ESAIM: M2AN 48 (2014) 1029–1060
DOI: 10.1051/m2an/2014009

ESAIM: Mathematical Modelling and Numerical Analysis
www.esaim-m2an.org

HIGH ORDER SEMI-LAGRANGIAN PARTICLE METHODS FOR TRANSPORT EQUATIONS: NUMERICAL ANALYSIS AND IMPLEMENTATION ISSUES

G.-H. Cottet[1], J.-M. Etancelin[1], F. Perignon[1] and C. Picard[1]

Abstract. This paper is devoted to the definition, analysis and implementation of semi-Lagrangian methods as they result from particle methods combined with remeshing. We give a complete consistency analysis of these methods, based on the regularity and momentum properties of the remeshing kernels, and a stability analysis of a large class of second and fourth order methods. This analysis is supplemented by numerical illustrations. We also describe a general approach to implement these methods in the context of hybrid computing and investigate their performance on GPU processors as a function of their order of accuracy.

Mathematics Subject Classification. 65M12, 65M75, 65Y05, 65Y20.

Received October 22, 2013. Revised January 28, 2014.
Published online June 30, 2014.

1. Introduction

Particle methods are Lagrangian methods that have been designed for advection dominated problems, with applications mostly in plasma physics [10], incompressible flows [4,7,16], or gas dynamics [20]. Following [12,13], particle methods are often associated with remeshing, in order to maintain the regularity of the particle distribution and, as a consequence, to control the accuracy of the method in presence of strong strain in the flow carrying the particles. Remeshing removes the grid free nature of particle methods but maintains some of its features like conservation, locality, and stability. It not only guarantees accuracy but also allows to combine in a seamless fashion particle methods with grid-based techniques, like domain decomposition methods and fast FFT-based Poisson solvers for field evaluations [5,22], multi-resolution and adaptive mesh refinement [1,2].

In practice particle remeshing occurs every *few* time steps. In that case, and as long as the remeshing frequency does not increase when the discretization parameters tend to zero, Remeshed Particle Methods (RPM in the sequel) can be viewed as *conservative forward semi-lagrangian* methods. Classical semi-lagrangian methods for transport equations combine tracking of trajectories originating at grid points and interpolation from the grid. Because they operate on local point values, semi-lagrangian methods in general do not conserve mass, unless some specific treatment is done. By contrast, particle methods transport masses which makes them inherently conservative.

Keywords and phrases. Advection equations, particle methods, semi-Lagrangian methods, GPU computing.

[1] Université Grenoble Alpes and CNRS, Laboratoire Jean Kuntzmann, BP 53 38041, Grenoble Cedex 9, France.
georges-henri.cottet@imag.fr

Forward Semi Lagrangian methods have already been designed and used with success in plasma physics [9] and geophysical flows [6]. Semi Lagrangian particles, as they come out of RPM, have some distinctive features. the projection on the grid is explicit, based on explicit remeshing kernels, and does not require to solve a linear system to recover grid values. The computational effort can thus be restricted to the support of the solution and efficient parallel algorithms can be implemented. Moreover, remeshed particle methods can be derived at any order in a systematic fashion, whereas the methods derived in [6, 9] seem to be restricted to first order in space [23]. One goal of this paper is indeed to derive remeshed particle methods of arbitrary order, independently of any CFL condition, and to give a complete numerical analysis of a large class of second and fourth order methods.

RPM only involve local operations and are therefore well adapted to parallel implementations. In [24] an implementation of remeshed particle methods on GPU processors was described for the two-dimensional Navier–Stokes equations with penalization to account for solid obstacles. In the present paper, following [18], we consider implementations of two or three-dimensional RPM combined with directional splitting, that is where particles are pushed and remeshed successively in the three directions. Directional splitting allows to use high order remeshing kernels, with large stencils, at a minimal cost. We in particular discuss how directional splitting impacts the memory management to optimize implementations of RPM on GPU.

The outline of this paper is as follows. In Section 2 we give the definition of RPM, recall how remeshing kernels are derived and produce a list of high order kernels. In Section 3 we give the numerical analysis of RPM, based on the regularity and moment properties of the remeshing kernels. In Section 4 we present numerical illustrations and refinement studies of RPM in one to three dimensions. Section 5 is devoted to our implementation of RPM on GPU, in the context of a software library devoted to hybrid computations where different solvers can be implemented on different platforms. Finally Section 6 is devoted to concluding remarks and we give in the appendix the analytical formulas of the remeshing kernels that are considered in the paper.

2. Remeshed particle methods

In all the sequel we consider advection equations in the following conservation form

$$\frac{\partial u}{\partial t} + \mathrm{div}\ (au) = 0, x \in \mathbb{R}^d, t > 0, \tag{2.1}$$

where a is a given smooth velocity field.

2.1. Definition

In this paper we consider the case of uniform grids (we refer to [1, 2] for the design and applications of adaptive RPM with variable grid-size). RPM consist of alternating particle motion and remeshing. Particles are at the beginning of every time-step on a regular grid $x_i = i\Delta x$, $i \in \mathbb{Z}^d$, then move with the following equation

$$x_i^{n+1} = x_i + \tilde{a}_i^n \Delta t. \tag{2.2}$$

In the above equation $\tilde{a}_i^n$ denotes an evaluation of the velocity field at time $t_n = n\Delta t$ and location x_i which depends on the chosen time-stepping scheme. Remeshing follows, through interpolation with a remeshing kernel Γ which satisfies $\Gamma(-x) = \Gamma(x)$. If u_i^n denotes the value approximating $u(x_i, t_n)$, this gives the following formula:

$$u_i^{n+1} = \sum_j u_j^n \Gamma\left(\frac{x_j^{n+1} - x_i}{\Delta x}\right), i \in \mathbb{Z}^d, n \geqslant 0. \tag{2.3}$$

In traditional remeshing schemes for multidimensional problems, multidimensional kernels Γ are derived as tensor products of 1D formulas. In the present study we follow the approach initiated in [18] which consists of reducing the advection problem (2.1) into one-dimensional advection equations through directional splitting. One the one hand, this approach has the drawback that increasing the order in time of the method is less straightforward than for the original multidimensional problem. On the other hand, it offers the advantage to significantly reduce the computational cost of the method if high order kernels, with large stencils, are used. In short, if the kernel Γ involves a stencil with N_s grid points, in three dimensions a multidimensional remeshing method will cost $O(N_s^3)$ per particle, while a directional splitting will cost $O(3N_s)$ operations. For values of $N_s \gtrsim 6$, a typical case we will consider in the sequel, the ratio is larger than 10. Directional splitting also allows to derive limiting techniques to reduce oscillations [18] and, for the numerical analysis of the method, one only needs to consider the case $d = 1$.

