Discontinuous Galerkin Finite Element Method as Communication-Avoiding Numerical Scheme

Natalia Saiapova *
Fakultät für Informatik
Technische Universität München

July 17, 2017

Abstract

In this paper, we consider Discontinuous Galerkin Finite Element Method for solving partial differential equations with focus on the amount of required communication between processes. The inter-process communication is one of the bottlenecks in parallel computing, therefore, to avoid the communication a domain must be divided on as independent sub-domains as possible. We discuss the finite difference, the finite volume, the finite element, and the discontinious Galerkin finite elements methods, as well as their advantages and disadvantages with regard to accuracy, ability to handle complex geometries and parallelization efficiency. We show that Discontinuous Galerkin finite elements method is spatially compact and does not require setting up a global system of equations, which makes it very attractive for large-scale parallelizations. We present the derivation of the method for one-dimensional hyperbolic conservation law. We experimentally show, that for continuous case the method's order corresponds to the theoretically predicted. The efficiency of the method's parallelization is investigated using weak and strong scaling, with up to 20 nodes of the second phase on SuperMUC cluster. For 20 nodes, we obtained parallel efficiency of more than 80% for strong scaling and more than 95% for weak scaling.

1 Introduction

The communication between processors is one of the bottlenecks in parallel computing. An access to remotely located memory through a network is limited by the network's latency and throughput, which depends on hardware and the distance between source and destination processes. Furthermore, modern architectures rely on tree-like network configurations, which might lead to another problem, when overload of the one branch of the network affects all nodes, which are connected to this branch [1]. To reduce the communication impact one can use computation and communication overlapping, optimized domain decomposition schemes and algorithms which enjoy a locality of the solution and avoid the communication.

One of the common techniques to parallelize a numerical scheme is a domain decomposition, where each process computes the solution in an individual part of the domain separately. In this article we focus on spatial discretization techniques for solving partial differential equations. To approximate a solution at some part of a domain, most algorithms use data from neighboring locations. This data-dependency pattern is called a *stencil* [2]. Since elements on the partition boundaries have data-dependencies on the neighboring partitions, we need to add appropriate communication between processes. To communicate the required data, we add ghost layers of elements to each partition. These layers duplicate the required data from the neighboring partitions and, thus, their size depends on size of the used stencil. Since ghost layers have to be updated with the new data at every timestep, algorithms with a smaller stencil are less affected by a communication factor.

We discuss the finite difference, the finite volume, the finite element, and discontinious Galerkin (DG)

*This work is a part of the seminar "Next Generation High Performance Computing" and I would like to thank my advisors Dr. rer. nat. Vasco Varduhn and M.Sc. Carsten Uphoff.

Preprint submitted as part of the BGCE Scientific Writing seminar – July 17, 2017

finite element methods. For the first- and second-order approximations, the stencils of the first three methods include only direct neighbors, and therefore, the methods can be efficiently parallelized. However, their stencils are not invariant to increasing accuracy order, so for high-order approximations we need more than one ghost-layer and parallelization efficiency will decrease.

We present the discontinuous Galerkin finite element scheme, which combines features of the finite volume and the finite element methods. In contrast to the first three methods, the stencil of this method does not depend on the accuracy order, which makes the DG method very attractive for large scale parallel computing. In fact, the SeisSol software for earthquake simulations, which is based on DG method, shows an efficiency of more than 90% on SuperMUC cluster [3]. Moreover, the authors of [4] developed a framework for unsteady turbulent flow simulations, which parallel efficiency for strong scaling tests exceeds 85% with only 1 element per process for 4096 processors.

For the sake of simplicity, we only study the simple one-dimensional problem with a solution $u = u(t, x)$, where $t \in [0, T]$, $T \geq 0$ and $x \in \Omega$:

$$u_t + f_x(u) = 0. \tag{1}$$

The function $f(u) = f(u(t, x))$ represents a flux. In the scope of this paper we constrain flux to be a linear function:

$$f(u(x, t)) = cu(x, t), \text{where } c \in \mathbb{R}. \tag{2}$$

Initial conditions are given by

$$u(0, x) = u_0(x). \tag{3}$$

We show that for this problem with continuous initial conditions u_0 the order of accuracy corresponds to that determined by Johnson in [5]. Then, following [6], we show a possible parallelization strategy and investigate it with the weak and the strong scaling for up to 20 nodes on the SuperMUC cluster. For 20 nodes, we obtained parallel efficiency of 84% for strong scaling and 96% for weak scaling.

