial t} \hat{\mathbf{u}} = M^{-1} \left(S \cdot \mathbf{F}(\hat{\mathbf{u}}) - M^F \cdot \mathbf{F}(\hat{\mathbf{u}}) \right) =: rhs(\hat{\mathbf{u}}). \tag{6}$$

where M,S denote the mass and the stiffness matrices and M^F are so called face mass lumping matrices. The above obtained ordinary differential equation (6) can be solved in time using any standard timestepping method, e.g. a Runge-Kutta method.

3 Choice of Basis Function

An important part of the DG formulation is the choice of the ansatz functions ϕ to represent the approximated solution $\mathbf{u}_h$ and test function ψ in (4). Typical choices are polynomials. This section briefly highlights the choice of the polynomial basis which allows for faster evaluation of integrals in (5). From theoritical point of view the choice of the polynomial basis is arbitrary, however, the computational effort required to evaluate the volume and surface integral term in (5) can differ based on this choice. For example, the mass matrix term $\int_{\Omega_i} \psi_h \mathbf{u}_h d\Omega$ can be cheaply computed when an orthogonal basis is choosen for both $\mathbf{u}_h$ and ψ_h. *Legendre* polynomials are orthogonal with the L_2 inner product in the interval -1 to 1. Unluckily, this orthogonality is lost for the stiffness matrix term $\int_{\Omega_i} \nabla\psi \cdot \mathbf{F}(\mathbf{u}_h) d\Omega$ since derivatives of Legendre polynomials do not simply reduce to another orthogonal basis. If the fluxes have non-linear dependence on the state (like the fluxes in the Euler equations), then this orthogonality is also lost for the surface integral term $\oint_{\partial\Omega_i} \psi \mathbf{F}(\mathbf{u}_h) \cdot \mathbf{n} ds$. For this reason, the *nodal* polynomial basis like *Lagrange* polynomials are quite common, as the coefficients can be directly identified as point values thereby allowing pointwise evaluation. The DG scheme emerging from using this type of basis are classified as *nodal DG*.

However, the evaluation of gradient is not cheap for Lagrange polynomials and when used naively the computational cost falls in the order of $\mathcal{O}(p^6)$. With some restrictions, it can be limited to $\mathcal{O}(p^4)$ operations in our case. However, for cubical domains a more efficient basis could be found which allows fast evaluation of both mass and stiffness matrix at the same time. For this we use Legendre polynomials, which is a special type of Jacobi polynomials, following a three term recursion. Being orthogonal it can cheaply evaluate the mass matrix and the recursion definition helps to assemble the stiffness matrix in $\mathcal{O}(p^3)$. The DG scheme based on these kind of basis functions for approximation are known as *modal DG*. Apart from this, modal DG has also other cheaper and efficient means when it comes to dealing with aliasing errors, filtering and stabilization techniques or fast projection of solution onto faces etc. We skip further details as it is not in the scope of this paper.

4 The Projection Algorithms

The Legendre polynomial series does not have a fast transform associated with it like a Chebyshev expansion. Therefore, a $p+1$-th order Legendre expansion would normally require $\mathcal{O}(p^2)$ operations to evaluate point values at p nodes. With multiple dimensions this high computational cost gets significant even for low orders, as the polynomials in all directions need to be considered, resulting in $\mathcal{O}(p^{2d})$ operations for d dimensions. With a tensor product formulation on cubical elements, a dim-by-dim algorithm can be deployed and the number of iterations reduced to $\mathcal{O}(d \cdot p^{d-1} \cdot p^2) = \mathcal{O}(p^{d+1})$. Without loss of generality, but for readability, we restrict ourselves

to a single dimension here. Next, we give a formal definition of the problem of polynomial projection and then go on describing the algorithms we use along with relevant implementation details in the subsequent subsections.

