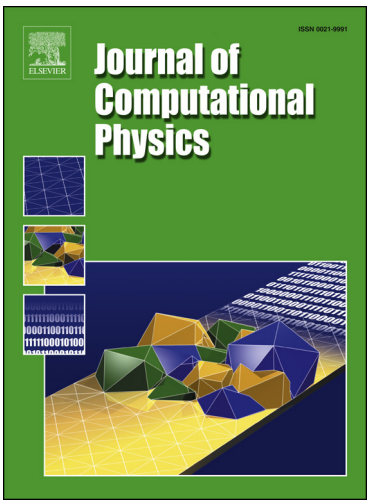
Accepted Manuscript



A performance comparison of semi-Lagrangian discontinuous Galerkin and spline based Vlasov solvers in four dimensions

Lukas Einkemmer

PII: S0021-9991(18)30669-7
DOI: https://doi.org/10.1016/j.jcp.2018.10.012
Reference: YJCPH 8314

To appear in: *Journal of Computational Physics*

Received date: 28 March 2018
Revised date: 24 August 2018
Accepted date: 4 October 2018

Please cite this article in press as: L. Einkemmer, A performance comparison of semi-Lagrangian discontinuous Galerkin and spline based Vlasov solvers in four dimensions, *J. Comput. Phys.* (2018), https://doi.org/10.1016/j.jcp.2018.10.012

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

Highlights

- The SLDG method is efficient for high dimensional Vlasov simulation.
- We obtain superior performance compared to a well established implementation.
- The local nature of SLDG helps significantly in parallelization (e.g. on GPUs).
- The performance in the asymptotic regime is a very poor predictor of accuracy.
- First performance comparison of SLDG in more than two dimensions.

ACCEPTED MANUSCRIPT

A performance comparison of semi-Lagrangian discontinuous Galerkin and spline based Vlasov solvers in four dimensions[☆]

Lukas Einkemmer[a,b,*]

[a] *Department of Mathematics, University of Tübingen, Germany*
[b] *Department of Mathematics, University of Innsbruck, Austria*

Abstract

The purpose of the present paper is to compare two semi-Lagrangian methods in the context of the four-dimensional Vlasov–Poisson equation. More specifically, our goal is to compare the performance of the more recently developed semi-Lagrangian discontinuous Galerkin scheme to cubic spline interpolation, which is widely used in Eulerian Vlasov simulation. To that end, we perform simulations for nonlinear Landau damping and a two-stream instability and provide benchmarks for the SeLaLib and SLDG codes, both on a workstation and using MPI on a cluster. In addition, we will present results for the graphic processing unit (GPU) implementation contained in SLDG.

We find that the semi-Lagrangian discontinuous Galerkin scheme shows a moderate improvement in run time for nonlinear Landau damping and a substantial improvement for the two-stream instability. It should be emphasized that these results are markedly different from results obtained in the asymptotic regime, which favor spline interpolation. Thus, we conclude that the traditional approach of evaluating numerical methods is misleading, even for short time simulations. In addition, the absence of any global communication in the semi-Lagrangian discontinuous Galerkin method gives it a decisive advantage for scaling to more than 256 cores.

Keywords: semi-Lagrangian discontinuous Galerkin method, Vlasov–Poisson equation, semi-Lagrangian methods, performance comparison, parallelization

1. Introduction

Many problems in computational plasma physics require the solution of kinetic equations. Due to the high dimensionality of these models, particle meth-

[☆]The computational results presented have been achieved [in part] using the Vienna Scientific Cluster (VSC).

[*]Corresponding author
Email address: einkemmer@uibk.ac.at (Lukas Einkemmer)

ACCEPTED MANUSCRIPT

ods have been, and still are, heavily used to conduct these simulations (see, for example, the review article [56]). However, due to the rapid increase in computational performance, numerical methods based on the discretization of the entire phase space have gained increasing attention in recent years, at least for four-dimensional and certain five-dimensional problems. These so-called Eulerian methods offer superior accuracy and the possibility to resolve regions of the phase space with low density, but are extremely demanding from a computational point of view. Consequently, it is important to develop efficient numerical methods that can be parallelized to state of the art supercomputers. A comparison of different methods has been performed in [32, 2]. We also refer the reader to the recent review article [48] focusing on astrophysical plasmas.

While standard techniques from fluid dynamics can be applied to kinetic equations, these usually suffer from a severe restriction in the time step size due to a Courant–Friedrichs–Lewy (CFL) condition. On the other hand, the use of implicit methods would significantly increase the memory footprint, which is already a scarce resource for high-dimensional simulations. This is an even more pronounced issue for accelerators such as graphic processing units (GPUs) and the Intel Xeon Phi, which in all likelihood will become more pervasive in the future. Semi-Lagrangian methods offer a viable alternative. These methods follow the characteristics backward in time. In many kinetic models, such as the Vlasov–Poisson and Vlasov–Maxwell equations, applying a splitting procedure results in sub-steps for which the characteristics can be determined analytically. If this is not the case an explicit ordinary differential equation solver is usually used. For the Vlasov–Poisson equation this approach has been first suggested in the seminal paper of Cheng & Knorr [11]. Later it was extended in various ways to the Vlasov–Maxwell equation [13, 53, 46]. However, since the endpoint of a characteristic curve does not necessarily coincide with the grid used, an interpolation procedure has to be employed. An attractive choice is to reconstruct the desired function by spline interpolation (see, for example, [54, 32]). This is arguably one of the most commonly employed scheme and a range of software packages have been developed that implement it (see, for example, [7, 52, 1]). Its performance has also been extensively compared to a number of different numerical methods (see, for example, [2, 32]). Consequently, we will use it as the reference scheme in the performance comparisons that are conducted in this paper. The main downside of this approach is that a tridiagonal linear system of equations has to be solved to construct the spline. While this can be done very efficiently in a sequential implementation, for modern supercomputers, which require scaling to hundreds of thousands of cores, such global dependencies considerably limit the utility of this algorithm.

On the other hand, the semi-Lagrangian discontinuous Galerkin method employs a piecewise polynomial approximation in each cell of the computational domain (see, for example, [51, 19, 26, 49]). In the case of an advection equation the discretized function is translated and then projected back to the appropriate subspace of piecewise polynomial functions. This method, per construction, is mass conservative and only accesses two adjacent cells in order to compute the necessary projection (this is true independent of the order of the approxima-

ACCEPTED MANUSCRIPT

tion). The local nature of this method also makes it easy to incorporate block structured mesh refinement and to employ limiters to maintain positivity. Positivity limiters were also investigated for spline interpolation (see [58, 59, 18]). Note, however, that limiters in the context of spline interpolation are much more involved compared to the discontinuous Galerkin case. For the semi-Lagrangian discontinuous Galerkin scheme mathematical rigorous convergence results are available in [51, 27, 26]. The method is fully explicit. That is, no linear system has to be solved to advance the solution in time and, consequently, the semi-Lagrangian discontinuous Galerkin scheme is easier to implement. This is particularly true on parallel architectures where the algorithm shows a more favorable communication pattern. In addition, a very efficient mixed precision implementation is available for the semi-Lagrangian discontinuous Galerkin method [22]. This is particular useful for modern accelerators (such as GPUs) and helps to alleviate the problem of memory scarcity on such systems. Furthermore, it is worth mentioning that a filamentation filtration strategy has been developed [25] for problems where numerical recurrence is an issue and that the error propagation is superior to other semi-Lagrangian schemes [55, 28]. Although, to our knowledge, for the latter no proof is available in the context of the Vlasov equation . Let us also note that the semi-Lagrangian discontinuous Galerkin scheme works best for the constant advection case. Consequently, it is often combined with a time splitting approach, as we do here. However, some work has been conducted to extent this approach to non-constant advections [20, 39, 8, 9] and even to relatively simple nonlinear problems [29]. To conclude this discussion, we should note that some measures have been taken to improve the parallel scalability of cubic spline interpolation [17]. However, even for this approach a relatively large communication overhead is incurred. This is due to the fact that the boundary condition for the local spline reconstruction requires a large stencil if the desirable properties of the global cubic spline interpolation are to be preserved (the method derived in [17] requires a centered stencil of size 21).

Before proceeding, let us note that a range of other methods have been used to solve the Vlasov equation. A method that is related to the discontinuous Galerkin approach and which has been widely employed in plasma simulations is the van Leer scheme (see, for example, [31, 46, 33, 10]). What distinguishes the discontinuous Galerkin method from the van Leer scheme is that the dG scheme is completely formulated in terms of the coefficients in the corresponding basis expansion. In contrast, the van Leer scheme replaces these coefficients by performing an approximation using suitable differences on an equidistant grid. Its implementation and performance characteristics are thus more closely related to other finite difference schemes. In the context of semi-Lagrangian methods a range of interpolation procedures have been proposed. This includes Lagrange interpolation, Hermite interpolation, and the Cubic interpolated propagation (CIP) scheme [47]. These schemes are usually referred to as backward semi-Lagrangian methods. On the other hand, forward semi-Lagrangian methods operate by moving a given amount of mass forward in time and then depositing it at an appropriate location. The semi-Lagrangian discontinuous Galerkin

ACCEPTED MANUSCRIPT

scheme, considered in this paper, and the convected scheme [36, 37] are examples in that class. Interpolation using cubic splines, as discussed before, is usually formulated as a backward semi-Lagrangian scheme. However, at least, for the purpose of analysis it is often advantageous to write it as a forward semi-Lagrangian scheme (see, for example, [21, 24]).