2 Overview of Numerical Schemes

Natural requirements for a discretization method are efficiency, ability to work with complex geometries, high-order accuracy, and easy and efficient parallelization. In the following we discuss finite difference, finite volume, finite element, and DG methods, as well as their advantages and disadvantages.

2.1 Finite Difference Method

The finite difference method is arguably the best-known technique. It discretizes space by a grid, and in every vertex, derivatives are approximated by linear combination of values in the neighboring nodes. The linear combination is obtained from the Taylor series expansion in the neighboring nodes. In the basic case, only direct neighbors are considered. The higher order accuracy is straightforward and usually obtained by increasing the stencil [7], although, as mentioned above, wider stencil limits efficiency of the parallelization. Another drawback of the method is that it is restricted to rectangular grids in its basic form and it is a non-trivial task to include irregular boundary conditions [8].

2.2 Finite Volume Method

Until the 2000s, finite volume methods were most common for solving hyperbolic problems in industry.

Unlike finite difference methods, finite volume methods concentrate on volumes in space, which are called *cells*. Each cell has an indexed center. As an approximation value in a cell we use an volume average $\bar{u}_k(t)$:

$$\bar{u}_k(t) = \frac{1}{h_k} \int_{\Omega_k} u(x, t) dx.$$

We consider now the cell-wise residual of problem 1:

$$R_k(x, t) = \frac{\partial u_h}{\partial t} + \frac{\partial f(u_h)}{\partial x}, \tag{4}$$

where u_h represents an approximated solution in the entire domain and is assumed to be a piecewise constant function consisting of cell averages. The approximation is built on the requirement that, for every cell, the average of the residual vanishes identically [9]:

$$\frac{1}{h_k} \int_{\Omega_k} R_k(x, t) dx = 0. \tag{5}$$

The complete derivation can be found in [10]. It is important to note, that during the derivation, the volume integral of the flux function over cell Ω_k was transformed into the integral over the cell surface $\partial\Omega_k$

Preprint submitted as part of the BGCE Scientific Writing seminar – July 17, 2017

with divergence theorem, and thus, the flux function has to be evaluated on the cell interface. But on the interface between contiguous cells, the solution u_h is doubly defined. To solve this issue, the original flux is reconstructed as function of averages of the both of the adjacent cells $\hat{f}\left(\bar{u}_i, \bar{u}_j\right)$, where i-th and j-th cells are contiguous.

Since this approach does not have any restrictions on the cell form or grid structure, it can easily handle complex geometries. Furthermore, the described approach is purely local and can therefore be efficiently parallelized.

Difficulties occur if one wants to increase the order of accuracy by approximating the solution locally not by a constant but by a polynomial of degree $p \geq 1$. As outlined in [9], the stencil of the method becomes wider, since, instead of one unknown, we have $p + 1$ unknown polynomial's coefficients. In order to find them we need to consider $p + 1$ equations, which can only be obtained if we consider $p+1$ cells, which means increased stencil and thus increase the amount of the inter-process communication. Furthermore, a wider stencil breaks the method's ability to handle complex geometries [9].

2.3 Finite Element Method

The finite element method was introduced as a way to handle complex geometries while retaining the ability to increase the order of accuracy. To ensure spatial flexibility, space is discretized into cells. On each cell, we introduce local basis functions and look for a solution u_h in the form of a linear combination of the basis. A common choice of the basis functions is linear functions. Closeness of the approximation u_h to the real solution u is ensured by the residual being orthogonal to every function from a test space V_h:

$$\int_\Omega \left(\frac{\partial u_h}{\partial t} + \frac{\partial f_h}{\partial x} \right) \psi_h(x) dx = 0, \forall \psi_h \in V_h.$$

With the Galerkin approach, the test space V_h and solution space defined by piecewise linear functions are equal. We expect the solution u_h to be continuous on the cell boundaries. The coefficients of the linear combination of the basis function are new unknowns and to retrieve them a global system of equations has to be set up.