2.2. Derivation of remeshing kernels

We consider the equation (2.1) and its approximation formulas (2.2), (2.3) for $d = 1$. As will be clear from the numerical analysis below, one key feature which controls the accuracy of RPM is the conservation of momentum up to a given order p. More precisely, one seeks to satisfy for the scheme (2.3) the following identity

$$\sum_i u_i^{n+1} x_i^\alpha = \sum_i u_i^n x_i^\alpha, 0 \le \alpha \le p, \tag{2.4}$$

for all sequences (u_i^n). It is readily seen that this is equivalent to the following moments conditions for the remeshing kernel Γ

$$\sum_{k\in\mathbb{Z}} k^\alpha \Gamma(x-k) = x^\alpha,\ 0 \le \alpha \le p,\ x \in \mathbb{R} \tag{2.5}$$

or, equivalently,

$$\sum_{k\in\mathbb{Z}} (x-k)^\alpha \Gamma(x-k) = \begin{cases} 1 & \text{if } \alpha = 0 \\ 0 & \text{if } 1 \le \alpha \le p \end{cases},\ x \in \mathbb{R}. \tag{2.6}$$

Note that for $\alpha = 0$ these conditions enforce the conservation of mass. Using these identities for a given value of p and assuming that the kernel Γ remeshes particle weights among the $p+1$ nearest grid points, one obtains a piecewise polynomial function of degree p. For $p = 1$ this gives the piecewise linear tent function. With $p = 2$ one obtains a piecewise quadratic function, the so-called Λ_2 formula, that has been used with success in the first particle simulation of the Navier–Stokes equations using remeshing in a systematic fashion [13].

This method to derive kernels is straightforward but has the drawback that it does not deliver smooth kernels. The kernel Λ_2 is not even continuous (this is the case more generally for kernels corresponding to even values of p; for odd values, the kernels are continuous but their derivatives are discontinuous). This lack of smoothness may result in a loss of accuracy or in oscillations, in particular if large time-steps are used. In [18] local correction techniques were derived to guarantee at least first order for the kernel Λ_2 and third order for the analogous kernel Λ_4 corresponding to $p = 4$.

To derive smooth kernels, one option is to use extrapolation techniques starting from smooth B-splines. If we denote by M_1 the top-hat filter with support in $[-1/2, +1/2]$, and by M_n its successive convolution: $M_n = M_1^{(*n)} \in W^{n,\infty}(\mathbb{R})$, the idea is to derive linear combinations of M_n, xM_n', x^2M_n'',... to cancel the successive continuous moments of Γ. More precisely, given an even integer $p \ge 2$ and $k > 1 + p/2$, one looks for coefficients $\alpha_1, \dots, \alpha_{p/2+1}$ such that $\Gamma = \sum_{l=0}^{p/2} \alpha_{l+1} x^l M_k^{(l)}$ satisfies

$$\int y^\alpha \Gamma(y)\,\mathrm{d}y = \begin{cases} 1 & \text{if } \alpha = 0, \\ 0 & \text{if } 1 \le \alpha \le p. \end{cases} \tag{2.7}$$

For symmetry reasons these conditions need only be enforced for even values of α. This leads to a square linear system of size $1 + p/2$ the coefficients of which only involve the even moments of M_k. With this method, one for instance readily obtains the following kernels

$$\Gamma = \frac{1}{2}(3M_4 + xM_4'), \tag{2.8}$$

and

$$\Gamma = \frac{1}{8}(15M_8 + 9xM_8' + x^2M_8''). \tag{2.9}$$

The first formula corresponds to a kernel of class C^1, piecewise cubic, with a support of size 4 and $p = 2$. It was derived under the name of M_4' in [19] and has been and still is extensively used in particle simulations, in particular of vortex flows. The second formula corresponds to a kernel of class C^4, piecewise polynomial of degree 7, with a support of size 8 and $p = 4$.

The moment conditions (2.7) concern *continuous* moments of the kernel. To check that the *discrete* moment conditions (2.6) are satisfied as well, one can use an equivalent condition using the Fourier transform of the kernel [7,29]. If

$$\widehat{\Gamma}(\xi) = \int \Gamma(y)\mathrm{e}^{-\mathrm{i}\xi y}\,\mathrm{d}y,$$

it can be shown ([7]) that the properties (2.5) are satisfied provided $\widehat{\Gamma}$ fulfills the following conditions:

$$\widehat{\Gamma}(\xi) - 1 \text{ has a zero of order } p+1 \text{ at } \xi = 0$$

$$\widehat{\Gamma}(\xi) \text{ has a zero of order } p+1 \text{ at all } \xi = 2\pi m, m \neq 0.$$

Since $\widehat{\Gamma}^{(\alpha)}$ is proportional to $y^\alpha \Gamma(y)$, the first condition above is clearly equivalent to the conditions (2.7). Moreover one has

$$\widehat{M_1}(\xi) = \frac{\sin(\xi/2)}{\xi/2}$$

and thus

$$\widehat{M_n}(\xi) = \left[\frac{\sin(\xi/2)}{\xi/2}\right]^n.$$

As a result, the Fourier transform of the functions $x^l M_k^{(l)}$ has a zero of order $k-l$ at all non zero multiple of π. From these observations, one can easily check that the kernels (2.8), and (2.9) do satisfy the discrete moment conditions (2.5) with $p = 2$ and $p = 4$ respectively.

The approach just presented leads to smooth and high order kernels. However the kernels derived in this fashion do not necessarily satisfy the following interpolation property

$$\Gamma(i) = \begin{cases} 1 \text{ if } i = 0, \\ 0 \text{ otherwise.} \end{cases} \tag{2.10}$$

Indeed the so-called kernel M_4' given in (2.8) does satisfy this condition, but not the one derived in (2.9) from M_8. Property (2.10) is natural as it ensures that, if the velocity is zero, the exact solution is algebraically conserved. Although this property does not enter the numerical analysis that follows, in practice it seems to have some importance, in particular to represent accurately the smallest scales in turbulent flows.

One can derive kernels which satisfy simultaneously and to any given order, regularity, moment conditions and the interpolation property. For a sake of simplicity, in the following we restrict ourselves to kernels with a stencil covering an even number $2M_s$ of grid points. We seek kernels Γ that have the following properties:

P1. Γ has support in $[-M_s, +M_s]$,
P2. Γ is even and piecewise polynomial of degree M in intervals of the form $[i, i+1]$,
P3. Γ is of class C^r,
P4. Γ satisfies the moment properties (2.5) for a given value of p,
P5. Γ satisfies the interpolation property (2.10).

TABLE 1. Kernels of various regularity, moment properties and complexity. In bold, the kernels that are considered in the numerical experiments of Section 4 and for which analytical formulas are given in the appendix.