A different approach to overcoming the restriction on the time step size is to use block structured adaptive mesh refinement [42, 43, 44, 57]. This is a promising approach as in many problems the most severe CFL condition is induced by a small number of cells that represent regions in phase space with large velocities. In addition, these schemes are able to refine the grid in regions of interest. Implicit numerical methods are another approach to overcome the CFL condition. This is a delicate approach as the amount of memory required and the computational effort has to be kept to a minimum. Note that even a matrix free implementation has to store the vectors of the Krylov subspace and is thus expensive with respect to storage. Recently an implicit Runge–Kutta integrator based on sweeping techniques has been proposed that is able to overcome some of these limitations [34].

In addition to the methods outlined above, more traditional discretization strategies have been considered as well. These include finite volume [30, 3, 16] and discontinuous Galerkin [41, 12] schemes.

Despite the many advantages of the semi-Lagrangian discontinuous Galerkin method outlined above, there is certainly no universal agreement that this method will be a viable alternative for large scale high-dimensional kinetic simulation. The main concern is that if a method requires more degrees of freedom per direction to achieve the same precision, compared to say spline interpolation, then in a higher-dimensional setting the computational cost increases significantly. To be more precise, let us assume that instead of n degrees of freedom per direction our numerical approach requires αn, with $\alpha > 1$. Then the memory footprint and run time are increased by a factor α^d, where d is the dimensionality of the problem. For example, even a moderate increase, for example $\alpha = 1.2$, results in a substantial increase of memory footprint/run time (a factor of 2 for $d = 4$ and factor of 3 for $d = 6$). It would then be difficult to overcome such a disadvantage by, for example, better scaling.

Of course, the important question then becomes if the semi-Lagrangian discontinuous Galerkin method actually requires more degrees of freedom and if so by how much. This is a common problem for classic (i.e. non semi-Lagrangian) discontinuous Galerkin methods. In this context so-called hybrid schemes (see, for example, [45]) have been developed to reduce the number of degrees of freedom. However, the situation considered here is quite different. Specifically, the problem is reduced to a sequence of one-dimensional advection equations which are then solved using the method of characteristics. In the present paper we will investigate if the semi-Lagrangian discontinuous Galerkin method requires more degrees of freedom compared to spline interpolation. This will be done in

ACCEPTED MANUSCRIPT

the context of the Vlasov–Poisson equation

$$\begin{aligned} &\partial_t f(t,x,v) + v \cdot \nabla_x f(t,x,v) + E(f)(x) \cdot \nabla_v f(t,x,v) = 0 \\ &\nabla \cdot E(f)(x) = \int f(t,x,v)\,\mathrm{d}v - 1, \qquad \nabla \times E(f)(x) = 0, \end{aligned} \tag{1}$$

This equation is commonly used to model the time evolution of collisionless plasmas. Then f is the electron density function and E is the self-consistent electric field. In this setting a uniform background density of ions is assumed. In dimensionless units the ion density is equal to 1 and the normalization $\int f\,\mathrm{d}(x,v) = 1$ is assumed. This implies that the entire system is neutral. However, local variations in the density do occur. These variations in charge density self-consistently determine the electric field according to a Poisson equation

$$\Delta\phi = \int f(t,\cdot,v)\,\mathrm{d}v - 1, \qquad E = \nabla\phi.$$

In all our numerical results we will employ periodic boundary conditions in both the x- and v-directions. Note that since the electron density function decays exponentially for large v, periodic boundary conditions in the velocity direction are justified as long as the computational domain is chosen large enough.

The Vlasov–Poisson equation has a number of physically relevant invariants. In particular, mass

$$\int f(t,x,v)\,\mathrm{d}(x,v)$$

momentum

$$\int v f(t,x,v)\,\mathrm{d}(x,v)$$

and energy

$$\frac{1}{2}\int v^2 f(t,x,v)\,\mathrm{d}(x,v) + \frac{1}{2}\int E^2(t,x)\,\mathrm{d}x.$$

In addition, all the L^p norms, for $p \in \mathbb{N}_{\geq 1}$ are conserved. These invariants can be used to evaluate the fidelity of a given numerical method.

2. The numerical error in Vlasov simulation

Before we discuss the comparative efficiency of the semi-Lagrangian discontinuous Galerkin method (section 4) and the four-dimensional simulations conducted (sections 5 and 6), let us dwell a bit more on the argument against the discontinuous Galerkin method outlined above. To do that we will consider nonlinear Landau damping as a test problem and, for the time being, restrict ourselves to the two-dimensional case (for reasons that will be apparent shortly; the four-dimensional variant of this problem is discussed in section 5). That is, we consider the initial value

$$f(0,x,v) = \frac{1}{\sqrt{2\pi}}(1 + \epsilon\cos(kx))\mathrm{e}^{-\frac{1}{2}v^2},$$

ACCEPTED MANUSCRIPT

where $\epsilon = \frac{1}{2}$ and $k = \frac{1}{2}$, on the domain $[0, 4\pi] \times [-6, 6]$. A splitting procedure is employed to obtain a sequence of advection equations which are discretized using either a semi-Lagrangian approach based on spline interpolation or the semi-Lagrangian discontinuous Galerkin approach. Both numerical methods will be discussed in some detail in section 3. The error committed by these two numerical methods is computed by comparing it to a reference solution with a sufficiently fine resolution.

Many publications (see, for example, [24, 19, 12]), including work by the author of this paper, have reported such results. To facilitate the present discussion. we have computed the error in the particle-density f at time $t = 10$ as a function of the degrees of freedom (per direction). These results are shown on the left in Figure 1. We clearly observe that, at best, the discontinuous Galerkin method struggles in a direct comparison with spline interpolation. The sixth-order discontinuous Galerkin method (dG6) requires approximately 50% more degrees of freedom to obtain the same accuracy. Thus, it seems that the discussion has come to a close and the criticism outlined above seems to be completely justified. However, the major caveat here is that in practical applications it is usually not of interest to solve nonlinear Landau damping only up to $t = 10$. Now, if we look at the error at $t = 50$ (right-hand side of Figure 1), the error does not appreciably decrease at all for up to 2048 degrees of freedom per direction and for any of the numerical methods considered. In the former case the simulation is conducted in the so-called asymptotic regime. That is, there is a clear reduction in the error as the number of degrees of freedom is increased. The rate of decrease is usually determined by the order of the method. Let us note that the results obtained from most mathematical convergence proofs describe the behavior of the numerical method in the asymptotic regime rather well (see, for example, [5, 6, 50, 26, 27]). In contrast, for the latter case the error is close to $\mathcal{O}(1)$ independent of the number of degrees of freedom used. If the number of degrees of freedom is further increased, we would eventually enter the asymptotic regime. However this is often prohibitive from a computational point of view, especially for high dimensional problems.

To even run a simulation with 2048 degrees of freedom in four dimensions would require at least 280 TB of memory, which would stress even the world's most powerful supercomputers. Note that the number one system on the present TOP500 supercomputing list has 1310 TB of memory. Thus while such a simulation would be possible in principle, it would certainly not very practical. We thus conclude that while the argument provided is valid for very small times (i.e. in the asymptotic regime), it is all but futile for realistic simulation. The question that we should ask instead is how well does a numerical method resolve the physics of the problem under consideration (assuming reasonable problem sizes).

The example considered here is two-dimensional and thus should not be considered the gold standard for higher-dimensional simulations. In fact, the reason why we consider this in two dimensions is that it is prohibitively expensive to run simulations with 2048^4 or even more degrees of freedom. This paper will show that for the four-dimensional case the performance of a numerical

ACCEPTED MANUSCRIPT

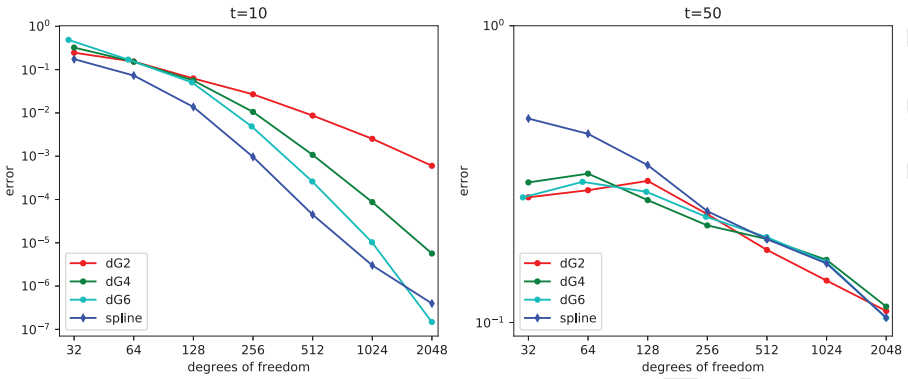


Figure 1: The error, as a function of the number of degrees of freedom (per direction), for nonlinear Landau damping is shown at $t = 10$ (left) and $t = 50$ (right). The error is computed in the infinity norm and is normalized by the norm of the reference solution. The reference solution is computed using spline interpolation with 8192 grid points (per direction). For all simulations a Strang splitting procedure with time step size $\tau = 0.1$ has been used.

method in the asymptotic regime is markedly different from the performance of interest in realistic simulations. More specifically, we will show that even though the semi-Lagrangian discontinuous Galerkin method is inferior to spline interpolation for small times, depending on the problem, similar or even superior performance can be achieved for more realistic simulations.