This method is well-suited for unstructured grids. But the main benefit of the method is that we can increase the order of accuracy locally by choosing higher order polynomials on every element. Constructed in that way, method remains suitable for complex geometries.

One of the drawbacks of the method is that while dealing with a time-dependent problem, we have to solve a linear system of equations at each time step [9]. That means that the method is implicit in space. Parallelization is possible while solving the system, although communication pattern is dependent on the problem. As another drawback, authors of [9] argue that this method is not suited for solving problems with discontinuities.

2.4 Discontinious Galerkin Finite Element Method

Discontinuous Galerkin finite element method or shortly DG method combines advantages of finite volume and finite element methods. It uses discretization by elements in space, and, similar to the finite element method, searches for the solution in a form of linear combination of basis functions on each element. The key difference with the basic finite element method is that continuity on the interface is not required by the DG method. As in finite volume methods, that leads to the doubly defined solution on the cell interfaces. In a similar way as it is done for finite volume methods a numerical flux is defined on the interface.

Constructed in such a way the method accuracy can be increased locally so the method remains suited for complex geometries. Additionally, due to discontinuity on the cell interface, no global system is required. The problem can be solved locally on an element, which makes the discontinuous Galerkin method a good candidate for parallelization.

3 DG Method Derivation

In this section we derive DG method for one-dimensional problem 1. We divide the domain Ω into K elements $\Omega_k = [x_k, x_{k+1}]$, where $1 \leq k \leq K$, such that $\Omega = \bigcup_{k=1}^{K} \Omega_k$. We refer local size of each element as $h_k = x_{k+1} - x_k$.

Preprint submitted as part of the BGCE Scientific Writing seminar – July 17, 2017

3.1 Weak Formulation

We now transform initial problem 1 to a weak formulation. We introduce a globally defined test space of piecewise continuous functions $V_h = \bigoplus_{k=1}^{K} V_h^k$, where $V_h^k = \text{span}\{\psi_n^k\}_{n=0}^{n=N}$ are locally defined finite-dimensional test spaces. Galerkin approach enforces the approximation u_h of the solution u to be defined in V_h.

Now we enforce the residual to be orthogonal to the space V_h on every element Ω_k:

$$\int_{\Omega_k} \frac{\partial u_h}{\partial t}\psi_i^k + \frac{\partial f(u_h)}{\partial x}\psi_i^k dx = 0. \tag{6}$$

After integration by parts of the second term, we arrive at the weak formulation of problem 1:

$$\begin{aligned} &\int_{\Omega_k} \frac{\partial u_h}{\partial t}\psi_i^k dx - \int_{\Omega_k} f(u_h)\frac{\partial \psi_i^k}{\partial x}dx + \\ &+\hat{f}(u_{k+1}^-, u_{k+1}^+)\psi_i^k(x_{k+1}) - \hat{f}(u_k^-, u_k^+)\psi_i^k(x_k) = 0. \end{aligned} \tag{7}$$

Since we do not ensure uniqueness of the solution on the cell interfaces, in the formula above we introduce a numerical flux function $\hat{f}(u^-, u^+)$. Arguments u_k^- and u_k^+ are the values of the solution u_h on the contrary sides of the node x_k. The choice of an appropriate numerical flux function depends on a problem and is discussed in [9]. A common solution is Lax-Friedrichs flux:

$$\hat{f}(u^-, u^+) = \frac{1}{2}\left(f(u^-) + f(u^+) + |\lambda|(u^- - u^+)\right),$$

where λ represents the maximal eigenvalue of the Jacobi matrix of the function f.