	Moments (p in 2.5)	Regularity	Nb of grid points in stencil	degree	Support
$\boldsymbol{\Lambda_{2,1}}$	2	C^1	4	3	$[-2;2]$
$\boldsymbol{\Lambda_{2,2}}$	2	C^2	4	5	$[-2;2]$
$\Lambda_{2,3}$	2	C^3	4	7	$[-2;2]$
$\Lambda_{2,4}$	2	C^4	4	9	$[-2;2]$
$\boldsymbol{\Lambda_{4,2}}$	4	C^2	6	5	$[-3;3]$
$\Lambda_{4,3}$	4	C^3	6	7	$[-3;3]$
$\boldsymbol{\Lambda_{4,4}}$	4	C^4	6	9	$[-3;3]$
$\Lambda_{6,3}$	6	C^3	8	7	$[-4;4]$
$\boldsymbol{\Lambda_{6,4}}$	6	C^4	8	9	$[-4;4]$
$\Lambda_{6,5}$	6	C^5	8	11	$[-4;4]$
$\boldsymbol{\Lambda_{6,6}}$	6	C^6	8	13	$[-4;4]$
$\boldsymbol{\Lambda_{8,4}}$	8	C^4	10	9	$[-5;5]$

Such kernels are determined by $M_s(M+1)$ coefficients. The regularity property P3 imposes $M_s(r+1)$ interface conditions at integer values, and $[(r+1)/2]$ conditions to express that derivatives of odd order vanish at zero. The properties P4 and P5 impose $p+1+M_s$ conditions. One reasonable constraint under which one can expect to find kernels satisfying these conditions is therefore

$$M_s(M+1) \geq (r+1)M_s + [(r+1)/2] + p + 1 + M_s.$$

Table 1 lists several kernels that have been obtained through this approach by symbolic calculations. In this table, the kernels have been labelled by 2 indices that refer to the regularity and the order to which moment conditions are satisfied, as we will see that these are the parameters which control the order of accuracy of the RPM: $\Lambda_{p,r}$ is a kernel in $W^{r+1,\infty}(\mathbb{R})$ which satisfies (2.5). $\Lambda_{2,1}$ corresponds to the kernel M'_4 already mentioned. The kernel $\Lambda_{4,2}$ was derived with this approach and used for the first time in [2] under the name of M'_6. For a sake of completeness, we have provided in the appendix the analytical formulas for the kernels which are considered in the numerical experiments of Section 4: $\Lambda_{2,1}$, $\Lambda_{2,2}$, $\Lambda_{4,2}$, $\Lambda_{4,4}$, $\Lambda_{6,4}$, $\Lambda_{6,6}$ and $\Lambda_{8,4}$.

3. Numerical analysis

We consider in this section RPM with kernels satisfying the moment properties (2.5) and the following regularity conditions:

$$\Gamma \in W^{r+1,\infty}(\mathbb{R}) \text{ and } \Gamma \in \mathcal{C}^\infty\left(]l, l+1[\right), \; l \in \mathbb{Z}. \tag{3.1}$$

The RPM is defined by the formulas (2.2), (2.3) and we will denote by $\mathcal{T}_i u(\cdot, t_n)$ the result of the scheme (2.3), at the grid point x_i, starting from grid values $u(x_j, t_n)$.

In this section we are interested by the stability and spatial accuracy of the method. For a sake of simplicity we will therefore assume that a does not depend on time and that particles advance with an explicit first-order Euler scheme. In this case we simply have $\tilde{a}_j^n = a(x_j)$.

A striking feature of RPM, common with all semi-lagrangian methods, is that their stability does not rely on CFL conditions. In this section we prove stability and consistency results under the condition

$$\Delta t \leq \frac{M}{\|a'\|_{L^\infty}} \tag{3.2}$$

for a given constant $M < 1$. This condition in particular ensures that particle trajectories cannot intersect. The constant M is often called Lagrangian CFL number.

3.1. Consistency

We will prove the following consistency result

Proposition 3.1. *Assume that the condition* (3.2) *is satisfied, and that the moment and regularity conditions* (2.5), (3.1) *hold for some* $r, p > 1$. *Let* $T > 0$ *and assume further that* a *and the solution* u *to equation* (2.1) *belong to* $L^\infty\left(0, T; W^{r+1,\infty}(\mathbb{R})\right)$. *Then, if we set* $\beta = \inf(r, p)$, *the following estimate holds*

$$u\left(x_i, t^{n+1}\right) = \mathcal{T}_i\left(u\left(\cdot, t^n\right)\right) + O\left(\Delta t^2\right) + O\left(\Delta t \Delta x^\beta + \Delta x^{\beta+1} + \Delta t^{\beta+1}\right). \tag{3.3}$$

Moreover if every cell of size Δx *contains exactly one particle after an advection step, then* $\beta = p$.

Proof. If we use the notation u_j^n for $u(x_j, t_n)$ and set $j = i + k$ we can write the following expansion

$$u_j^n = \sum_{l=0}^{\beta} \frac{k^l \Delta x^l}{l!} u^{(l)}(x_i) + O\left(\Delta x^{\beta+1}\right), \tag{3.4}$$

where $u^{(l)}(x_i)$ stands for $\partial^l u / \partial x^l (x_i, t_n)$. Next, one can write $x_j^{n+1} = x_i + k\Delta x + a_{i+k}\Delta t$ and therefore

$$\Gamma\left(\frac{x_j^{n+1} - x_i}{\Delta x}\right) = \Gamma(k + \lambda_{i+k}),$$

where λ_j is the local CFL number: $\lambda_j = a_j \Delta t / \Delta x$. We then write $\lambda_j = \lambda_i + (a_j - a_i)\Delta t/\Delta x$ and use the following formula

$$(f \circ g)^{(m)}(x) = \sum_{|q|=1}^{m} c_q f^{q_1+\ldots+q_m}(g(x)) \prod_{s=1}^{m} \left(g^{(s)}(x)\right)^{q_s}$$

where $q = (q_1, \ldots, q_m)$ is a multi-index, $|q| = q_1 + 2q_2 + \ldots + mq_m$ and c_q are positive coefficients, to obtain

$$\begin{aligned}\Gamma\left(\frac{x_j^{n+1} - x_i}{\Delta x}\right) = &\sum_{m=0}^{\beta} (k\Delta x)^m \sum_{|q|=1}^{m} c_q \Gamma^{(q_1+\ldots+q_m)}(k + \lambda_i)\, \nu^{q_1+\ldots+q_m} \prod_{s=1}^{m} \left(a^{(s)}(x)\right)^{q_s} \\ &+ O\left((k\Delta x)^{\beta+1} \|\Gamma\|_{\beta+1,\infty} \|a\|_{\beta+1,\infty} \sum_{|q|=\beta+1} \nu^{q_1+\ldots+q_{\beta+1}}\right).\end{aligned} \tag{3.5}$$

In the above equation we have denoted by ν the ratio $\Delta t/\Delta x$. Since $q_1 + \ldots + q_{\beta+1} \leq |q|$ we have

$$\sum_{|q|=\beta+1} \nu^{q_1+\ldots+q_{\beta+1}} \leq C\left(\nu + \nu^{\beta+1}\right).$$

The remainder in (3.5) can thus be bounded by $O\left(\Delta x^{\beta+1}+\Delta x^{\beta}\Delta t+\Delta t^{\beta+1}\right)$. Combining (3.4) and (3.5) we obtain

$$\begin{aligned}
\mathcal{T}_i(u(\cdot,t_n)) &= \sum_j u_j^n \Gamma\left(\frac{x_j^{n+1}-x_i}{\Delta x}\right) \qquad (3.6)\\
&= \sum_k \sum_{0\le l+m\le\beta} (k\Delta x)^{l+m}\frac{u^{(l)}(x_i)}{l!}\sum_{|q|=1}^{m} c_q \Gamma^{(q_1+\ldots+q_m)}(k+\lambda_i)\,\nu^{q_1+\ldots+q_m}\prod_{s=1}^{m}\left(a^{(s)}(x_i)\right)^{q_s}\\
&\qquad + O\left(\Delta x^{\beta+1}+\Delta x^{\beta}\Delta t+\Delta t^{\beta+1}\right)\\
&= \sum_k E_k(x_i) + O\left(\Delta x^{\beta+1}+\Delta x^{\beta}\Delta t+\Delta t^{\beta+1}\right).
\end{aligned}$$