One might argue that this behavior is not entirely surprising. In the context of Hamiltonian systems the field of geometric numerical integration has been devoted to numerical methods that provide qualitatively correct results even if the error in the numerical solution is large. Such methods are, for example, used in the long time integration of the solar systems. In this context one is not interested in the exact position of a given planet, but rather on the general stability of the solar system as a whole. From a mathematical standpoint rigorous results can be shown regarding the long time behavior of the error in energy. That is, it can be shown, for certain methods, that although energy is not exactly conserved, the error stays small over very long time intervals. We refer the interested reader to the book [40]. In the context of partial differential equations, in general, and the Vlasov equation, in particular, it has also been pointed out that to faithfully resolve the physics over long times requires specifically constructed numerical methods. These methods usually conserve certain physical invariants and are thus labeled conservative methods. Furthermore, a numerical method should produce as little numerical diffusion as possible. Many examples of such schemes can be found in the literature. For the Vlasov equation see, for example, [18, 32] for spline interpolation and [24, 19] for the discontinuous Galerkin approach.

What distinguishes the present situation from these considerations is that in Figure 1 the result at $t = 50$ already shows the behavior we would expect for long times. However, at $t = 50$ we have only conducted 500 time steps.

ACCEPTED MANUSCRIPT

Thus, even for relatively short times the performance in the asymptotic regime is less important than the qualitative properties of the numerical solution. The mechanism that causes the observed behavior is filamentation. That is, the appearance of smaller and smaller scales in phase space limits the accuracy of the numerical method. Due to computational constraints resolving these fine scales is only possible for very short times. Since the Vlasov equation is often used to simulate plasma instabilities, the time evolution of certain averaged quantities, most notably the total electric energy, are of significant interest. These quantities can often be described accurately even if the error in the phase space distribution is large. These aspects will be considered further in sections 5 and 6.

3. Numerical methods

To advance the numerical solution in time we use the splitting approach introduced in [11]. That is, we first consider the free-streaming sub-flow

$$\partial_t f(t,x,v) + v \cdot \nabla_x f(t,x,v) = 0, \qquad f(0,x,v) = g(x,v). \tag{2}$$

The corresponding solution at time τ is then denoted by

$$\mathrm{e}^{\tau A} g(x,v) = f(\tau,x,v) \qquad \text{with} \qquad A = -v \cdot \nabla_x.$$

For the second part of the splitting algorithm we consider the acceleration sub-flow

$$\partial_t f(t,x,v) + E(x) \cdot \nabla_v f(t,x,v) = 0, \qquad f(0,x,v) = g(x,v), \tag{3}$$

where the electric field E is determined from the (charge) density $\rho(x) = \int g(x,v)\,\mathrm{d}v$ and is thus taken constant during that step. This property actually follows from equation (3) as the translation is only in the v-direction and thus leaves the density invariant. We denote the corresponding solution by

$$\mathrm{e}^{\tau B} g(x,v) = f(\tau,x,v) \qquad \text{with} \qquad B = -E(x) \cdot \nabla_v.$$

Using this notation we can easily formulate a time step of the splitting algorithm. In all our numerical results we will use the second-order Strang splitting given by

$$f^{m+1} = \mathrm{e}^{\frac{\tau}{2}A} \mathrm{e}^{\tau B} \mathrm{e}^{\frac{\tau}{2}A} f^m,$$

where $f^m(x,v)$ is an approximation of $f(t_m,x,v)$ and $\tau = t_{m+1} - t_m$ is the time step size. Let us note, however, that higher order splitting methods have been constructed as well (see, for example, [14]).

The main computational advantage of the splitting scheme is that the resulting sub-flows, given by equations (2) and (3), are in the form of an advection equation, where the advection speed is independent of the variables being advected. Thus, we can immediately solve equation (2) to obtain

$$\mathrm{e}^{\tau A} g(x,v) = g(x - \tau v, v)$$

ACCEPTED MANUSCRIPT

and solve equation (3) to obtain

$$\mathrm{e}^{\tau B} g(x,v) = g(x, v - \tau E(x)).$$

Thus, by using the splitting approach outlined above we have reduced the task of computing a numerical approximation to the four-dimensional Vlasov--Poisson equation to computing two-dimensional translations in phase space. However, this can be further simplified by splitting equation (2) into an advection in the x_1 direction

$$\partial_t f(t,x,v) + v_1 \partial_{x_1} f(t,x,v) = 0, \qquad f(0,x,v) = g(x,v)$$

and an advection in the x_2 direction

$$\partial_t f(t,x,v) + v_2 \partial_{x_2} f(t,x,v) = 0, \qquad f(0,x,v) = g(x,v),$$

where we have used ∂_{x_1} and ∂_{x_2} to denote the derivative with respect to x_1 and x_2, respectively. Let us denote the corresponding flows at time τ by $\mathrm{e}^{\tau A_1} g(x,v)$ and $\mathrm{e}^{\tau A_2} g$, respectively. Then since $[v_1 \partial_{x_1}, v_2 \partial_{x_2}] = 0$, i.e. the two operators commute, we have

$$e^{\tau A_2} \mathrm{e}^{\tau A_1} g = e^{\tau A} g$$

and no further splitting error is incurred. A very similar procedure can be carried out for the acceleration sub-flow. Thus, the four-dimensional Vlasov–Poisson equation has been reduced to a sequence of one-dimensional advections. This has significant advantages from a computational point of view and, in particular, implies that we only have to formulate our semi-Lagrangian scheme in a single dimension. We should remark that this approach can be extended to more complicated models, such as the Vlasov–Maxwell equations [13].

We will now describe the two competing approaches considered in this paper. First, for the semi-Lagrangian method based on spline interpolation an equidistant grid is used. However, for any grid point $(x_{1i}, x_{2j}, v_{1k}, v_{2l})$ the translations $x_{1i} - \tau v_{1k}$, $v_{1k} - \tau E_1(x_{1i}, x_{2j})$, etc., in general, do not coincide with a grid point. Here E_1 denotes the first component of the electric field. Thus, we have to use an interpolation scheme in order to evaluate $g(x_{1i} - \tau v_{1k}, x_{2j}, v_{1k}, v_{2l})$, $g(x_{1i}, x_{2j}, v_{1k} - \tau E_1(x_{1i}, x_{2j}), v_{2l})$. This is done by cubic spline interpolation. Note that only splines in one space dimension have to be constructed. This is essential for the computational efficiency of the scheme. Further computational ramifications are discussed in more detail in section 4. For more details we refer to [11, 54].

Second, the semi-Lagrangian discontinuous Galerkin scheme divides the domain into cells $C_{ijkl} = I_i^{(1)} \times I_j^{(2)} \times I_k^{(3)} \times I_l^{(4)}$, where the $I_i^{(1)}, I_j^{(2)}, I_k^{(3)}, I_l^{(4)}$ are one-dimensional intervals of length h. We further assume that a function g is given such that $g|_{C_i}$, i.e. the restriction of g to the ith cell, is a polynomial of degree ℓ. Then the function g lies in the approximation space. Note that we do not enforce a continuity constraint across cell interfaces. However, in general, this is not true for the translated function (for the other three translations we proceed in exactly the same way)

$$(T_\tau g)(x_1, x_2, v_1, v_2) = g(x_1 - \tau v_1, x_2, v_1, v_2).$$

ACCEPTED MANUSCRIPT

Thus, we perform an approximation by applying a projection operator P and obtain $PT_\tau g$. In order to turn this into a sensible numerical scheme, it must hold that $Pf|_{C_{ijkl}}$ is a polynomial of degree ℓ for any function f. Then $PT_\tau g$ constitutes the sought-after approximation of $g(x_1 - \tau v_1, x_2, v_1, v_2)$. In the present case the operator P is the L^2 projection on the (finite dimensional) subspace of cell-wise polynomials of degree ℓ. Note that this projection is the only approximation made. There is some freedom in selecting appropriate degrees of freedom for this method. Two obvious choices are to use the coefficients in a Legendre expansion or the value of function evaluations at the Gauss–Legendre nodes. In our implementation we use the latter. The value at the beginning of each time step is (per assumption) a cell-wise polynomial of degree ℓ. Translating this function still results in a piecewise polynomial of degree ℓ. We can then perform a (piecewise) Gauss–Legendre quadrature to obtain the coefficients required to implement the projection. Note that this is an analytic result as a Gauss–Legendre quadrature of appropriate order integrates polynomials of degree ℓ exactly. A detailed derivation and the obtained formulas are given in [23]. The performance implications will be discussed in section 4.

4. Computational consideration and performance results

The goal of this section is to discuss the performance of the cubic spline and discontinuous Galerkin based semi-Lagrangian methods on modern computer architectures. We will do this both from a theoretical and a practical (i.e. measuring specific implementations of these algorithms) point of view.