3.2 Parameterization of Elements

In order to simplify calculations, we parameterize every element Ω_k through a canonical element $\Omega_0 = [-1, 1]$:

$$\begin{aligned} \xi_k : \Omega_0 &\to \Omega_k \\ r &\mapsto \frac{1}{2}\left(h_k r + x_k + x_{k+1}\right). \end{aligned}$$

Let $J_k = \partial\xi_k(r)/\partial r = h_k/2$. Then, in terms of r and $\xi_k(r)$, dx can be written as

$$dx = \frac{\partial \xi_k(r)}{\partial r}dr = J_k dr.$$

And the argument of the second integral in the equation 7 can be expressed through basis functions on the canonical element $\phi_i(r)$:

$$\frac{\partial \psi_i^k(x)}{\partial x} = \frac{\partial \psi_i^k(\xi_k(r))}{\partial r}\frac{\partial r}{\partial \xi_k(r)} = \frac{\partial \phi_i(r)}{\partial r}J_k^{-1}.$$

Now lets consider the evaluation of the flux function on the cell interface in the equation 7. Further we refer to this value as F_{ki}:

$$\begin{aligned} F_{ki} &= \hat{f}(u_{k+1}^-, u_{k+1}^+)\psi_i^k(x_{k+1}) - \hat{f}(u_k^-, u_k^+)\psi_i^k(x_k) = \\ &= \hat{f}(u_{k+1}^-, u_{k+1}^+)\phi_i(1) - \hat{f}(u_k^-, u_k^+)\phi_i(-1). \end{aligned}$$

The equation 7 takes the following form:

$$J_k \int_{\Omega_0} \frac{\partial u_h}{\partial t}\phi_i(r)dr - \int_{\Omega_0} f(u_h)\frac{\partial \phi_i(r)}{\partial r}dr + F_{ki} = 0. \tag{8}$$

3.3 Solution Approximation

We assume that the solution approximation $u_h \in V_h$ is piecewise continuous function $u_h = \bigoplus_{k=1}^{K} u_h^k$ and in each element Ω_k the local solution is represented by

$$u_h^k(\xi_k(r)) = \sum_{j=0}^{N} a_j^k \phi_j(r), \tag{9}$$

where functions $a_j^k = a_j^k(t)$ do not depend on spacial variable. The equation 8 gets form

$$J_k \sum_{j=0}^{N} (a_j^k)_t M_{ij} - \int_{\Omega_0} f\left(\sum_{j=0}^{N} a_j^k \phi_j(r)\right)\frac{\partial \phi_i(r)}{\partial r}dr + F_{ki} = 0. \tag{10}$$

where mass matrix M is defined by

$$M_{ij} = \int_{\Omega_0} \phi_i(r)\phi_j(r)dr.$$

The equation 10 can be already solved with time integration, although evaluation of the volume integral is required every on timestep. The evaluation are typically done by Gaussian quadrature rules. But this approach is not well-suited for our algorithm since we store the solution as a vector of the coefficients. In 1996 H.L. Atkins and Chi-Wang Shu developed the quadrature-free DG method, where flux function is

Preprint submitted as part of the BGCE Scientific Writing seminar – July 17, 2017

expanded in terms of basis functions $\phi_j(r)$ [11].

In our case with linear flux function 2 the expansion is trivial and after plugging it into the equation 10 we retrieve the quadrature free form:

$$J_k \sum_{j=0}^{N} M_{ij}(a_j^k)_t - c \sum_{j=0}^{N} S_{ij} a_j^k + F_{ki} = 0, \qquad (11)$$

where matrix S is a stiffness matrix defined by

$$S_{ij} = \int_{\Omega_0} \phi_j(r) \frac{\partial \phi_i(r)}{\partial r} dr.$$

Note that matrices M and S can be pre-computed.

3.4 Time Integration

At the end, the equation 11 can be written in the vector form:

$$J^k (M \boldsymbol{a}_t^k)_i + F_{ki} - c(S\,\boldsymbol{a}^k)_i = 0.$$

The vector $\boldsymbol{a}^k$ represents the vector of the unknown coefficients of the basis expansion and is time-dependent. The final semidiscrete system

$$a_t^k = M^{-1} J^{-1} (cS\,\boldsymbol{a}^k - F_{k*}) \qquad (12)$$

can be solved by standard time integration schemes, for instance, explicit fourth-order Runge-Kutta methods with four stages [9].