Given is a function $f : [-1, 1] \rightarrow \mathbb{R}$ expressed by an n term finite Legendre expansion of the form

$$f(t) = \sum_{i=0}^{n-1} \hat{u}_i \cdot L_i(t). \tag{7}$$

We want to convert this expansion to n nodal point values of f evaluated at the points $t_1, t_2, \ldots, t_n$. Similarly, for the inverse operation, given tabulated values of function $f : [-1, 1] \rightarrow \mathbb{R}$ at n nodes $t_1, t_2, \ldots, t_n$ we want to evaluate the coefficients $\hat{u}_0, \hat{u}_1, \ldots, \hat{u}_{n-1}$ such that

$$f(t_j) = \sum_{i=0}^{n-1} \hat{u}_i \cdot L_i(t_j) \tag{8}$$

holds. These transformations in the above noted general formulation require $\mathcal{O}(n^2)$ operations. For a polynomial representation in 3D the number of operations required for this transformation would be proportional to n^6. Thus, a naive implementation of the general transformation is prohibitively expensive even for moderately large polynomial degrees n. This creates the need to look for an efficient way to transform modal expansions to point values and back in order to retain high performance for higher order.

Alpert and Rokhlin presented a fast and stable algorithm for fast transformations of Legendre polynomials [2]. Various alternatives and extensions were proposed since then e.g [1, 3]. A detailed analysis of fast polynomial transformations can be found in [4]. Figure 1 shows some existing fast algorithms that can deliver fast transformation from modal coefficients to point values and back. The solid line in the figure higlights the transformation algorithm we implemented tailored to fit our highly parallel framework. In the following subsections we briefly discuss the projection algorithms and its implementation details. Then we compare the algorithms and analyse them.

4.1 Direct L_2 Projection

The direct, but expensive transformation between the nodal and modal basis is the L_2 projection, which refer to as L2P in this paper. Mathematically, the L_2 projection f_h, for Legendre expansion f of the form (7) projected onto any arbitrary function space $\theta \in L_2(\Omega)$ is given by

$$\langle f_h - f, v \rangle_{L_2(\Omega)} = 0 \qquad \forall v \in \theta, \tag{9}$$

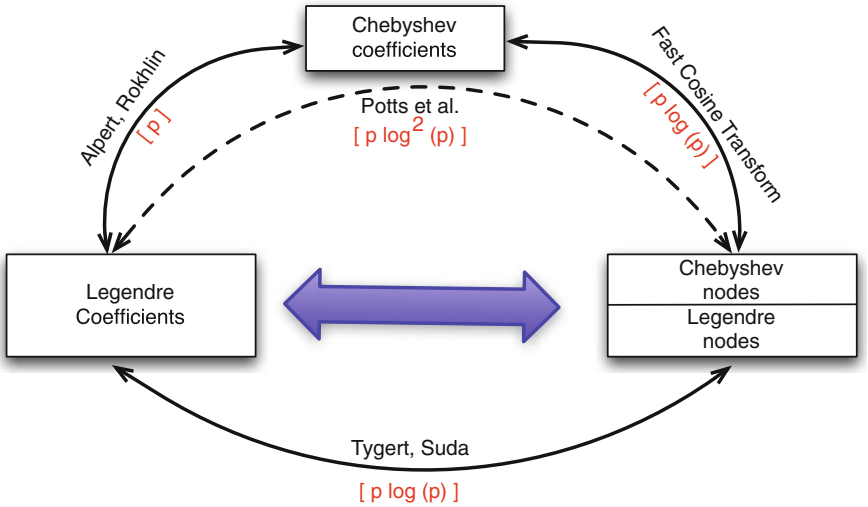


Fig. 1 This figure shows some of the available fast algorithms for converting Legengre coefficients to point values along with their expected computational complexity for transformation in single dimension marked in *red*. *Solid arrows* highlight the algorithms we used in this work

or equivalently

$$\langle f_h, v\rangle_{L_2(\Omega)} = \langle f, v\rangle_{L_2(\Omega)} \qquad \forall v \in \theta. \tag{10}$$

Then, for each element the discrete counterpart of this system can be stated as

$$Mx = b, \tag{11}$$

where components of matrix $M_{i,j} = \langle v_j, v_i\rangle$, $\{v_i\}$ being basis of space θ, x is the vector containing degrees of freedom of $f_h = \sum_i x_i v_i$ and components of b_i is given by $b_i = \langle f, v\rangle$.

By choosing the Lagrange polynomials as target space, this approach be be used to transform the represenation from Legendre modes to Legendre nodes and back. For an accurate mapping and to allow in-place transformation, we use as many points as modal coefficients. As can be easily seen from the matrix formulation the costs of this transformation grows quadratically with the number of degrees of freedoms. Furthermore, the costs increase also with the dimensionality of the polynomial. For example, a three dimensional p-th order element has p^3 coefficients for each variable. Evaluating it at all the p^3 nodes would take p^6 operations, which would quickly get prohibitive expensive for higher orders. However, tensor product basis functions and cubical elements help us to reduce the problem formulation to multiple one dimensional operations, which in general is not possible for non tensor-product elements like tetrahedral elements.