As has been discussed in the previous section, only a set of one-dimensional advection equations has to be solved for both algorithms. Thus, to simplify the notation, we only consider the one-dimensional advection

$$u^{m+1}(x) = u^m(x - v\tau) \tag{4}$$

in this section. This is, in fact, exactly how the numerical algorithm is implemented on a computer. A function is provided that solves the one-dimensional advection given by equation (4). This function is then executed multiple times with different parameters.

For spline interpolation, the function values $u_i^m = u^m(x_i)$ are given at the grid points x_i. The index i runs from 0 to $n - 1$, where n is the number of degrees of freedom. From this a spline that covers the entire domain has to be constructed. Since it would be wasteful to represent a spline as a collection of piecewise polynomials (as most of the coefficients of these polynomials are fixed by the continuity constraints), a B-spline basis is used instead. That is, the spline interpolation $\tilde{u}^m$ of u_i^m is represented as

$$\tilde{u}^m(x) = \sum_k \omega_k S(x - x_k),$$

ACCEPTED MANUSCRIPT

where S is given by

$$6S(x) = \begin{cases} 4 - 6(x/h)^2 + 3|x/h|^3 & 0 \leq |x| \leq h \\ (2 - |x/h|)^3 & h \leq |x| \leq 2h \\ 0 & \text{otherwise} \end{cases}$$

and h is the grid spacing. The coefficients ω_k are uniquely determined by the function values and can be obtained by solving the following linear system of equations (assuming periodic boundary conditions)

$$\frac{1}{6}\begin{bmatrix} 4 & 1 & 0 & \cdots & 0 & 1 \\ 1 & 4 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & 1 & 4 & 1 \\ 1 & 0 & \cdots & 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} \omega_0 \\ \omega_1 \\ \vdots \\ \vdots \\ \vdots \\ \omega_{n-1} \end{bmatrix} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ \vdots \\ \vdots \\ u_{n-1} \end{bmatrix}.$$

Note that the matrix is tridiagonal and symmetric and thus efficient methods exist to solve this system. For example, using the Thomas algorithm requires $\mathcal{O}(n)$ operations. Once the spline is constructed it is used to solve equation (4) as follows

$$u_i^{m+1} = \tilde{u}^m(x_i - v\tau).$$

Thus, the numerical algorithm proceeds in two steps. First, the spline is constructed by solving a linear system and this intermediate representation is then used to evaluate the numerical solution at the grid points. Both of these steps require $\mathcal{O}(n)$ memory and arithmetic operations, where n is the number of degrees of freedom. In the construction step this is due to the fact that linear complexity solvers are available for tridiagonal systems, such as the Thomas algorithm, and for the evaluation step this is true since the support of S is compact and independent of n.

On modern computer hardware both of these steps are memory bound. That is, performance is dictated by how fast data can be read and written to memory, as opposed to how many arithmetic operations the computer can perform. In the present case we have to at least read all u_i^m and write all u_i^{m+1}. In addition, we have to write and read all ω_k at least once. The latter is more subtle than one might expect. Since in a sequential implementation only one one-dimensional array of ω_ks needs to exist at any given time, this array could be kept in cache and then reads and writes to it would not count as a memory operation in the sense that those do not negatively impact performance. Note, however, that this is more challenging in a parallel implementation. For CPU systems this is usually not an issue. However, for GPUs with their high degree of parallelism and limited cache sizes this represents a severe obstacle to obtaining good performance. In this context we note that, to our knowledge, no results for cubic spline interpolation on GPUs have been published.

ACCEPTED MANUSCRIPT

For the semi-Lagrangian discontinuous Galerkin scheme the degrees of freedom are given by function values $u_{ij}^m \approx u^m(x_{i-1/2} + \xi_j)$, where i is the cell-index, which runs from 0 to n_C, $x_{i-1/2}$ the left cell interface, and ξ_j is the jth Gauss–Legendre node scaled to the interval $[0, h]$. The indices j run from 0 to ℓ, where ℓ is the degree of the polynomial approximant. The resulting scheme has order $\ell + 1$ and we will henceforth denote it as $\mathrm{dG}(\ell + 1)$. It is clear that to perform the L^2 projection on the ith cell only data from at most two adjacent cells is required. This is illustrated in Figure 2. We denote the index of these two cells by $i^\star$ and $i^\star + 1$, respectively. Since the projection is a linear operator we can write the resulting numerical scheme as follows

$$u_{ij}^{m+1} = \sum_l A_{jl} u_{i^\star l}^m + \sum_j B_{jl} u_{i^\star+1;l}^m, \tag{5}$$

where $A \in \mathbb{R}^{(\ell+1)\times(\ell+1)}$ and $B \in \mathbb{R}^{(\ell+1)\times(\ell+1)}$ are matrices that only depend on $v\tau$. The matrices can be precomputed at the beginning of each time step and thus their cost will not be a major concern except for extremely coarse space discretizations. The derivation of their exact form is somewhat lengthy and we thus refer the interested reader to [23].

For the implementation equation (5) is employed. The size of the matrices is negligible in comparison to the cache size and we therefore do not have to count reads and writes to them as memory accesses. This is true on both CPU and GPU architectures. The present algorithm has to read all entries in u_{ij}^m and write to all entries in u_{ij}^{m+1} ; it thus requires $\mathcal{O}(n)$ memory operations, where $n = n_C(\ell + 1)$ is the number of degrees of freedom. Technically, the algorithm requires $\mathcal{O}(\ell n)$ arithmetic operations and thus does not scale linearly in the degrees of freedom. However, since for reasonable values of ℓ we still have an extremely memory bound problem, this is of no consequence. That is, performance is dictated by the memory accesses and not by amount of arithmetic operations that need to be performed.

In an idealized setting both algorithms could achieve very similar levels of performance as both require us to read u^m once and write to u^{m+1} once. Of course, achieving this in an actual implementation is not a trivial task. However, it is our believe that the local nature of the discontinuous Galerkin approach simplifies this task. This is particularly true for GPU based systems, where a large number of threads (fine grained parallelism) is required to obtain optimal performance. In the following we will compare the performance of two Vlasov solvers. SeLaLib [1] is a library that facilitate kinetic simulations based on the semi-Lagrangian approach. It has been used extensively (see, for example, [15, 38, 4]). The library is written in Fortran and provides an efficient implementation of the cubic spline based semi-Lagrangian method. It should be noted, however, that SeLaLib also includes other interpolation schemes, such as Lagrange and Hermite interpolation. All numerical simulations conducted in this paper are performed using version `14ca223`. SeLaLib is compared with our own implementation (called SLDG) of the semi-Lagrangian discontinuous Galerkin approach. The code is written in C++ and due to the use of tem-

ACCEPTED MANUSCRIPT

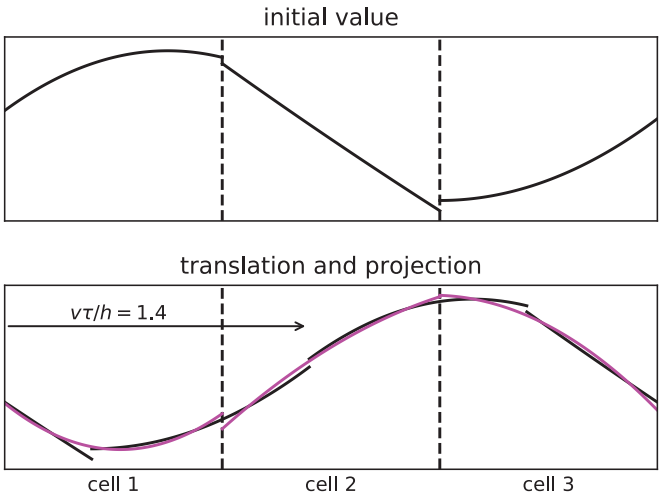


Figure 2: Illustration of the semi-Lagrangian discontinuous Galerkin scheme. The piecewise polynomial function of degree ℓ (top picture) is translated by $v\tau$ (bottom picture). The translated function is then projected back to the approximation subspace, which requires data from at most two adjacent cells.

plates facilitates implementations that work in arbitrary dimensions, while still maintaining good performance. This architecture is described in [23] and the source code can be found at `https://bitbucket.org/leinkemmer/sldg`. All numerical simulations conducted in this paper have been performed using version 0b6a4a4. In addition to solving the Vlasov equation, SLDG has been used in non-traditional applications of semi-Lagrangian methods. For example, it has been applied to the the Kadomtsev–Petviashvili equation, which requires the solution of a nonlinear advection [29].

Another major difference in the implementation of these two codes is how data is presented to the routines that perform the one-dimensional advections. For SeLaLib the corresponding arrays are always contiguous in memory. That is, for the advection in the x_1-direction the elements in the x_1 direction are laid out sequentially in memory, for the advection in the x_2-direction the elements in the x_2 direction are laid out sequentially in memory, etc. This requires that the four dimensional array is converted (i.e. transposed) to the appropriate form for each advection. This approach has the advantage that the same code can be used for all advection steps and cache locality is ensured almost automatically. It also fits very well with spline interpolation, as the global algorithm for constructing the spline is pointed at one contiguous array of memory. In fact, the approach is commonly used in implementations that employ cubic spline interpolation. For example, it is also used in the Gysela code [35]. The disadvantage of this approach is that the transpositions do not have a very favorable memory access pattern and thus incur a significant computational cost. Although it is possible to obtain scaling to a large number of cores, the transpose can become a limiting

ACCEPTED MANUSCRIPT

factor in that context as well (see, for example, [7, 52]). In this context, OpenMP on the node and MPI between nodes, i.e. a hybrid parallelization strategy, can be used to increase the scalability of the transposition.