By differentiation, the moment conditions (2.5) yield

$$\sum_k k^{q'}\Gamma^{(q)}(k+x) = (-x)^{q'-q}q'(q'-1)\ldots(q'-q+1),\ \text{for } 0\le q\le q'\le\beta.$$

Using these identities with $x=\lambda_i$ and observing that $\lambda_i^{l+m-(q_1+\ldots+q_m)}\nu^{q_1+\ldots+q_m} = d_i(\Delta t/\Delta x)^{l+m}$ where d_i are coefficients only depending on a, we get

$$\sum_k E_k(x_i) = \sum_{0\le l+m\le\beta} O\left(\Delta t^{l+m}\right) + O\left(\Delta x^{\beta+1}+\Delta x^{\beta}\Delta t+\Delta t^{\beta+1}\right). \qquad (3.7)$$

One can easily check that the zero and first order terms in the above expansion, corresponding to $0\le l+m\le 1$, result in

$$u(x_i,t_n) - \Delta t\, a(x_i)\partial u/\partial x(x_i,t_n) - \Delta t\, a'(x_i)u(x_i,t_n) = u(x_i,t_n) - \Delta t\,\partial(a\,u)/\partial x(x_i,t_n).$$

We therefore finally obtain

$$\begin{aligned}
\mathcal{T}_i(u(\cdot,t_n)) &= u_i^n - \Delta t\frac{\partial(au)}{\partial x}(x_i,t_n) + O\left(\Delta t^2\right) + O\left(\Delta x^{\beta+1}\right) + O\left(\Delta x^{\beta}\Delta t\right) \qquad (3.8)\\
&= u(x_i,t^{n+1}) + O\left(\Delta t^2\right) + O\left(\Delta x^{\beta+1}+\Delta x^{\beta}\Delta t+\Delta t^{\beta+1}\right). \qquad (3.9)
\end{aligned}$$

To prove the second assertion of our proposition we observe that if, at the end of an advection step, every cell within the stencil centered at the grid point x_i contains exactly one particle, since particles cannot cross this means that λ_j and λ_i lie in the same interval between successive integers. The kernel Γ is $\mathcal{C}^\infty$ in such intervals and thus the expansion 3.5 is valid to any order. The coefficient β can therefore be taken equal to p. □

3.2. Stability

We start with the linear stability analysis and assume that the velocity field a is constant.

Linear stability

In this paragraph, we will show unconditional stability (with respect to $\lambda = a\,\Delta t/\Delta x$) for a certain class of remeshing kernels. We set, for $k\in\mathbb{Z}$,

$$\alpha_k(\lambda) = \Gamma(k+\lambda)\ ,\ A_k(\lambda) = \sum_i \alpha_i(\lambda)\alpha_{i+k}(\lambda). \qquad (3.10)$$

In all the calculations that follow we will write α_k and A_k for $\alpha_k(\lambda)$ and $A_k(\lambda)$, respectively, when there is no ambiguity on the value of λ. We start with the following result

Lemma 3.2. *If the kernel Γ satisfies* (2.5) *then*

$$\sum_{k,l} (k-l)^q \, \alpha_k \alpha_l = 0 \, , \; \sum_{k \geq 1} k^q \, A_k = 0, \textit{if } 1 \leq q \leq p, \; q \textit{ even.} \tag{3.11}$$

Proof. We have

$$\sum_{k,l} (k-l)^q \, \alpha_k \alpha_l = \sum_{k,l} \sum_{m=0}^{q} C_q^m k^m (-l)^{q-m} \, \alpha_k \alpha_l.$$

By the moment properties (2.5) $\sum_{k,l} k^m (-l)^{q-m} \, \alpha_k \alpha_l = (-\lambda)^m \lambda^{q-m}$ and thus

$$\sum_{k,l} (k-l)^q \, \alpha_k \alpha_l = \sum_{m=0}^{q} C_q^m \, (-\lambda)^m \lambda^{q-m} = 0.$$

The second identity readily follows, since $A_{-k} = A_k$. □

We consider the case of kernels involving stencils with an even number of points. We further restrict our analysis to second order kernels involving 4 points (which means that the support of Γ is the interval $[-2, +2]$) and fourth order kernels involving 6 points (the support of Γ is the interval $[-3, +3]$). The key properties of the remeshing kernel that are needed to ensure stability are the moment conditions and a sign and a decay properties for the kernel values. More precisely we will prove the following result

Proposition 3.3. *Assume that the velocity field a is constant, and that the kernel Γ satisfies one of the following conditions:*

(i) *Γ satisfies* (2.5) *with $p = 2$, Γ has support in* $[-2, +2]$ *and satisfies for all values of* $\lambda \in [0, 1]$

$$A_1 \geq 0 \, , \; A_2 \leq 0 \, , \; A_3 \geq 0 \, , \; -A_2 \geq 6A_3. \tag{3.12}$$

(ii) *Γ satisfies* (2.5) *with $p = 4$, Γ has support in* $[-3, +3]$ *and satisfies for all values of* $\lambda \in [0, 1]$

$$A_1 \geq 0 \, , \; A_2 \leq 0 \, , \; A_3 \geq 0 \, , \; A_4 \leq 0 \, , \; A_5 \geq 0 \, , \; A_3 \geq -8A_4. \tag{3.13}$$

Then the remeshed particle method is unconditionally stable.

Before proceeding with the proof of this result, some comments are in order. For all the kernels that are used in practice one has $\Gamma(x) \geq 0$ for $x \in [-1, 1]$ and Γ alternates sign in successive integer intervals. The conditions (3.12), (3.13) therefore mean that the kernel decays fast enough. In the case *i*, $A_2 = \alpha_{-2}\alpha_0 + \alpha_{-1}\alpha_1$ and $A_3 = \alpha_{-1}\alpha_1$. The condition $-A_2 \geq 6A_3$ is thus satisfied as soon has $\alpha_0 \geq 6\alpha_1$. For the $\Lambda_{2,1}$ kernel, this condition can easily be checked: one has, for $\lambda \in [0, 1]$

$$|\alpha_0/\alpha_1| = \frac{2 + 2\lambda - 3\lambda^2}{(1-\lambda)\lambda} = 2 + \frac{2 - \lambda^2}{(1-\lambda)\lambda} \geq 2 + \frac{1}{(1-\lambda)\lambda} \geq 6.$$

Similar calculations show that the other 2nd order kernels in the Table 1 satisfy this decay condition. For 4th order, 6 points kernels, the analytic calculations are more involved, but it is possible to check by symbolic calculations that the conditions (3.13) hold for the kernels that are indicated in Table 1. Our result only covers 2nd and 4th order kernels. However one can conjecture from the proof that will be given below that it extends to higher order kernels, with larger stencils, under an appropriate decay condition for the coefficients A_k.