3.5 Initial Conditions

The next step is to approximate the initial conditions 3. For the function u_0 we want to find such linear combination of the basis functions that

$$\int_{\Omega_k} u_0 \phi_i^k dx = \int_{\Omega_k} \left(\sum_{j=0}^{N} a_j^k(0) \phi_j^k \right) \phi_i^k dx.$$

The right-hand side of the equation can be written in terms of matrix M:

$$\int_{\Omega_k} \left(\sum_{j=0}^{N} a_j^k(0) \psi_j^k \right) \psi_i^k dx = \sum_{i=0}^{N} a_j^k(0) J^k M_{ij} = J^k (M \boldsymbol{a}_0^k)_i,$$

where $\boldsymbol{a}_0^k \in R^{N+1}$ and $(\boldsymbol{a}_0^k)_j = a_j^k(0)$.

Through quadrature or analytic solution we can compute the integral directly:

$$\int_{\Omega_k} u_0(x) \phi_i^k(x) dx = J^k \int_{\Omega_0} u_0(\xi_k(r)) \phi_i(r) dr.$$

Which leads us to the linear system

$$\boldsymbol{a}^k(0) = M^{-1} U^k,$$

where vector U^k is defined by $U_i^k = \int_{\Omega_0} u_0(\xi_k(r)) \phi_i(r) dr$.

3.6 Basis Functions

A common choice for finite element space is polynomial space $\mathbb{P}_N([-1, 1])$ up to degree N. This approach is beneficial since matrices M and S can be computed exactly using Gauss quadrature rules.

A first guess of the basis would be the monomial basis [1, x, x^2, x^3, ... x^N], although the mass matrix formed by this basis has the Hilbert form. The condition number of the $n \times n$ Hilbert matrix grows as $O(1 + \sqrt{2})^{4n} / \sqrt{n}$ [12] and, consequently, when N is considerably large (e.g. $N > 10$), we can not accurately compute M^{-1} and the overall problem is ill-conditioned.

The solution of this issue, as demonstrated in [9], is obtained by orthogonalization of the monomial basis. The resulting polynomials can be written as

$$\phi_n(r) = \frac{L_n(r)}{\sqrt{\gamma_n}},$$

where $L_n(r)$ is Legendre polynomial of degree n. Coefficient γ_n serves for normalization and equals

$$\gamma_n = \frac{2}{2n + 1}.$$

With this basis the mass matrix becomes an identity matrix of a dimension $N + 1 \times N + 1$.

4 Accuracy of Solution

To verify that the solver works as expected, we test it with the linear problem 1 on the uniform mesh. The analytical solution of this problem is a linear shift with velocity c:

$$u(t, x) = u_0(x - ct).$$

Preprint submitted as part of the BGCE Scientific Writing seminar – July 17, 2017

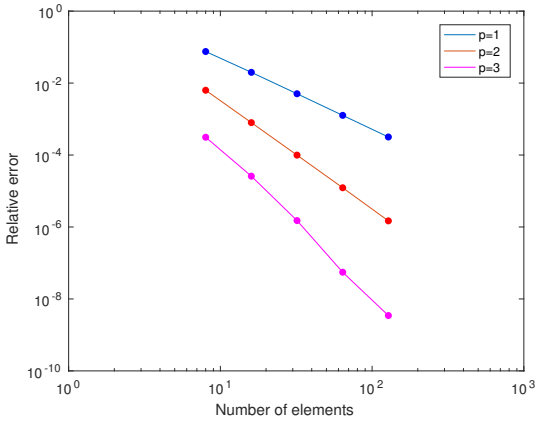

Figure 1: L_2 norm of relative errors for linear one-dimensional continuous problem with periodic boundary conditions after time $t = 0.1$ using Legendre polynomials of degree p equal to 1, 2, and 3.

We consider periodic boundary conditions and initial conditions

$$u_0 = \sin(\pi x), \text{ where } x \in [-1, 1]. \tag{13}$$

In 1986 Johnson [5] proved that in continuous case for basis polynomials of degree p, for elements small enough the solver has accuracy order at least $p + \frac{1}{2}$ on general meshes. Although, in practice order of $p + 1$ is usually observed [13].

Errors after $t = 0.1$ are presented in Fig. 1 for polynomials of degree 1, 2, and 3. We can see that obtained order is actually equal to $p + 1$.

5 Parallelization

In this section, following [6], we describe one of the ways to parallelize the DG method.

5.1 Strategy

One timestep of a sequential version of the above described algorithm is:

1. Compute u^- and u^+ for every node x_k;
2. Compute F_{k*} for every element;
3. Compute vector of coefficients $\boldsymbol{a_k}$ using explicit time integration scheme for every element.