On the other hand, the semi-Lagrangian discontinuous Galerkin implementation uses the ability of abstractions in C++ to hide the actual memory layout from the function that performs the advection. Thus, it is still possible to write the one-dimensional advection code as if the arrays are contiguous, even though the actual memory layout is more complicated. As a consequence, no transpositions have to be carried out in this implementation. The main concern here is that cache locality has to be taken into account. However, the local nature of the discontinuous Galerkin approach makes this less problematic. To further alleviate this issue we use cache blocking for the dimension with large strides.

We have measured both the performance and the amount of memory used for these two codes. The results are shown in Table 1. For both methods one dual socket node of the described computer systems is used. Both codes are run using 16 parallel threads. We observe that the discontinuous Galerkin implementation is both faster (by approximately a factor of 2 on the newer Haswell CPUs and approximately a factor of 1.7 on the Ivy Bridge CPUs) and uses significantly less memory (by approximately a factor of 3), if the same number of degrees of freedom are used. Let us note that the disadvantage in memory consumption observed for SeLaLib is not an intrinsic property of the cubic spline algorithm. Thus, an improved implementation would likely show a similar behavior with respect to memory usage as does SLDG.

In addition, results for simulation on a GPU system are shown in Table 1. We observe a speedup by approximately a factor of 5 compared to the SLDG CPU implementation and a speedup by approximately a factor of 10 compared to the SeLaLib cubic spline interpolation. This is mainly due to increased memory bandwidth on the GPU and, as has been discussed before, the fact that the SLDG algorithm is particularly well suited to exploit the fine grained parallelism needed on that platform.

Both codes support parallelization using MPI. Note that parallelization is essential even to obtain good performance on a single node (i.e. for the results presented in Table 1). This is due to the fact that, although we have a memory bound problem, a single core is not sufficient to saturate the memory bandwidth. However, both codes can also be run on distributed memory clusters (supercomputers). In this case the issue of how to distribute the computational domain to the different MPI processes becomes important. Although this choice of data distribution is identical for both codes it is instructive to discuss it. An important consideration here is the method used to solve Poisson's equation. In both codes this in done using an algorithm based on the fast Fourier transform. While the cost of the FFT is negligible, as this is only a 2 dimensional problem, it still has consequences with regard to the data distribution. The main issue here is that most FFT libraries only allow a so-called pencil decomposition. This is, in particular, true for FFTW, which is used in both implementations. Thus it is not straightforward to parallelize the Fourier transforms without incurring a penalty associated with additional data transfer. Both codes avoid this problem

ACCEPTED MANUSCRIPT

Workstation

	Time per step (time per dof, bandwidth)		Memory used	
	dof=64	dof=128	dof=64	dof=128
SeLaLib	0.194 s (12 ns, 11 GB/s)	3.25 s (12 ns, 11 GB/s)	0.85 GB	13.0 GB
SLDG (dG4)	0.105 s (6 ns, 20 GB/s)	1.62 s (6 ns, 21 GB/s)	0.28 GB	4.2 GB

VSC3

	Time per step (time per dof, bandwidth)	
	dof=64	dof=128
SeLaLib	0.219 s (13 ns, 10 GB/s)	4.02 s (15 ns, 9 GB/s)
SLDG (dG4)	0.151 s (9 ns, 14 GB/s)	2.31 s (9 ns, 15 GB/s)

GPU (K80)

	Time per step (time per dof, bandwidth)	
	dof=64	dof=128
SLDG (dG4)	0.042 s (2.5 ns, 51 GB/s)	0.35 s (1.3 ns, 98 GB/s)

Table 1: Comparison of performance and memory usage between SeLaLib (cubic spline interpolation) and SLDG (fourth order semi-Lagrangian discontinuous Galerkin method). For both codes the nonlinear Landau damping initial value of section 5 is used and 64 and 128 degrees of freedom (dof) per direction are considered. The time required per degree of freedom per time step `and the achieved memory bandwidth` are shown in parenthesis. The Workstation consists of two Intel Xeon E5-2630 v3 CPUs and 32 GB of DDR4 memory `and has a theoretical memory bandwidth of 59 GB/s`. One node of the VSC3 consists of two Intel Xeon E5-2650v2 and 64GB of DDR3 memory. The memory consumption is measured using the `/usr/bin/time` utility. For the GPU runs one graphic card (GK210 chip) of the K80 is used. The theoretical memory bandwidth of that system is 240 GB/s. For all simulations a Strang splitting procedure with time step size $\tau = 0.1$ has been used.

ACCEPTED MANUSCRIPT

by only parallelizing in the velocity directions. Then, the Fourier transforms can be conducted independently on each of the processes. For four/six dimensional simulations there are still two/three dimensions available for parallelization. The main difference in the two codes is that while SLDG only performs halo communication (due to the local nature of the semi-Lagrangian discontinuous Galerkin scheme), SeLaLib requires two calls to `MPI_Alltoall` in each time step (due to the global data dependency inherent in constructing the spline).

We now investigate the scaling of both codes on the Vienna Scientific Cluster 3 (VSC3). For more information and details on the hardware of that system we refer the reader to `http://vsc.ac.at/systems/vsc-3/`. On the left hand side of Figure 3 we present strong scaling results (i.e. the problem size is kept constant as the number of cores is increased) for the problem considered in section 5. We observe that the discontinuous Galerkin scheme maintains its advantage independently of the number of cores the program is run on. That is, both codes show a similar behavior with respect to scaling. On the right hand side of Figure 3 we present weak scaling results (i.e. the problem size increases proportionally to the number of cores). In this case we observe that SLDG significantly outperforms SeLaLib. This is, in particular, true as we consider more than 256 cores. It is interesting to investigate the reason for this discrepancy in performance. To that end we have subtracted the time required for the calls to `MPI_Alltoall,` which are used to map the velocity grid into contiguous memory on each process, in the SeLaLib code from the total run time. Note that even on a single node this operation requires a significant amount of time, as computing the transpose is a memory intensive operation. The resulting line is very similar to the run time of the SLDG code. That is, almost all of the difference in performance between the two codes is explained by the global communication required for the spline interpolation.

5. Nonlinear Landau damping

In this section we consider nonlinear Landau damping in four dimensions. That is, we impose the initial value

$$f(0, x_1, x_2, v_1, v_2) = \frac{1}{2\pi} \left(1 + \epsilon(\cos(kx_1) + \cos(kx_2))\right) \mathrm{e}^{-(v_1^2+v_2^2)/2} \tag{6}$$

on the domain $[0, 4\pi]^2 \times [-6, 6]^2$. We set $\epsilon = \frac{1}{2}$ and $k = 0.5$. This is precisely the setup used in [32]. Strang splitting with step size $\tau = 0.1$ is used for the time integration. We are interested in resolving the dynamics of nonlinear Landau damping up to $t = 50$. Note that, as has been discussed in section 2, the error in the density function f is not a viable indicator for the accuracy of the numerical method. In accordance with the literature we will mainly focus on the time evolution of the electric energy

$$\frac{1}{2} \int E(t, x)^2 \, \mathrm{d}x.$$

ACCEPTED MANUSCRIPT

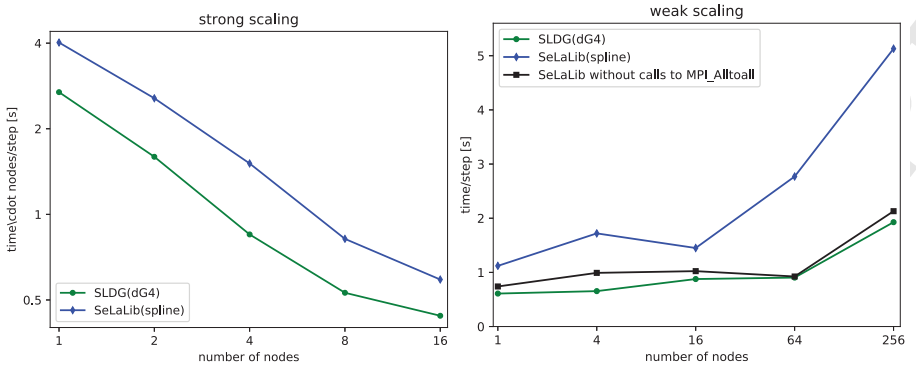


Figure 3: Strong and weak scaling for SeLaLib (cubic spline interpolation) and SLDG (fourth order semi-Lagrangian discontinuous Galerkin method). For strong scaling the nonlinear Landau damping initial value of section 5 with 128 degrees of freedom per direction is used. For weak scaling $64^2 \times (\sqrt{n} \cdot 128)^2$ degrees of freedom, where n is the number of nodes employed, are used. The simulations are conducted on the VSC3, each node of which consists of two Intel Xeon E5-2650v2 and 64GB of DDR3 memory. For all simulations a Strang splitting procedure with time step size $\tau = 0.1$ has been used.