We now proceed with the proof of Proposition 3.3 and begin with the case *i*.

We define by I the integer such that $\lambda = a\Delta t/\Delta x \in [I, I+1[$ and we set $\mu = \lambda - I \in [0, 1[$. We write $j = i - I + k$ and set $v_i = u_{i-I}^n$. The scheme (2.3) now reads

$$u_i^{n+1} = \sum_k v_{i+k} \Gamma(k + \mu). \tag{3.14}$$

Since the support of $\varGamma$ is in $[-2,+2]$, the coefficients $\alpha_k(\mu)$ vanish if $k \le -3$ or $k \ge 2$. Using (3.10) we obtain

$$\sum_i |u_i^{n+1}|^2 = \sum_i \sum_{-2\le k,l\le 1} v_{i+k}\, v_{i+l}\, \alpha_k(\mu)\, \alpha_l(\mu) = \sum_i v_i^2 \sum_k \alpha_k^2 + 2\sum_i \sum_{-2\le k<l\le 1} v_{i+k}\, v_{i+l}\, \alpha_k\, \alpha_l. \tag{3.15}$$

We then write

$$2v_{i+k}\, v_{i+l} = v_{i+k}^2 + v_{i+l}^2 - |v_{i+k} - v_{i+l}|^2.$$

The conservation of the first moment gives

$$1 = \left(\sum_k \alpha_k\right)^2 = \sum_k \alpha_k^2 + 2 \sum_{-2\le k<l\le 1} \alpha_k\, \alpha_l,$$

and therefore

$$\sum_i \left|u_i^{n+1}\right|^2 = \sum_i v_i^2 - \sum_i S_i \tag{3.16}$$

where $S_i = \sum_{-2\le k<l\le 1} \alpha_k\, \alpha_l |v_{i+k} - v_{i+l}|^2$.

Since obviously $\sum_i v_i^2 = \sum_i |u_i^n|^2$, it remains to prove that $\sum_i S_i \ge 0$. Using the change of index $l = k+m$ for $l > k$ and the notations in (3.10) allows to write

$$\sum_i S_i = \sum_k \sum_{1\le m\le 3} \alpha_k\, \alpha_{k+m} \sum_i |v_{i+m} - v_i|^2 = \sum_{1\le m\le 3} A_m \sum_i |v_{i+m} - v_i|^2.$$

We set $\delta_i = v_{i+1} - v_i$ to rewrite the above expression as

$$\sum_i S_i = A_1 \sum_i \delta_i^2 + A_2 \sum_i (\delta_i + \delta_{i+1})^2 + A_3 \sum_i (\delta_i + \delta_{i+1} + \delta_{i+2})^2. \tag{3.17}$$

Expanding the above sums we obtain

$$\sum_i S_i = (A_1 + 2A_2 + 3A_3) \sum_i \delta_i^2 + (2A_2 + 4A_3) \sum_i \delta_i\delta_{i+1} + 2A_3 \sum_i \delta_i\delta_{i+2}.$$

We again write $2\delta_i\delta_{i+1} = \delta_i^2 + \delta_{i+1}^2 - |\delta_i - \delta_{i+1}|^2$ and a similar identity for $2\delta_i\delta_{i+2}$ to obtain

$$\sum_i S_i = (A_1 + 4A_2 + 9A_3) \sum_i \delta_i^2 - (A_2 + 2A_3) \sum_i |\delta_i - \delta_{i+1}|^2 - A_3 \sum_i |\delta_i - \delta_{i+2}|^2.$$

By (3.11) with $q = 2$ we get $A_1 + 4A_2 + 9A_3 = 0$. If we set $\eta_i = \delta_i - \delta_{i+1}$ we thus have

$$\sum_i S_i = -(A_2 + 2A_3) \sum_i |\eta_i|^2 - A_3 \sum_i |\eta_i + \eta_{i+1}|^2.$$

Since $A_3 > 0$ and $|\eta_i + \eta_{i+1}|^2 \le 4(\eta_i^2 + \eta_{i+1}^2)$ we can write

$$\sum_i S_i = -(A_2 + 6A_3) \sum_i |\eta_i|^2 \ge 0,$$

provided the decay property (3.12) is satisfied.

We now turn to the second assertion of the proposition. We start from (3.15) and obtain an estimate similar to (3.17), with $m = 5$ instead of 3 since the kernel has now a support of size 6

$$\sum_i S_i = \sum_k \sum_{1\le m\le 3} \alpha_k\, \alpha_{k+m} \sum_i |v_{i+m} - v_i|^2 = \sum_{1\le m\le 5} A_m \sum_i |v_{i+m} - v_i|^2. \tag{3.18}$$

We set $\delta_i = v_{i+1} - v_i$ and obtain:

$$\begin{aligned}\sum_i S_i &= A_1 \sum_i \delta_i^2 + A_2 \sum_i (\delta_i + \delta_{i+1})^2 + A_3 \sum_i (\delta_i + \delta_{i+1} + \delta_{i+2})^2 \\ &\quad + A_4 \sum_i (\delta_i + \delta_{i+1} + \delta_{i+2} + \delta_{i+3})^2 + A_5 \sum_i (\delta_i + \delta_{i+1} + \delta_{i+2} + \delta_{i+3} + \delta_{i+4})^2 \\ &= (A_1 + 2A_2 + 3A_3 + 4A_4 + 5A_5) \sum_i \delta_i^2 + (2A_2 + 4A_3 + 6A_4 + 8A_5) \sum_i \delta_i \delta_{i+1} \\ &\quad + (2A_3 + 4A_4 + 6A_5) \sum_i \delta_i \delta_{i+2} + (2A_4 + 4A_5) \sum_i \delta_i \delta_{i+3} + 2A_5 \sum_i \delta_i \delta_{i+4}.\end{aligned}$$

Rewriting double products as above and introducing $\eta_i = \delta_i - \delta_{i+1}$, we obtain

$$\begin{aligned}\sum_i S_i &= (A_1 + 4A_2 + 9A_3 + 16A_4 + 25A_5) \sum_i \delta_i^2 \\ &\quad - (A_2 + 2A_3 + 3A_4 + 4A_5) \sum_i \eta_i^2 - (A_3 + 2A_4 + 3A_5) \sum_i (\eta_i + \eta_{i+1})^2 \\ &\quad - (A_4 + 2A_5) \sum_i (\eta_i + \eta_{i+1} + \eta_{i+2})^2 - A_5 \sum_i (\eta_i + \eta_{i+1} + \eta_{i+2} + \eta_{i+3})^2.\end{aligned}$$

By (3.11) with $q = 2$ we have $A_1 + 4A_2 + 9A_3 + 16A_4 + 25A_5 = 0$. Expanding the remaining squares in the above identity we get

$$\begin{aligned}\sum_i S_i &= -(A_2 + 4A_3 + 10A_4 + 20A_5) \sum_i \eta_i^2 - 2(A_3 + 4A_4 + 10A_5) \sum_i \eta_i \eta_{i+1} \\ &\quad - 2(A_4 + 2A_5) \sum \eta_i \eta_{i+2} - 2A_5 \sum \eta_i \eta_{i+3}.\end{aligned}$$