Due to its compact formulation, the DG method is perfectly suited for highly efficient parallelizations. The next parallelization approach is easy to implement but leads to redundant flux calculations on partition boundaries. We assume that we have p processes with K/p local elements each. By Ω_b we denote any element on the partition boundaries and by Ω_I we denote any inner element. The outline of the strategy is following:

1. Retrieve the solution $u_h|_{\partial\Omega_b}$ on the partition boundaries $\partial\Omega_b$;
2. Non-blocking send of $u_h|_{\partial\Omega_b}$ to the neighboring partitions;
3. Compute fluxes for inner elements;
4. Compute new coefficients for inner elements;
5. Receive $u_h|_{\partial\Omega_b}$ from the neighboring partitions;
6. Compute coefficients on boundaries Ω_b.

This sequence of calculations ensures that the maximal overlap of communications and computation occurs. In our one-dimensional linear case, at each timestep it is enough to communicate to neighbors only the obtained by the process solution on the interface.

The drawback of this approach is that on the boundary of partitions fluxes are calculated twice, by each of neighboring processes. There are possible strategies of avoiding these redundant computations. As authors of [6] presented, a process can be marked as owner of an edge if it is calculating the numerical flux of this edge. Then the process sends the computed value to the adjacent process. This strategy leads to more complicated pattern of computations and communications, where the maximal overlap is lost.

5.2 Parallel Efficiency and Scaling

The test code is written on C++11 using MPI non-blocking point-to-point communication. Linear algebra computations is done with the sequential version of Intel MKL library. The code was compiled with Intel compiler version 16.0 and Intel MPI 5.1. We run the program on Haswell nodes on the SuperMUC cluster, phase 2. Each node can handle 28 tasks. Details about processor type can be found under the link [14].

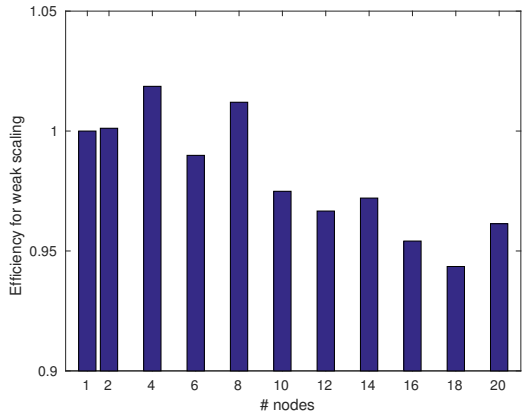

Figure 2: Weak scaling efficiency on the SuperMUC phase 2 nodes. Each node computes $28 \cdot 10^5$ elements with Legendre polynomials of the degree 3.

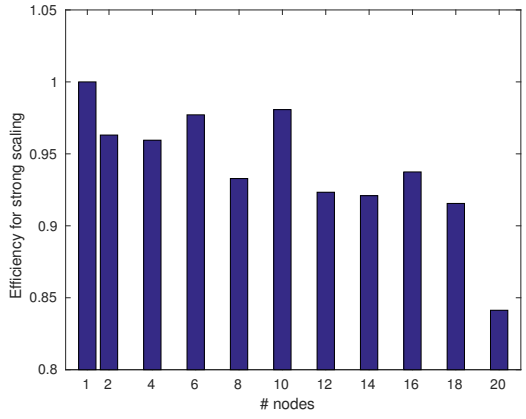

Figure 3: Strong scaling efficiency on the SuperMUC phase 2 nodes. The problem is solved with $4.51584 \cdot 10^7$ elements with Legendre polynomials of the degree 3.

The amount of work was defined in [15] as a product of three values: the number of elements, the number of timesteps (T), and the number of basis functions ($N+1$). As a test case we use the linear continuous problem on the uniform grid 13.

5.3 Weak scaling

By the weak scaling we refer tests where local amount of work is fixed per process. Consequently, when the number of processes p is increased the global problem size is increased by the same factor.

We fix the problem size with $K = 1 \cdot 10^5$ elements, $T = 50$ timesteps, and Legendre polynomials of degree $N = 3$. We run the program 10 times for every p and as result take the mean value of execute times. On the figure 2 we show that our implementation has an average efficiency of more than 90% up to 20 nodes on the SuperMUC cluster.