This allows us to evaluate if the numerical method faithfully resolves the physics of this system. More precisely, the characteristic growth and decay of the electric energy should be indistinguishable (in a plot) from the exact solution until $t = 50$. To accomplish this for spline interpolation we need 128 grid points per direction (see Figure 1). Note that despite the fact that, as the results in Figure 1 show, the error in the density function is on the order of unity, the physically important quantity is resolved very well. Therefore, there is little advantage of running the integrator in the asymptotic regime, which would not be feasible in four dimensions in any case.

In the introduction second, fourth, and sixth order discontinuous Galerkin schemes were considered. It was already clear at that point that the second order scheme is not competitive. However, the order plots (for $t = 10$) presented there, seem to indicate that the sixth order discontinuous Galerkin approach performs best. However, this is misleading as numerical simulations have shown that for $t = 50$ the fourth order method should actually be favored. Therefore, in this section we will exclusively use dG4. Before proceeding, let us note, however, that for long time integration this is not necessarily the case. For example [24] found by performing numerical simulations in two dimensions that for long times the sixth order method can be significantly more efficient.

If we run the same simulation for the discontinuous Galerkin approach we see that the solution is somewhat worse. In particular, between $t = 40$ and $t = 50$ the electric energy is slightly increased compared to the reference solution. In order to remedy this we can increase the number of degrees of freedom per direction to 144 (corresponding to $\alpha = 1.125$). This means that in four dimensions 60% more degrees of freedom are required by the discontinuous Galerkin approach to achieve the same accuracy. However, as we have seen in section 4,

ACCEPTED MANUSCRIPT

the implementation of the discontinuous Galerkin scheme is also more efficient (by a factor of 2 on Haswell CPUs) and uses less memory (by a factor of 3). Thus, the discontinuous Galerkin scheme retains an advantage in performance (approximately 25%). In addition, the amount of memory required is reduced by a factor of 2.

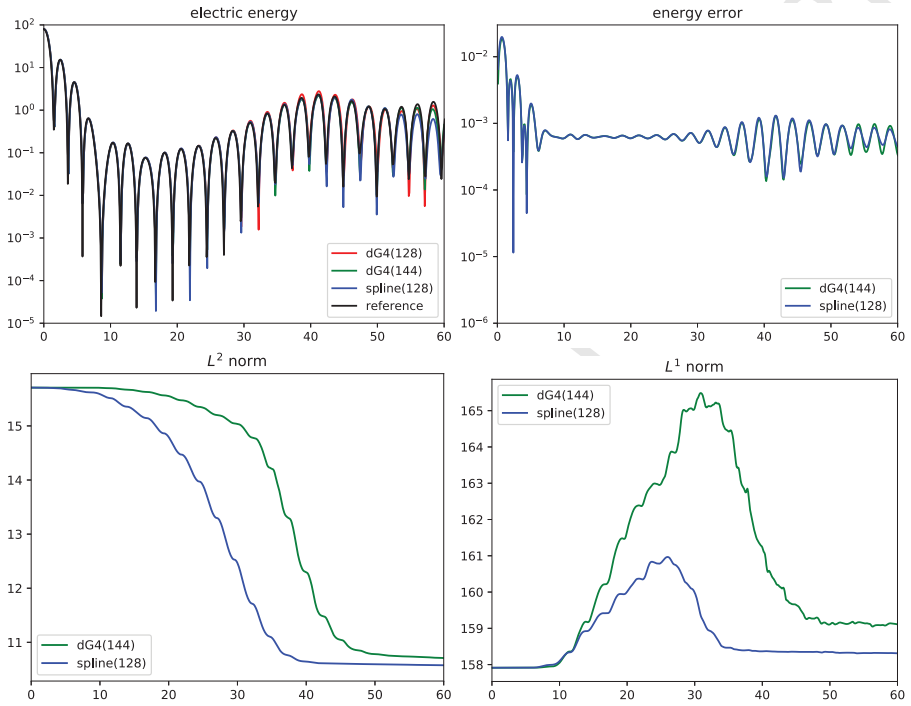


Figure 4: The electric energy (top left), the energy error (top right), the L^2 norm (bottom left), and the L^1 norm (bottom right) are shown as a function of time for the nonlinear Landau damping problem given in equation (6). The reference solution is computed using spline interpolation and 512 grid points (per direction). For all simulations a Strang splitting procedure with time step size $\tau = 0.1$ has been used.

In addition, Figure 4 shows the important physical invariants of energy, L^2 norm, and L^1 norm. Although the main concern for the nonlinear Landau damping is the time evolution of the electric energy, these invariants give additional diagnostics of how well physical properties of the exact solution are respected by a numerical method.

The error in energy is almost identical for both methods and is on the order of 10^{-3} (after a transient phase). This is due to the fact that the energy error for the present problem is dominated by the time integration error.

For collisionless plasma simulations the L^2 error is an important diagnostic. A decrease in the L^2 norm implies that our numerical scheme adds numerical diffusion to the simulation. Although most numerical methods introduce a certain amount of diffusion, it is clearly preferable to limit this as much as possible.

ACCEPTED MANUSCRIPT

For nonlinear Landau damping the discontinuous Galerkin method outperforms spline interpolation in this metric (in particular, between $t = 10$ and $t = 40$).

Finally, we look at the L^1 norm. For the exact solution mass and L^1 norm are equal in value and the L^1 norm is thus conserved. However, if a mass conservative numerical scheme commits a numerical error that introduces negative values, the result is a change in the L^1 norm. This is, in particular, a problem for schemes that are susceptible to oscillations (such as spectral methods). However, most numerical methods for the Vlasov equation experience this behavior to some extent. Let us emphasize that, contrary to say the Euler equations, this neither leads to numerical instabilities nor necessarily diminishes averaged quantities (such as the electric energy). However, since negative values make the physical interpretation of the density function difficult, it is still desirable to limit this behavior as much as possible. For the nonlinear Landau damping the discontinuous Galerkin scheme shows worse performance with respect to L^1 conservation compared to spline interpolation.

Overall we conclude that the discontinuous Galerkin implementation has a moderate advantage with respect to performance and L^2 conservation as well as a significant advantage with respect to memory consumption. On the other hand, spline interpolation has a moderate advantage with respect to L^1 conservation. If we assume that the present results generalize to the six-dimensional case (i.e. $\alpha = 1.125$ as above), then the performance of both methods would be evenly matched. This is due to the fact that $1.125^6 \approx 2$ more degrees of freedom would be required for the discontinuous Galerkin scheme, which exactly cancels the factor of 2 advantage the corresponding implementation provides.

6. Two-stream instability

In this section we consider the two-stream instability in four dimensions. That is, we impose the initial value

$$f(0, x_1, x_2, v_1, v_2) = (1 + \epsilon \cos(kx_1) \cos(kx_2)) f^{\mathrm{eq}}(v_1, v_2) \tag{7}$$

with

$$f^{\mathrm{eq}} = \frac{1}{8\pi} \left(\mathrm{e}^{-(v_1 - v_0)^2/2} + \mathrm{e}^{-(v_1 + v_0)^2/2} \right) \left(\mathrm{e}^{-(v_2 - v_0)^2/2} + \mathrm{e}^{-(v_2 + v_0)^2/2} \right),$$

where $\epsilon = 10^{-3}$, $k = 0.2$, and $v_0 = 2.4$. The computational domain is $[0, 10\pi]^2 \times [-6, 6]^2$. Strang splitting with step size $\tau = 0.1$ is used for the time integration.

This is an interesting problem as the electric energy is small and even decreases by three orders of magnitude compared to the initial value during a quiescent phase (from $t = 0$ to approximately $t = 175$). However, eventually a physical instability develops that results in an exponential increase of the electric energy (approximately $t = 175$ to $t = 225$). This is followed by nonlinear saturation. In this nonlinear phase the electric energy oscillates around a constant value with an amplitude that decreases in time.

ACCEPTED MANUSCRIPT

It turns out that to obtain agreement between a numerical and the exact solution for the quiescent phase is challenging. In fact, simulations conducted for a similar two-dimensional problem indicate that for spline interpolation this is only achieved for 1024 grid points per direction, which is extremely expensive. Nevertheless, simulations with fewer degrees of freedom are still able to give pertinent information on the plasma instability. In particular, at which time the instability occurs, the growth rate of the instability, and the behavior of the electric energy in the nonlinear regime (i.e. after saturation).

Figure 5 shows a comparison of the fourth order discontinuous Galerkin method and spline interpolation for 32, 64, and 128 degrees of freedom per direction. For the dG4 method already 32 degrees of freedom are sufficient in order to accurately predict the time and growth of the instability. On the other hand, this is obviously not true for spline interpolation, which shows an instability that is too late and does not reach the peak for the electric energy. While increasing the number of degrees of freedom for spline interpolation results in significant improvements with respect to these metrics, the discontinuous Galerkin approach always retains an advantage in the nonlinear phase. More specifically, spline interpolation exhibits a decay of the electric energy that is not present in the exact solution.

This behavior can be explained by looking at the time evolution of the L^2 norm (see Figure 5). We observe that the L^2 norm for dG4 is significantly better conserved compared to spline interpolation once we enter the nonlinear phase. Thus, more diffusion is introduced by spline interpolation which results in the corresponding unphysical decrease in electric energy that is observed (and the absence of the same artefact for the discontinuous Galerkin approach). For the two-dimensional case this behavior has already been discussed in [24].