We again write $2\eta_i\eta_{i+1} = \eta_i^2 + \eta_{i+1}^2 - |\eta_i - \eta_{i+1}|^2$ and similar identities for the other double products to obtain

$$\begin{aligned}\sum_i S_i &= -(A_2 + 6A_3 + 20A_4 + 50A_5) \sum_i \eta_i^2 \\ &\quad + (A_3 + 4A_4 + 10A_5) \sum_i (\eta_i - \eta_{i+1})^2 + (A_4 + 4A_5) \sum_i (\eta_i - \eta_{i+2})^2 + A_5 \sum_i (\eta_i - \eta_{i+3})^2.\end{aligned}$$

Subtracting (3.11) with $q = 4$ from the corresponding identity with $q = 2$ yields $A_2 + 6A_3 + 20A_4 + 50A_5 = 0$. Moreover, since $A_4 \le 0$ and $A_5 \ge 0$ we can write

$$(A_4 + 4A_5) \sum_i (\eta_i - \eta_{i+2})^2 \ge A_4 \sum_i (\eta_i - \eta_{i+2})^2 \ge 4A_4 \sum_i (\eta_i - \eta_{i+i})^2.$$

Therefore

$$\sum_i S_i \ge (A_3 + 8A_4) \sum_i (\eta_i - \eta_{i+i})^2 \ge 0,$$

provided the decay property (3.13) is satisfied. This concludes our proof in the case of a constant velocity field.

General case

We now consider the general case of a smooth, non-constant, velocity field a.

Proposition 3.4. *Assume that the velocity field a is in $W^{1,\infty}(\mathbb{R})$ and that the assumptions of Proposition 3.3 are satisfied. Assume in addition that the kernel Γ satisfies the interpolation property (2.10). Then there exists a constant C, independent of Δt and Δx, such that*

$$\sum_i \left|u_i^{n+1}\right|^2 \le (1 + C\Delta t) \sum_i \left|u_i^n\right|^2. \qquad (3.19)$$

Proof. We will only give the proof in the case of a second order method corresponding to the case *i* of Proposition 3.3. The proof in the case of *ii* follows along the same lines.

The local Courant number λ can now vary from one particle to the next. To start with we will consider the case when this number remains in the same integer interval, that is we make the assumption that

$$\exists\, I \in \mathbb{Z} \text{ such that, } \forall i \in \mathbb{Z},\ \lambda_i = a(x_i, t_n)\Delta t/\Delta x \in [I, I+1].$$

We proceed like in the proof of Proposition 3.3. We set $\mu_j = \lambda_j - I \in [0, 1[$, $v_i = u^n_{i-I}$ and $j = i - I + k$, and the scheme (3.14) becomes

$$u_i^{n+1} = \sum_k v_{i+k}\Gamma(k + \mu_j) = \sum_k v_{i+k}\alpha_k(\mu_j). \tag{3.20}$$

We first observe that, by the regularity of the velocity field a,

$$|\lambda_j - \lambda_{j'}| = O\left(|j - j'|\Delta t\right)$$

and, since $\Gamma \in W^{1,\infty}(\mathbb{R})$, for $-2 \le k < l \le 1$, $j = i - I + k$, $j' = i - I + l$

$$\alpha_k(\mu_j) = \alpha_k(\mu_{j'}) + O(\Delta t). \tag{3.21}$$

In particular $\alpha_k(\mu_j) = \alpha_k(\mu_{i-I}) + O(\Delta t)$ if $j = i - I + k$ with $k \in [-2, 1]$. This allows to write

$$\sum_i |u_i^{n+1}|^2 = \sum_i \left[\sum_{-2\le k\le 1} v_{i+k}\alpha_k(\mu_{i-I})\right]^2 + O\left(\Delta t|v|^2\right),$$

where we have used the notation $|v|^2 = \sum_i |v_i|^2$. Proceeding like in estimates (3.16), (3.17) we can write:

$$\sum_i |u_i^{n+1}|^2 = O\left(\Delta t|v|^2\right) + \sum_i \left[\sum_k |v_{i+k}|^2\alpha_k^2(\mu_{i-I}) + \sum_{k<l}\left(|v_{i+k}|^2 + |v_{i+l}|^2 - |v_{i+k} - v_{i+l}|^2\right)\alpha_k(\mu_{i-I})\alpha_l(\mu_{i-I})\right]. \tag{3.22}$$

Using again (3.21) and setting $j = i + k$ we have

$$\sum_i\sum_k |v_{i+k}|^2\alpha_k^2(\mu_{i-I}) = \sum_j\sum_k |v_j|^2\alpha_k^2(\mu_{j-I}) + O\left(\Delta t|v|^2\right),$$
$$\sum_i\sum_{k<l} |v_{i+k}|^2\alpha_k(\mu_{i-I})\alpha_l(\mu_{i-I}) = \sum_j\sum_{k<l} |v_j|^2\alpha_k(\mu_{j-I})\alpha_l(\mu_{j-I}) + O\left(\Delta t|v|^2\right),$$
$$\sum_i\sum_{k<l} |v_{i+l}|^2\alpha_k(\mu_{i-I})\alpha_l(\mu_{i-I}) = \sum_j\sum_{k<l} |v_j|^2\alpha_k(\mu_{j-I})\alpha_l(\mu_{j-I}) + O\left(\Delta t|v|^2\right),$$
$$\sum_i\sum_{k<l} |v_{i+k} - v_{i+l}|^2\alpha_k(\mu_{i-I})\alpha_l(\mu_{i-I}) = \sum_j\sum_m A_m(\mu_{j-I})|v_j - v_{j+m}|^2 + O\left(\Delta t|v|^2\right).$$

This allows us to rewrite the summation in the right hand side of (3.22) as

$$\sum_j |v_j|^2\left[\sum_k \alpha_k^2(\mu_{j-I}) + 2\sum_{k<l}\alpha_k(\mu_{j-I})\alpha_l(\mu_{j-I})\right] - \sum_j\sum_m A_m(\mu_{j-I})|v_j - v_{j+m}|^2 + O\left(\Delta t|v|^2\right).$$

By the conservation of the first moment we have

$$\sum_k \alpha_k^2(\mu_{j-I}) + 2\sum_{k<l}\alpha_k(\mu_{j-I})\alpha_l(\mu_{j-I}) = 1$$

and, like for the constant velocity case, it only remains to check the sign of $S = \sum_j \sum_m A_m(\mu_{j-I})|v_j - v_{j+m}|^2$. We proceed like in the constant velocity case and set $\delta_i = v_{i+1} - v_i$ and $\eta_i = \delta_{i+1} - \delta_i$. We expand the squares in S using the fact that

$$\delta_{i+1}^2 A_2(\mu_{i-I}) = \delta_{i+1}^2 A_2(\mu_{i+1-I}) + O(\Delta t)\left(|v_{i+2}|^2 + |v_{i+1}|^2\right)$$

and similar expressions for $\delta_{i+1}^2 A_3(\mu_{i-I})$ and $\delta_{i+2}^2 A_3(\mu_{i-I})$. This leads to

$$\begin{aligned} S =& |v|^2 O(\Delta t) + \sum_i \delta_i^2 \left[A_1(\mu_{i-I}) + 4A_2(\mu_{i-I}) + 16A_3(\mu_{i-I})\right] - \sum_i \eta_i^2 \left[A_2(\mu_{i-I}) + 2A_3(\mu_{i-I})\right] \\ & - \sum_i (\eta_i + \eta_{i+1})^2 A_3(\mu_{i-I}). \end{aligned} \tag{3.23}$$