5.4 Strong scaling

For strong scaling we fix the global problem. That means that for size remains the same although number of processes is increasing.

Figure 3 shows strong scaling efficiency. Global problem size is fixed with $K = 4.51584 \cdot 10^7$ elements, $T = 50$ timesteps, and Legendre polynomials of degree $N = 3$. In order to obtain resulting efficiency we again compute the mean value of 10 runs. We achieved the efficiency of more than 80% for the 20 nodes of the SuperMUC cluster.

6 Conclusion and Outlook

This work substantiates benefits of the discontinuous Galerkin finite element (DG) methods in comparison with finite difference, finite volume, and finite element techniques with regard to parallelization, accuracy and ability to handle complex geometries. We demonstrate the derivation of the method for one-dimensional advection problem. The accuracy of the method can be increased locally on each element while retaining parallel efficiency. In fact, parallel efficiency of the higher-order method is expected to be higher due to increasing computational work inside each sub-domain.

We implemented the described algorithm and test it on the continuous problem with uniform mesh. Our implementation achieved an efficiency of more than 90% with respect to weak scaling and more than 80% with respect to strong scaling for 20 phase 2 nodes on the SuperMUC cluster.

The source code can be found on [16].

References

[1] A. Greenberg *et al.*, "Vl2: A scalable and flexible data center network," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, Aug. 2009.

[2] D. J. Holmes and C. Laoide-Kemp, "Streams as an alternative to halo exchange," in *ParCo*, 2015.

[3] A. Heinecke *et al.*, "Petascale high order dynamic rupture earthquake simulations on heterogeneous supercomputers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14, 2014, pp. 3–14.

[4] F. Hindenlang, G. J. Gassner, C. Altmann, A. Beck, M. Staudenmaier, and C.-D. Munz, "Explicit discontinuous galerkin methods for unsteady problems," *Computers & Fluids*.

[5] C. Johnson and J. Pitkäranta, "An analysis of the discontinuous galerkin method for a scalar hyperbolic equation," *Math. Comput.*, vol. 46, no. 173, pp. 1–26, Jan. 1986.

[6] A. Baggag, H. Atkins, and D. Keyes, "Parallel implementation of the discontinious galerkin method," *ICASE Report*, no. 99-35, 1999.

[7] R. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems (Classics in Applied Mathematics Classics in Applied Mathemat)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007.

[8] B. Gustafsson, *Finite Difference Methods*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 145–171.

[9] J. S. Hesthaven and T. Warburton, *Nodal Discontinuous Galerkin Methods, Algorithms, Analysis, and Applications*, 2008.

[10] R. LeVeque, *Finite Volume Methods for Hyperbolic Problems*, ser. Cambridge Texts in Applied Mathematics.

[11] H. L. Atkins and C.-W. Shu, "Quadrature-free implementation of the discontinuous galerkin method for hyperbolic equations," *AIAA Journal*, vol. 36, pp. 775–782, 1996.

[12] B. Beckermann, "The condition number of real vandermonde, krylov and positive definite hankel matrices," 2000.

[13] R. Hartmann, "Discontinuous Galerkin methods for compressible flows: higher order accuracy, error estimation and adaptivity," in *VKI LS 2006-01: CFD-Higher Order Discretization Methods, Nov. 14-18, 2005*, H. Deconinck and M. Ricchiuto, Eds. Von Karman Institute for Fluid Dynamics, Rhode Saint Genèse, Belgium, 2005.

[14] Intel, *Intel Xeon Processor E5-2697 v3*, accessed 23 January, 2017. [Online]. Available: http://ark.intel.com/de/products/81059/Intel-Xeon-Processor-E5-2697-v3-35M-Cache-2_60-GHz

[15] R. Biswas, K. D. Devine, and J. E. Flaherty, "Parallel, adaptive finite element methods for conservation laws," *APPL. NUMER. MATH*, vol. 14, pp. 255–283, 1994.

[16] N. Saiapova, *Source Code*, 23 January, 2017. [Online]. Available: https://github.com/nasay/seminar-DG.git