For the present example, the difference between performance in the asymptotic regime (valid only for small times) and performance for time scales of physical interest, can be highlighted very clearly. In fact, if we look at the plots in Figure 5, we see that in the quiescent phase, the numerical solution obtained with spline interpolation follows the exact solution somewhat longer. However, even for 128 degrees of freedom agreement is only achieved up to approximately $t = 30$ (for spline interpolation). Nevertheless, the dG4 method is clearly superior in terms of both predicting the onset of the instability and in providing qualitatively correct predictions for the nonlinear phase. That is, in obtaining the physics of interest.

Let us further remark that the L^1 norm is significantly better conserved by the discontinuous Galerkin approach compared to spline interpolation. Both methods conserve energy quite well (at least up to 10^{-3}). Although for the present simulation there is some advantage for spline interpolation.

We conclude that for the two-stream instability considered here, the discontinuous Galerkin approach is clearly the superior choice compare to spline interpolation.

ACCEPTED MANUSCRIPT

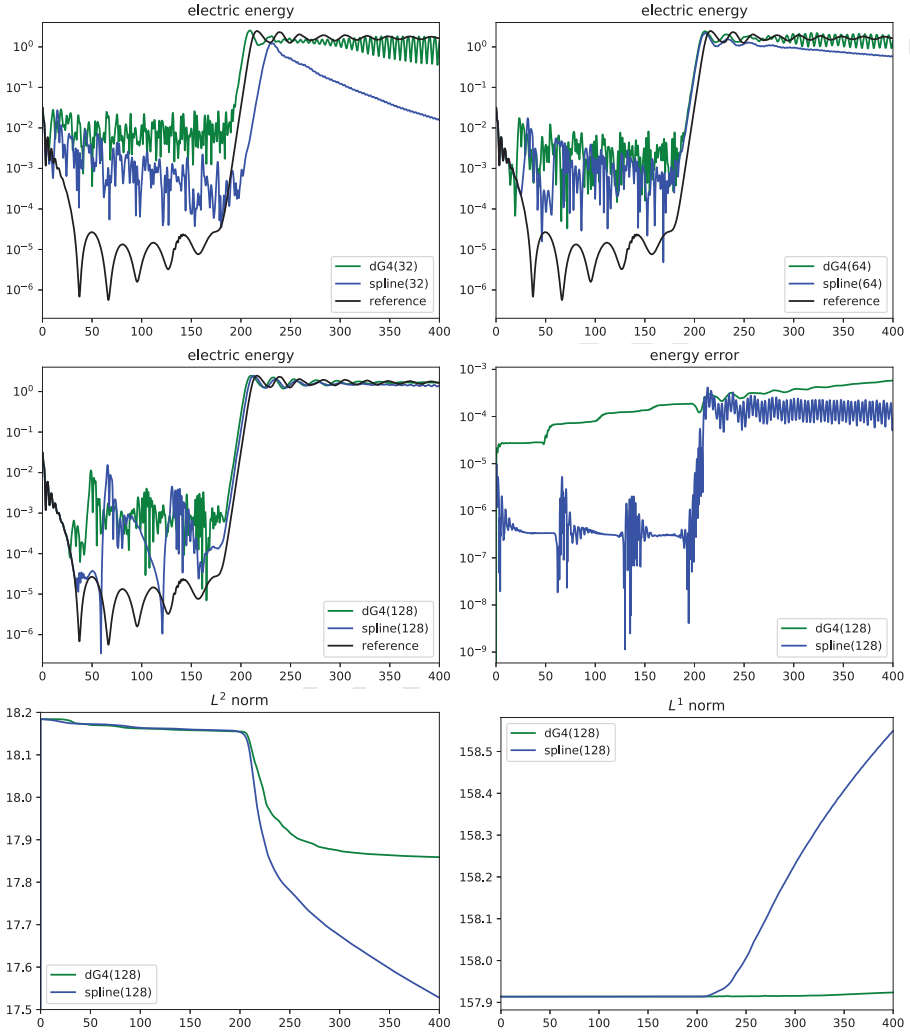


Figure 5: The time evolution of the electric energy for the two-stream instability given by equation (7) is shown for 32 (top left), 64 (top right), and 128 (center left) degrees of freedom. In addition, the energy error (center right), the L^2 norm (bottom left), and the L^1 norm (bottom right) are shown as a function of time. The reference solution is computed using spline interpolation and 512 grid points (per direction). For all simulations a Strang splitting procedure with time step size $\tau = 0.1$ has been used.

ACCEPTED MANUSCRIPT

7. Conclusion

Two semi-Lagrangian schemes and their corresponding implementation have been compared. Namely, the semi-Lagrangian discontinuous Galerkin based SLDG code and the cubic spline based SeLaLib code. The comparison has been conducted in the context of four dimensional nonlinear Landau damping and a four dimensional two-stream instability. We find that the discontinuous Galerkin approach outperforms the cubic spline interpolation in both cases. In the case of nonlinear Landau damping the difference is rather modest, while for the two-stream instability a large difference in performance can be observed. It has to be emphasized that this true despite the fact that results in the asymptotic regime favor spline interpolation. This is explained by the improved structure preserving properties of the semi-Lagrangian discontinuous Galerkin approach, which are important even for short time simulations. We have further investigated the performance of these two codes on distributed memory clusters. The obtained results reveal that the local nature of the discontinuous Galerkin approach yields a substantial advantage in terms of performance for simulations that employ more than 256 cores. We thus conclude that the semi-Lagrangian discontinuous Galerkin scheme is a viable alternative to cubic spline interpolation for performing simulations of the Vlasov–Poisson equation, even in multiple dimensions. This is, in particular, true if these simulations are run on state of the art supercomputers or GPUs, as those necessitate a high degree of parallelism. Such considerations will, in all likelihood, become even more important on the next generation of high performance computing systems.

[1] SeLaLib. `http://selalib.gforge.inria.fr/`.

[2] T.D. Arber and R.G.L. Vann. A critical comparison of Eulerian-grid-based Vlasov solvers. *J. Comput. Phys.*, 180(1):339–357, 2002.

[3] J.W. Banks and J.A.F. Hittinger. A new class of nonlinear finite-volume methods for Vlasov simulation. *IEEE Trans. Plasma Sci. IEEE*, 38(9):2198–2207, 2010.

[4] Y. Barsamian, J. Bernier, S. Hirstoaga, and M. Mehrenberger. Verification of 2Dx2D and two-species Vlasov-Poisson solvers. *preprint, hal-01668744*, 2017.

[5] N. Besse. Convergence of a semi-Lagrangian scheme for the one-dimensional Vlasov–Poisson system. *SIAM J. on Numer. Anal.*, 42(1):350–382, 2004.

[6] N. Besse and M. Mehrenberger. Convergence of classes of high-order semi-Lagrangian schemes for the Vlasov–Poisson system. *Math. Comput.*, 77(261):93–123, 2008.

[7] J. Bigot, V. Grandgirard, G. Latu, C. Passeron, F. Rozar, and O. Thomine. Scaling Gysela code beyond 32K-cores on BlueGene/Q. In *ESAIM: Proceedings*, volume 43, pages 117–135. EDP Sciences, 2013.

ACCEPTED MANUSCRIPT

[8] X. Cai, W. Guo, and J. Qiu. A high order conservative semi-Lagrangian discontinuous Galerkin method for two-dimensional transport simulations. *J. Sci. Comput.*, 73(2):514–542, 2017.

[9] X. Cai, W. Guo, and J. Qiu. A high order semi-Lagrangian discontinuous Galerkin method for Vlasov–Poisson simulations without operator splitting. *J. Comput. Phys.*, 354:529–551, 2018.

[10] F. Califano, L. Galeotti, and A. Mangeney. The Vlasov-Poisson model and the validity of a numerical approach. *Phys. Plasmas*, 13(8):082102, 2006.

[11] C. Cheng and G. Knorr. The integration of the Vlasov equation in configuration space. *J. Comput. Phys.*, 22(3):330–351, 1976.

[12] Y. Cheng, I.M. Gamba, F. Li, and P.J. Morrison. Discontinuous Galerkin Methods for the Vlasov–Maxwell Equations. *SIAM J. Numer. Anal.*, 52(2):1017–1049, 2014.

[13] N. Crouseilles, L. Einkemmer, and E. Faou. Hamiltonian splitting for the Vlasov–Maxwell equations. *J. Comput. Phys.*, 283:224–240, 2015.

[14] N. Crouseilles, E. Faou, and M. Mehrenberger. High order Runge-Kutta-Nyström splitting methods for the Vlasov-Poisson equation. *preprint, inria-00633934*, 2011.

[15] N. Crouseilles, P. Glanc, S.A. Hirstoaga, E. Madaule, M. Mehrenberger, and J. Pétri. A new fully two-dimensional conservative semi-Lagrangian method: applications on polar grids, from diocotron instability to ITG turbulence. *Eur. Phys. J. D*, 68(9):252, 2014.

[16] N. Crouseilles, P. Glanc, M. Mehrenberger, and C. Steiner. Finite volume schemes for Vlasov. In *ESAIM: Proceedings*, volume 38, pages 275–297, 2012.

[17] N. Crouseilles, G. Latu, and E. Sonnendrücker. A parallel Vlasov solver based on local cubic spline interpolation on patches. *J. Comput. Phys.*, 228(5):1429–1446, 2009.