Considering directions $k \in \{1, 2, 3\}$ for 3 dimensions and order p representation in an element Ω_i we can define the spatial ansatz function as a product of ansatz function in one dimension by

$$\begin{aligned} \phi_l(\mathbf{x}) = \phi_l(x_1, x_2, x_3) &= \phi_{l_1}(x_1) \cdot \phi_{l_2}(x_2) \cdot \phi_{l_3}(x_3) \\ l &= 1 + l_1 + l_2 \cdot p + l_3 \cdot p^2 \\ 1 &\leq l_1, l_2, l_3 \leq p^3 \end{aligned}$$

Using this, the terms of the form $\langle \phi_a(\mathbf{x}), \phi_b(\mathbf{x}) \rangle$ can be broken down into

$$\langle \phi_a(\mathbf{x}), \phi_b(\mathbf{x}) \rangle = \prod_{k=1}^{3} \langle \phi_{a_k}(x_k), \phi_{b_k}(x_k) \rangle \,,$$

From computational point of view, this makes the transformation applicable for dimension by dimension. Exploiting this we are able to reduce the complexity of the L_2 projection down to $\mathcal{O}(p^4)$ in three dimension.

4.2 Fast Polynomial Transformation

The FPT algorithm by Alpert and Rokhlin [2] is based on the idea to exploit the already known fast transformation for Chebyshev polynomials in the form of the fast cosine transform. As we are looking for the transformation of a Legendre expansion (7), the missing component is a fast transformation between Legendre and Chebyshev coefficients. In this section we describe the basic concept and implementation for this fast polynomial transformation, which we refer to as FPT.

Assuming, a function can be described by a finite Legendre expansion as given in (7) it then can also be described by a finite Chebyshev expansion of the form

$$f(x) = \sum_{i=0}^{n-1} \hat{u}_i^c \cdot T_i(x) \tag{12}$$

where $T_i(x)$ is the i-th Chebyshev polynomial. The coefficients $\hat{u}_i^l$ and $\hat{u}_i^c$ are then related by the equation

$$\hat{u}^c = M \cdot \hat{u}^l \tag{13}$$

where, $\hat{u}^c = (\hat{u}_0^c, \hat{u}_1^c, \ldots, \hat{u}_{n-1}^c)$ and $\hat{u}^l = (\hat{u}_0^l, \hat{u}_1^l, \ldots, \hat{u}_{n-1}^l)$. Also, conversely, if f is a function defined as the Chebyshev expansion in (12) then it can be expressed as the Legendre expansion of the form (7), with the coefficients $\hat{u}^l$ given by

$$\hat{u}^l = L \cdot \hat{u}^l \tag{14}$$

Alpert and Rokhlin showed that the entries of matrices M and L can be expressed by meromorphic functions of the matrix indices and have the following structure:

$$M_{i,j} = \begin{cases} \frac{1}{\pi}\Lambda\left(j/2\right) & \text{if } 0 = i \leq j < g+1 \text{ and } j \text{ is even} \\ \frac{2}{\pi}\Lambda\left(\frac{j-i}{2}\right)\Lambda\left(\frac{j+i}{2}\right) & \text{if } 0 < i \leq j < g+1 \text{ and } j+i \text{ is even} \\ 0 & \text{otherwise} \end{cases} \tag{15a}$$

$$L_{i,j} = \begin{cases} 1 & \text{if } i = j = 0 \\ \sqrt{\pi}/\left(2\Lambda(i)\right) & \text{if } 0 < i = j < g+1 \\ \frac{-j(i+1/2)}{(j+i+1)(j-i)}\Lambda\left(\frac{j-i-2}{2}\right)\Lambda\left(\frac{j+i-1}{2}\right) & \text{if } 0 \leq i < j < g+1 \text{ and } i+j \text{ is even} \\ 0 & \text{otherwise} \end{cases} \tag{15b}$$

The function $\Lambda : \mathbf{C} \to \mathbf{C}$ is defined by $\Lambda(z) = \Gamma(z+1/2)/\Gamma(z+1)$. Therefore, blocks in the matrices can be approximated cheaply by e.g. a Taylor series expansion using only a few coefficients. The farther away the blocks from the diagonal, the less accurate their approximation needs to be. Thus, approximation costs can drop with distances to the diagonal. For further details and analysis we refer to [2] The matrices M and L are subdivided into entries close to the diagonal, triangle submatrices and blocks. Diagonals and triangles are evaluated directly, while for the blocks only an approximation is used. The block-size grows with the distance from the diagonal. A sketch of this decomposition of the matrices is given in Fig. 2.