We next write

$$(\eta_i + \eta_{i+1})^2 A_3(\mu_{i-I}) \le 2\left(\eta_i^2 A_3(\mu_{i-I}) + \eta_{i+1}^2 A_3(\mu_{i+1-I})\right) + C\Delta t\left(|v_{i+3}|^2 + |v_{i+2}|^2 + |v_{i+1}|^2\right).$$

Due to (3.11), under the condition (3.12), (3.23) finally yields

$$S \ge -C\Delta t|v|^2$$

and therefore

$$\sum_i |u_i^{n+1}|^2 \le \sum_i |u_i^n|^2 + C\Delta t.$$

We finally turn to the general case, when neighboring particles can have their local CFL numbers in different integer intervals. We can always group particles in subsets where the CFL numbers are in the same integer interval. For $I \in \mathbb{Z}$, we set

$$J_I = \{\, j \in \mathbb{Z} \text{ such that } \lambda_j \in [I, I+1[\ \}.$$

We can rewrite the remeshed particle method as

$$u_i^{n+1} = \sum_I \sum_{j \in J_I} u_j^n \Gamma(\lambda_j + j - i) = \sum_I S_I(i). \tag{3.24}$$

The next step is to take the square of the above identity. The key point is to observe that, by the interpolation property (2.10), double products of the form $S_I(i)S_{I'}(i)$ with $I \ne I'$ are of order Δt. We indeed observe that, due to the regularity of a, we have

$$|\lambda_j - \lambda_{j'}| \le C\Delta t|j - j'|$$

and therefore, if the kernel Γ has a support of size $2M_s$ and if Δt is small enough

$$\Gamma(\lambda_j + j - i)\Gamma(\lambda_{j'} + j' - i) \ne 0 \Rightarrow |j - j'| \le 2M_s + 1. \tag{3.25}$$

Next, again in view of the regularity of a, if two neighboring particles have indices which belong to a different set J_I this implies that their local CFL number is close to an integer value. More precisely

$$\begin{aligned} &[j \in J_I, j' \in J_{I'}, I \ne I', |j - j'| \le 2M_s + 1] \Rightarrow \\ &\quad \exists (K, K') \in \mathbb{Z}^2, K \ne K' \text{such that } j + \lambda_j = K + O(\Delta t),\ j' + \lambda_{j'} = K' + O(\Delta t). \end{aligned}$$

By the interpolation property (2.10) and the regularity of Γ this implies

$$[j \in J_I, j' \in J_{I'}, I \ne I', |j - j'| \le 2M_s + 1] \Rightarrow \Gamma(\lambda_j + j - i)\Gamma(\lambda_{j'} + j' - i) = O(\Delta t).$$

Combined with (3.25) this shows that, for $I \neq I'$,

$$S_I(i)S_{I'}(i) \le C\Delta t \sum_{j\in J_I \bigcup J_{I'}} |u_j^n|^2. \tag{3.26}$$

It remains now to sum over i the above identities. To identify the indices j which, for a given index i, contribute to the sum $S_I(i)$, we denote by ϕ the application such that

$$\phi(x) = x + a(x)\Delta t.$$

In view of (3.2) we have $0 < 1 - M \le \phi'(x) \le 1 + M$. ϕ is thus one-to-one and its inverse ϕ^{-1} is strictly increasing. If we set $\psi(i) = \phi^{-1}(i\Delta x)/\Delta x$, the indices j which contribute to $S_I(i)$ must satisfy

$$i\Delta x - M_s\Delta x \le j\Delta x + a_j\Delta t \le i\Delta x + M_s\Delta x$$

which yields

$$\psi(i - M_s) \le j \le \psi(i + M_s).$$

If a particle j contributes to two different terms $S_I(i)$ and $S_I(i')$ with, say, $i < i'$ this means that

$$\psi(i' - M_s) \le j \le \psi(i + M_s)$$

and therefore

$$|i' - i| \le M_s.$$

As a result, a given particle of index j contributes to at most $2M_s$ sums $S_I(i)$. One can thus deduce from (3.26) that, for a certain constant C independent of Δt and Δx

$$\sum_i \sum_{I\neq I'} S_I(i)S_{I'}(i) \le C\Delta t|u^n|^2,$$

and, from (3.24),

$$\sum_i |u_i^{n+1}|^2 \le \sum_i \sum_I S_I^2(i) + C\Delta t|u^n|^2. \tag{3.27}$$

Finally, if we set

$$w_j^I = \begin{cases} u_j^n \text{ if } j \in J_I \\ 0 \quad \text{otherwise,} \end{cases}$$

by the preceding proof in the case when I takes a constant value, we have

$$\sum_i S_I^2(i) \le (1 + C\Delta t)|w^I|^2$$

and

$$\sum_I \sum_i S_I^2(i) \le (1 + C\Delta t) \sum_I |w^I|^2.$$

In view of (3.27) this leads to

$$\sum_i |u_i^{n+1}|^2 \le |u^n|^2 + C\Delta t|u^n|^2$$

and our proof is completed. □

Remark 3.5. For $CFL \le 1$, Remeshed Particle Methods reduce to finite-difference schemes [8]. The convergence analysis above contains thus as a by-product a convergence proof of a class of conservative second and fourth order finite-difference schemes.

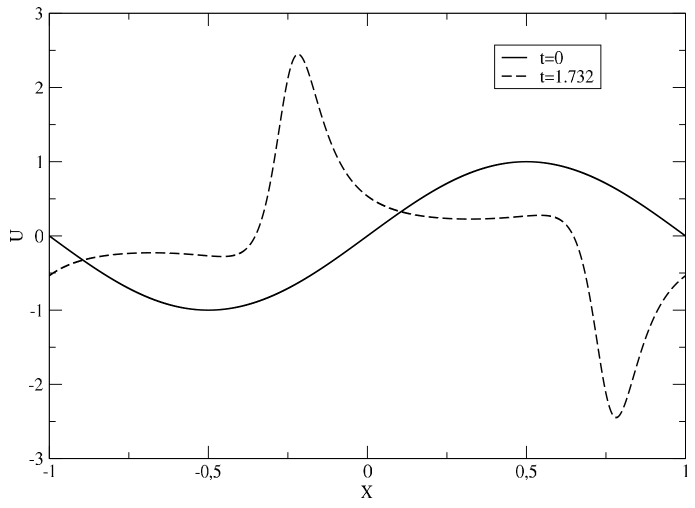


FIGURE 1. Advection equation with data (4.1). Initial condition (solid curve) and solution at $t = \sqrt{3}$ (dashed curve).