[18] N. Crouseilles, M. Mehrenberger, and E. Sonnendrücker. Conservative semi-Lagrangian schemes for Vlasov equations. *J. Comput. Phys.*, 229(6):1927–1953, 2010.

[19] N. Crouseilles, M. Mehrenberger, and F. Vecil. Discontinuous Galerkin semi-Lagrangian method for Vlasov-Poisson. In *ESAIM: Proceedings*, volume 32, pages 211–230. EDP Sciences, 2011.

[20] N. Crouseilles, M. Mehrenberger, and F. Vecil. A Discontinuous Galerkin semi-Lagrangian solver for the guiding-center problem. *HAL preprint, hal–00717155*, 2012.

ACCEPTED MANUSCRIPT

[21] N. Crouseilles, T. Respaud, and E. Sonnendrücker. A forward semi-Lagrangian method for the numerical solution of the Vlasov equation. *Comput. Phys. Commun.*, 180(10):1730–1745, 2009.

[22] L. Einkemmer. A mixed precision semi-Lagrangian algorithm and its performance on accelerators. In *High Performance Computing and Simulation (HPCS), International Conference on*, 2016.

[23] L. Einkemmer. High performance computing aspects of a dimension independent semi-Lagrangian discontinuous Galerkin code. *Comput. Phys. Commun.*, 202:326–336, 2016.

[24] L. Einkemmer. A study on conserving invariants of the Vlasov equation in semi-Lagrangian computer simulations. *J. Plasma Phys.*, 83(2), 2017.

[25] L. Einkemmer and A. Ostermann. A strategy to suppress recurrence in grid-based Vlasov solvers. *Eur. Phys. J. D*, 68(7):197, 2014.

[26] L. Einkemmer and A. Ostermann. Convergence analysis of a discontinuous Galerkin/Strang splitting approximation for the Vlasov–Poisson equations. *SIAM J. Numer. Anal.*, 52(2):757–778, 2014.

[27] L. Einkemmer and A. Ostermann. Convergence analysis of Strang splitting for Vlasov-type equations. *SIAM J. Numer. Anal.*, 52(1):140–155, 2014.

[28] L. Einkemmer and A. Ostermann. On the error propagation of semi-Lagrange and Fourier methods for advection problems. *Comput. Math. Appl.*, 69(3):170–179, 2015.

[29] L. Einkemmer and A. Ostermann. A semi-Lagrangian discontinuous Galerkin scheme for Burgers' equation with application to the Kadomtsev–Petviashvili equation. *preprint, arXiv:1701.05602*, 2017.

[30] N.V. Elkina and J. Büchner. A new conservative unsplit method for the solution of the Vlasov equation. *J. Comput. Phys.*, 213(2):862–875, 2006.

[31] E. Fijalkow. Numerical solution to the Vlasov equation: The 1D code. *Comput. Phys. Commun.*, 116(2):329 – 335, 1999.

[32] F. Filbet and E. Sonnendrücker. Comparison of Eulerian Vlasov solvers. *Comput. Phys. Commun.*, 150(3):247–266, 2003.

[33] L. Galeotti and F. Califano. Asymptotic Evolution of Weakly Collisional Vlasov-Poisson Plasmas. *Phys. Rev. Lett.*, 95:015002, 2005.

[34] C.K. Garrett and C.D. Hauck. A Fast Solver for Implicit Integration of the Vlasov–Poisson System in the Eulerian Framework. *J. Sci. Comput.*, 40(2):B483–B506, 2018.

ACCEPTED MANUSCRIPT

[35] V. Grandgirard, M. Brunetti, P. Bertrand, N. Besse, X. Garbet, P. Ghendrih, G. Manfredi, Y. Sarazin, O. Sauter, E. Sonnendrücker, L. Vaclavik, and L. Villard. A drift-kinetic Semi-Lagrangian 4D code for ion turbulence simulation. *J. Comput. Phys.*, 217(2):395–423, 2006.

[36] Y. Güçlü, A.J. Christlieb, and W.N.G. Hitchon. Arbitrarily high order convected scheme solution of the Vlasov–Poisson system. *J. Comput. Phys.*, 270:711–752, 2014.

[37] Y. Güçlü and W.N.G. Hitchon. A high order cell-centered semi-Lagrangian scheme for multi-dimensional kinetic simulations of neutral gas flows. *J. Comput. Phys.*, 231(8):3289–3316, 2012.

[38] S. Guisset, M. Gutnic, P. Helluy, M. Massaro, L. Navoret, N. Pham, and M. Roberts. Lagrangian/Eulerian solvers and simulations for Vlasov-Poisson. *ESAIM: proceedings and surveys*, 53:120–132, 2016.

[39] W. Guo, R.D. Nair, and J. Qiu. A conservative semi-Lagrangian discontinuous Galerkin scheme on the cubed sphere. *Mon. Weather Rev.*, 142(1):457–475, 2014.

[40] E. Hairer, C. Lubich, and G. Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, volume 2nd edition. Springer, 2006.

[41] R.E. Heath, I.M. Gamba, P.J. Morrison, and C. Michler. A discontinuous Galerkin method for the Vlasov–Poisson system. *J. Comput. Phys.*, 231(4):1140–1174, 2012.

[42] J.A.F. Hittinger and J.W. Banks. Block-structured adaptive mesh refinement algorithms for Vlasov simulation. *J. Comput. Phys.*, 241:118–140, 2013.

[43] D. Jarema, H. Bungartz, T. Görler, F. Jenko, T. Neckel, and D. Told. Block-structured grids for Eulerian gyrokinetic simulations. *Comput. Phys. Commun.*, 198:105–117, 2016.

[44] D. Jarema, H. Bungartz, T. Görler, F. Jenko, T. Neckel, and D. Told. Block-structured grids in full velocity space for Eulerian gyrokinetic simulations. *Computer Physics Communications*, 215:49–62, 2017.

[45] C. Lehrenfeld. High order exactly divergence-free hybrid discontinuous Galerkin methods for unsteady incompressible flows. *Comput. Methods Appl. Mech. Eng.*, 307:339–361, 2016.

[46] A. Mangeney, F. Califano, C. Cavazzoni, and P. Travnicek. A numerical scheme for the integration of the Vlasov–Maxwell system of equations. *J. Comput. Phys.*, 179(2):495–538, 2002.

ACCEPTED MANUSCRIPT

[47] T. Nakamura and T. Yabe. Cubic interpolated propagation scheme for solving the hyper–dimensional Vlasov–Poisson equation in phase space. *Comput. Phys. Commun.*, 120(2-3):122–154, 1999.

[48] M. Palmroth, U. Ganse, Y. Pfau-Kempf, M. Battarbee, L. Turc, T. Brito, M. Grandin, S. Hoilijoki, A. Sandroos, , and S. von Alfthan. Vlasov methods in space physics and astrophysics. *Living Rev. Comput. Astrophys.*, 4(1), 2018.

[49] J.M. Qiu and C.W. Shu. Positivity preserving semi-Lagrangian discontinuous Galerkin formulation: theoretical analysis and application to the Vlasov–Poisson system. *J. Comput. Phys.*, 230(23):8386–8409, 2011.

[50] T. Respaud and E. Sonnendrücker. Analysis of a new class of forward semi-Lagrangian schemes for the 1D Vlasov–Poisson equations. *Numer. Math.*, 118(2):329–366, 2011.

[51] J.A. Rossmanith and D.C. Seal. A positivity-preserving high-order semi-Lagrangian discontinuous Galerkin scheme for the Vlasov–Poisson equations. *J. Comput. Phys.*, 230(16):6203–6232, 2011.

[52] Fabien Rozar, Guillaume Latu, and Jean Roman. Achieving memory scalability in the gysela code to fit exascale constraints. In *International Conference on Parallel Processing and Applied Mathematics*, pages 185–195. Springer, 2013.

[53] N.J. Sircombe and T.D. Arber. VALIS: A split-conservative scheme for the relativistic 2D Vlasov–Maxwell system. *J. Comput. Phys*, 228:4773–4788, 2009.

[54] E. Sonnendrücker, J. Roche, P. Bertrand, and A. Ghizzo. The semi-Lagrangian method for the numerical resolution of the Vlasov equation. *J. Comput. Phys.*, 149(2):201–220, 1999.

[55] C. Steiner, M. Mehrenberger, and D. Bouche. A semi-Lagrangian discontinuous Galerkin convergence. *preprint, hal-00852411*, 2013.

[56] J. P. Verboncoeur. Particle simulation of plasmas: review and advances. *Plasma Physics and Controlled Fusion*, 47(5A):A231, 2005.

[57] B.S. Wettervik, T.C. DuBois, E. Siminos, and T. Fülöp. Relativistic Vlasov–Maxwell modelling using finite volumes and adaptive mesh refinement. *Eur. Phys. J. D*, 71(6):157, 2017.

[58] M. Zerroukat, N. Wood, and A. Staniforth. A monotonic and positive-definite filter for a semi-Lagrangian inherently conserving and efficient (SLICE) scheme. *Q.J.R. Meteorol. Soc.*, 131(611):2923–2936, 2005.

[59] M. Zerroukat, N. Wood, and A. Staniforth. The parabolic spline method (PSM) for conservative transport problems. *Int. J. Numer. Meth. Fluids*, 51(11):1297–1318, 2006.