Due to the fact that the number of rows scales proportional to p and the number of blocks scale as $\mathcal{O}(log(p))$, the computational complexity of the algorithm is $\mathcal{O}(p\ log(p))$ as a constant effort for all blocks is used no matter their size.

Also, this algorithm is more stable than the direct L_2 projection as the scheme does not give rounding errors when the points are located close to the boundary of the reference element.

4.3 Spherical Harmonic Transform Using Fast Multipole Method

Spherical harmonics are set of spatial functions forming orthogonal system defined on the surface of a sphere. Several transformation exist for performing spherical harmonic transforms [5–7]. Suda proposes a fast transformation algorithm for the same using the Fast Multipole Method (FMM) [7, 8]. In this section we describe the basic idea of algorithm proposed by Suda.

Spherical harmonic can be expressed as product of an associated Legengre function and a trigonometric function. This way, the spherical harmonic transform breaks down into successive evaluation of an associated Legendre transform and an inverse

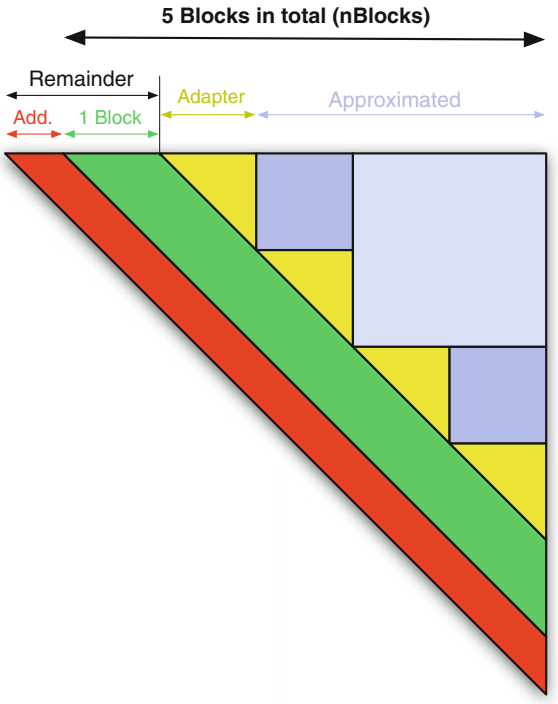


Fig. 2 Subdivision of matrices M and L. Entries close to the diagonal are directly applied while the blocks further away are approximated using a Taylor series expansion. The *triangles* are also computed directly, just like the diagonals

fourier transform. Fast algorithm for the latter already exists, so it is sufficient to accelerate the associated Legendre transform for fast spherical harmonic transform. Suda [7] uses polynomial interpolation to evaluate associated Legendre transform and accelerates the polynomial interpolation using Fast Multipole Method (FMM). For detailed algorithm we refer to [7].

A spherical Harmonic function $Y_j^k(\lambda, \mu)$ can be represented as a product of an associated Legendre function and a trigonometric function as

$$Y_j^k(\lambda, \mu) = L_j^k(\mu) e^{ik\lambda} \tag{16}$$

where μ and λ are spherical angular coordinates. L_j^k is the associated Legendre function of degree j and order k. The evaluation of the spherical harmonic expression

$$g(\lambda, \mu) = \sum_{k=1}^{p+1} \sum_{j=k}^{p+1} g_j^k Y_j^k(\lambda, \mu) \tag{17}$$

can also be split into an associated Legendre function transform and a Fourier transform as

$$g^k(\mu) = \sum_{j=k}^{p+1} g_j^k L_j^k(\mu), \tag{18}$$

$$g(\lambda, \mu) = \sum_{k=1}^{p+1} g^k(\mu) e^{ik\lambda} \tag{19}$$

The inverse spherical harmonic transform involves the computation of $g(\lambda_r, \mu_s)$ from the input coefficients g_j^k, while the spherical harmonic transform is the computation of g_j^k from sampled values from $g(\lambda_r, \mu_s)$. For the so-called alias free condition, the indices are restricted in the following way

$$\{r \in (1, R), s \in (1, S); R \geq 3p + 1, S \geq \frac{3p + 1}{2}\}. \tag{20}$$

Thus the inverse transform consists of associated Legendre function transform (18) and Fourier transform (19). The Fast Fourier Transform already enables (19) to be computed optimally in $\mathcal{O}(p\ log(p))$. Thus accelerating the associated Legendre function transform is sufficient enough to reduce the complexity of the whole transformation. Suda proposes an algorithm based on this idea, and the computational complexity of the whole transformation is $\mathcal{O}(p\ log(p))$ for 1D. There is also a set of routines publicly available as a C library to perform these transformations [9]. We integrated the FXTPACK routines into our program and use it to perform transformations.

5 Hybrid Parallelization of the Projection Algorithms

As we discussed earlier, a high locality coming from loosely coupled elements in DG is key to good scalability on distributed systems. Also, the workload per element gets high with increasing order. For example, for p-th order scheme there are p^3 unknows per variable per element. However, this workload can not be distributed among different processes efficiently, as access pattern within an element is quite random and tightly coupled. However, with shared memory parallelization of operations within each element the scalability can be increased, especially for high order discretizations. So, the elements can be distributed among the processes and within elements shared memory parallelism can be deployed. Using this technique, its possible to scale down to one element per node. We used OpenMP to parallelise the L2P projection algorithm. We present and discuss the results for hybrid parallelization in Sect. 6.

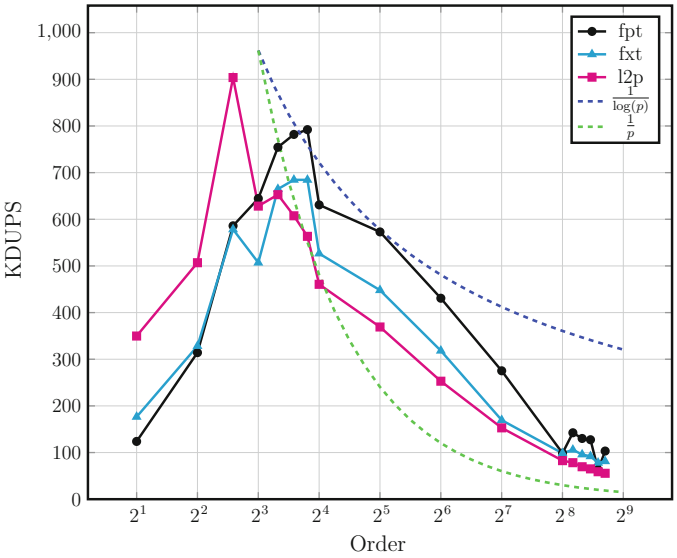
6 Comparison of Different Algorithms

In this section we discuss the comparison of different transformation algorithms, highlighting the expected and achieved performances. Then we go on presenting some results on the hybrid parallelization as discussed in Sect. 5. Finally, we briefly analyse the performance behavior with respect to vectorization and OpenMP.

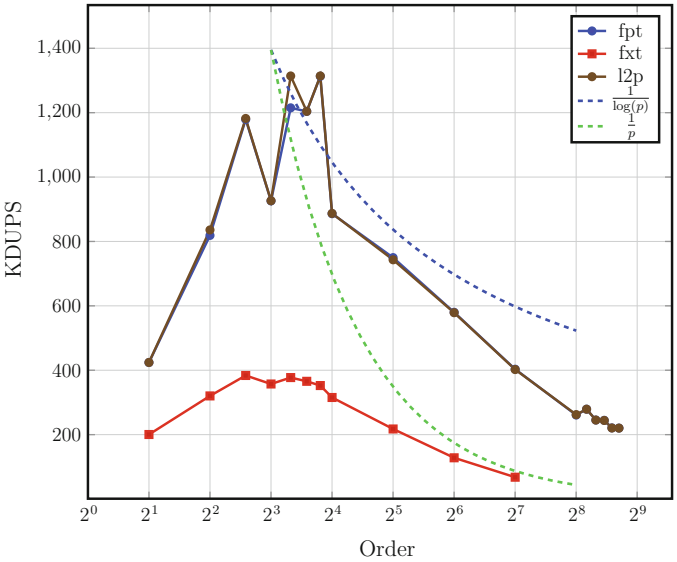
As a testcase we use the compressible Euler equation in 3D to simulate the flow of a fluid in a simple cubic domain with periodic boundaries. For time stepping, we use 4-step explicit Runge-Kutta (RK4) method. Because of the non-linear flux term the computation requires the conversion from modal to nodal coefficients and back in the otherwise modal scheme. With explicit RK4 time stepping, it needs to do this transformation 4 times for each single time step iteration and for each conservative variable (a total of 5 conservative variables).

To measure the performance, we consider the whole computation loop. There are, of course, several other operations apart from polynomial transformations contributing to the overall performance. However, for sufficiently high orders the transformations are the most significant factor. And also for the overall behavior, which is in the end the relevant measure, the impact of the transformation performance can be observed. With all other simulation parameters remaining the same, we believe the overall performance is a valid indication for a comparison between the different transformation algorithms in the actual simulation setup. We use the measure of thousand degree of freedom updates per second (KDUPS) for the performance assessment. A degree of freedom update refers to the time taken to update a single degree of freedom from one timestep to the next. Larger KDUPS imply better performance and vice versa. Also notice that the performance attained includes effects of the implementation and the computing hardware, such as caching or vectorization.

In Fig. 3 we plot the performance of the different transformation algorithms. First in Fig. 3a we measure the performance on our small development platform, which uses intel Xeon X5650 (Westmere) processors. The performance index KDUPS is plotted against the increasing order on the horizontal axis. The overall problem size is kept nearly constant around 80 million degrees of freedom per variable. Thus, with an increasing order the mesh resolution gets coarser to maintain the overall problem size (or total number of degrees of freedom). The peak in the low order range shows a caching effect, where a single elements can be kept completely in the cache. There, the computation is faster as it benefits of the data locality for all operations inside the element. For higher orders this effect gets lost as data needs to be fetched from memory even for element local operations. The performance flattens out. On this machine we observe that that after the 8th order the performance of FPT is better when compared to others, even though the asymptotic fast regime seems to be achieved only for very high orders. At an order of 256 we observe a small dip in the performance of the FPT, but apart from that the FPT always appears to be the fastest option. Running exactly the same setup on an Intel Xeon E5-2680v3 (Haswell) processor, we observe a different behavior. While the performance for all transformation methods improves due to the faster processor, we can also observe a speed-up for L2P, which becomes



(a) Comparison of projetion algorithms on intel Xeon X5650 (Westmere) processor



(b) Comparison of projetion algorithms on Intel Xeon E5-2680v3 (Haswell) processor

Fig. 3 Performance measure of the transformation algorithms. L2P denotes the direct L2 projection, FPT is the fast polynomial transformation and FXT is the spherical harmonic transform of the FXTPACK library

comparable to the performance of the FPT. This benefit is likely due to the well vectorizable parts of the L2P algorithm, which is of increased importance on the newer processor. At the same time, the FXT implementation can not profit from the improved performance on the newer processor, which is probably due to the irregular memory access patterns in the FMM.

We observe the direct L_2 projection with the dim-by-dim optimization yields a peformance equivalent to the FPT up to scheme orders as high as 400. Though, the FPT should asymptotically provide a computational complexity of $p^3\ logp$ and the L2P of p^4. These operation estimations correspond to a line following $\frac{1}{logp}$ and $\frac{1}{p}$ respectively. The expected asymptotic behaviors are included in the figures for reference. As we can see, it is hard for the fast algorithms to compete with the simple direct transformation, which just inflicts a matrix-vector product that can be computed very efficiently.

As we mentioned in Sect. 5, the DG scheme can exploit shared memory parallelism for higher orders as the number of degrees of freedom within an element increases and with them the data parallelism. Due to its simplicity, it is possible to trivially parallelize the matrix vector multiplications in the L2P. Fig. 4 shows the intra-node performance with OpenMP and MPI.

Plotted is the performance for various combinations of MPI processes and OpenMP thread counts on a single node of Hazel Hen. The problem size (approx. 800 million degrees of freedom) is kept constant for all the runs. Thus with an increasing spatial scheme order along the X-axis, the total number of elements in the mesh

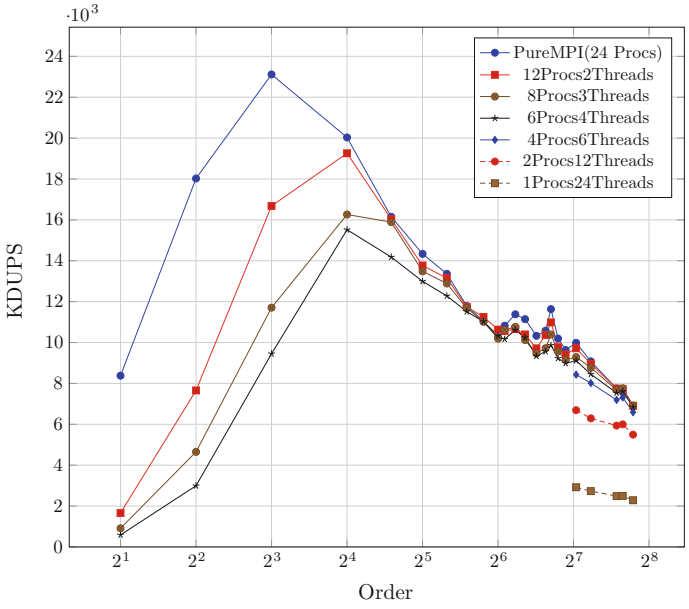


Fig. 4 OpenMP performance of the L2P for various process-thread counts

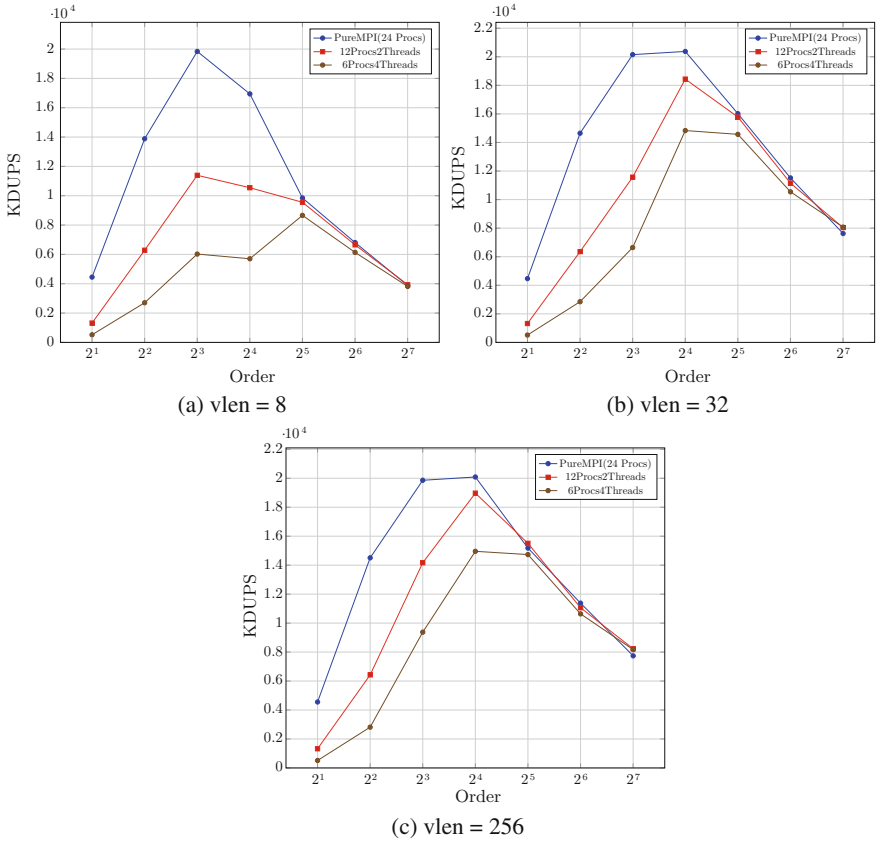


Fig. 5 L2P performance for different proc-thread combination with varying vector length from **a**, **b**, **c**

decreases. Also, the number of elements used is always a multiple of 24, such that it can always be evenly distributed among the up to 24 MPI processes on the single node. This ensures there is no load imbalance due to different number of elements on each MPI ranks and thereby the pure MPI performance does not get distorted. The run with the highest order in the graph makes use of 24 elements. We would expect the performance of hybrid runs to be close to pure MPI runs so that it would enable us to use OpenMP without loosing performance. In Fig. 4 we see the performance of pure MPI is clearly better for scheme orders up to 16. This is expected as for 16th order (or 4096 degrees of freedeom per variable and element), the computational load within a single element is relatively low and insufficient to break even with the overheads introduced by the OpenMP parallelization. However, we can see that the performance of hybrid runs closes in to MPI-only computations for increasing scheme orders. We observe using 4 threads gets us quite close to the performance obtained using 24 MPI processes for high orders. At the same time, it doesn't pay

off to use large number of threads (>6) as the performance deteoriates as soon as threads span multiple NUMA domains. Also, its worth mentioning that hybrid parallelization allows us to fit larger problem per element within a node still attaining optimal performance. For example, when hit the memory bound with a certain order on a node, with shared memory parallelisn we can reduce the number of elements per node to half and use 2 OpenMP threads instead and use even higher orders, while still utilizing all cores.

Many modern hardwares offer possibilities of speeding up the computation using data level parallelization with vectorization. We also exploit this feature inside our L2P implementation by vectorizing our loop operations. We perform matrix vector multiplication using chunks or vectors of specific length. This vector length can be set up during the compilation and helps us boost the performance on vector machines. OpenMP parallelism is implemented on this vector chunks. Thus, it needs to be tuned to obtain the optimum performance on a given system. Figure 3 shows the OpenMP performance for 3 different vector lengths. When the vector size is too small (e.g. Fig. 5a), we see that the OpenMP overheads are too high and, therefore, a larger difference in performance. As we increase the vector length, the OpenMP threads get more work and we see the improvement in the performance of hybrid runs.

7 Conclusion

In this work we presented some of the fast algorithms available as an option to efficiently transform between modal and nodal spaces specially needed when dealing with non-linear terms in modal high order Discontinuous Galerkin methods. We discussed the implementation aspects and the performance comparison of the algorithms we implemented in our code. Then we also talked about hybrid parallelizing the DG method and presented some performance plots highlighting efficient implementation. However, we did not find the performance of the fast algorithms convincing especially for lower orders. They mostly start to pay off for orders which are not feasible for 3D simulations. We found the L2P algorithm quite handy and a decent option since it is easy to optimise because of its simple structure. We still look out for some fast algorithms paying off for order less then hundred. We are further working on implementing shared memory parallelism of our FPT implementation and make it suit our framework and exploit dimension by dimension approach.

Acknowledgements The performance measurement were performed on the Hornet supercomputer at the High Performance Computing Center Stuttgart (HLRS). The authors wish to thank for the computing time and the technical support.

References

1. Hale, N., Townsend, A.: A fast, simple, and stable Chebyshev-Legendre transform using an asymptotic formula. SIAM J. Sci. Comput. **36**(1), A148–A167 (2014)
2. Alpert, B.K., Rokhlin, V.: A fast algorithm for the evaluation of legendre expansions. SIAM J. Sci. Stat. Comput. **12**(1), 158–179 (1991). doi:10.1137/0912009
3. Potts, D., Steidl, G., Tasche, M.: Fast algorithms for discrete polynomial transforms. Math. Comp. **67**(224), 1577–1590 (1998). doi:10.1090/S0025-5718-98-00975-2
4. Iserles, A.: A fast and simple algorithm for the computation of legendre coefficients. Numerische Mathematik **117**(3), 529–553 (2011). doi:10.1007/s00211-010-0352-1
5. Mohlenkamp, M.J.: A fast transform for spherical harmonics. J. Fourier Anal. Appl. **5**(2), 159–184 (1999). doi:10.1007/BF01261607
6. Schaeffer, N.: Efficient spherical harmonic transforms aimed at pseudospectral numerical simulations. Geochem. Geophys. Geosyst. **14**, 751–758 (2013). doi:10.1002/ggge.20071
7. Suda, R., Takami, M.: A fast spherical harmonics transform algorithm **71**(238), 703–715 (2002)
8. Suda, R., Kuriyama, S.: Another preprocessing algorithm for generalized one-dimensional fast multipole method. J. Comput. Phys. **195**(2), 790–803 (2004). doi:10.1016/j.jcp.2003.10.018
9. Suda, R.: Fxtpack. http://sudalab.is.s.u-tokyo.ac.jp/~reiji/fxtpack.html