4. Numerical illustrations

In this section we compare the accuracy of the various kernels considered above in 1D, 2D and 3D smooth flows. The reference [15] deals with remeshed particle methods using the $\Lambda_{4,2}$ kernel in the context of the transport of passive scalar in turbulent flows. It in particular emphasizes the efficiency of this method in the case of high Schmidt numbers, that is when the diffusivity of the scalar is small compared to the viscosity of the flow. The extension of this work to higher order kernels introduced in the present paper will be reported in a future work.

We first consider a 1D advection equation in the interval $[-1, +1]$ with periodic boundary conditions. The velocity a and the initial condition u_0 are given by

$$a(x) = 1 + \sin(\pi x)/2 \ , \ u_0(x) = \sin(\pi x). \tag{4.1}$$

This flow induces compression and dilatation which successively increase and decrease the value of the solution. The exact solution is given by the following formula :

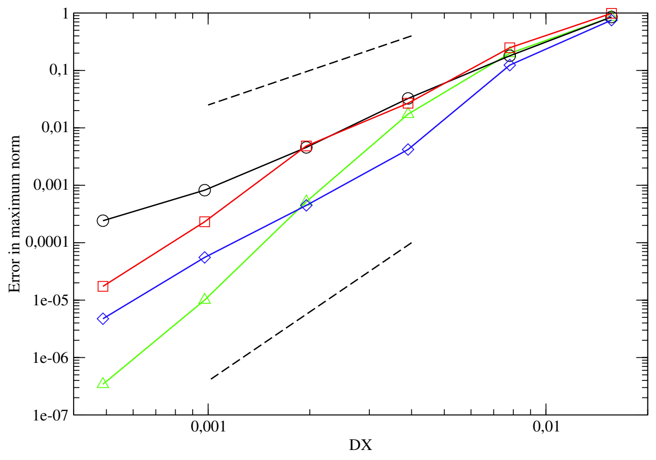
$$u(x(t), t) = u_0(x_0) \frac{2 + \sin 2\pi x_0}{2 + \sin 2\pi x(t)}$$

where $x(t)$ is the trajectory with origin x_0 associated to the velocity field a, given by

$$\arctan[\tan(\pi x(t)) + 1/2] = \arctan[\tan(\pi x_0) + 1/2] + t\pi\sqrt{3}/2.$$

The solutions are periodic in time with period $T = 4/\sqrt{3}$.

Figure 1 shows the solution at initial time and $t = \sqrt{3}$. For this case we compare in Figure 2 the results obtained with the kernels $\Lambda_{2,1}$, $\Lambda_{4,2}$, $\Lambda_{2,2}$ and $\Lambda_{4,4}$. In all cases, and throughout this section, particles where pushed with a fourth order Runge–Kutta time-stepping. The number of grid points ranged from 128 to 4096. For the purpose of this refinement study, we had to choose a constant value for the ratio $\Delta t/\Delta x$. We fixed a CFL value of 12. This value corresponds, for the coarsest resolution considered in this test, to a constant M in (3.2) equal to 0.7. The error is measured in the maximum norm. The numerical analysis above indicates that the two last kernels should deliver respectively second and fourth order methods, while the two first kernels,



Kernel	Order of convergence
$\Lambda_{2,1}$	2.35
$\Lambda_{2,2}$	3.15
$\Lambda_{4,2}$	3.45
$\Lambda_{4,4}$	4.25

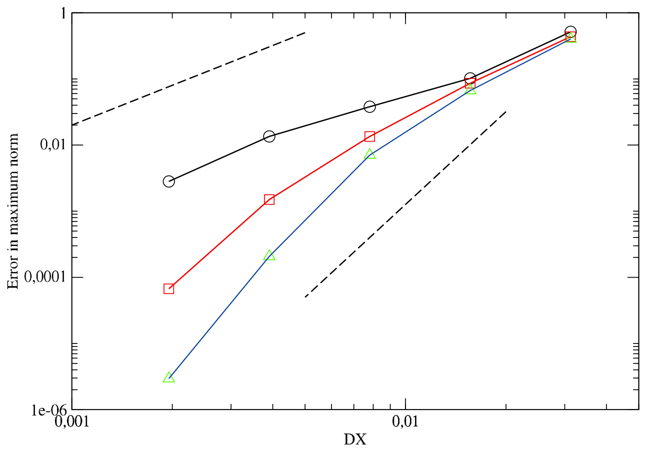
FIGURE 2. Refinement study for the 1D advection equation (4.1) and several first to fourth order RPM and CFL value equal to 12. *Left picture*: *black-circle curve*: kernel $\Lambda_{2,1}$; *red-square*: kernel $\Lambda_{2,2}$; *blue-diamond*: kernel $\Lambda_{4,2}$; *green-triangle*: $\Lambda_{4,4}$; dashed lines indicate slopes corresponding to second and fourth order convergence. Right table: average order of convergence for these RPM.

because of their lack of regularity, should be limited to first and second order, respectively. One can indeed observe that, for a given number of conserved moments, increasing the regularity does improve the accuracy of the RPM. However it is interesting to note that the kernel $\Lambda_{2,1}$ already gives a second order method. The reason is that, in this example, zones where the solution undergoes large variations, and thus where numerical errors are likely to be larger, only rarely correspond to grid points where the local CFL numbers cross integer values.

To illustrate next the directional splitting which we use to address multidimensional problems, we first consider the classical two-dimensional case of an off-centered circle strained in an incompressible rotational flow. More precisely, the computational box is the square $[0, 1]^2$. The velocity field is given by

$$\mathbf{a}(x_1, x_2, t) = f(t)\left(-\sin^2(\pi x_1)\sin(2\pi x_2), \sin(2\pi x_1)\sin^2(\pi x_2)\right). \tag{4.2}$$

The initial condition is a sign function whose zero level set is a circle of radius 0.15 and centered at the point $(0.5, 0.15)$. The function $f(t)$ is introduced to allow the solution to eventually return to the initial condition. We take $f(t) = \cos(\pi t/12)$, and compare the computed solution at time $t = 12$ with the initial condition. In this experiment, like in the 3D case below, we use the classical second order Strang formula for the dimensional splitting, that is we successively push and remesh particles in the x_1, x_2, x_2 and x_1 directions for $\Delta t/2$. It is indeed possible to derive and use higher order splitting methods, but our choice of a second order splitting was made for a practical reason – fourth order splitting methods are more demanding in terms of CPU and memory requirement. Furthermore, although second order splitting limits the theoretical order of convergence to 2, one can still see the benefit of using higher order kernels. Using a 6th order kernel in this example indeed yields an effective order of convergence close to 6. In this experiment the number of grid points in each direction ranged form 32 to 512. Like in the previous experiment, for the purpose of this refinement study, the CFL number was kept constant, equal to 12. The average order of convergence is shown on the right picture of Figure 3. In this experiment the kernels $\Lambda_{2,2}$, $\Lambda_{4,4}$ and $\Lambda_{6,6}$ would give similar results to the kernels $\Lambda_{2,1}$, $\Lambda_{4,2}$ and $\Lambda_{6,4}$ respectively. Figure 4 shows the solution obtained with $N = 256$ grid points in each direction and the kernels $\Lambda_{2,1}$ and $\Lambda_{6,4}$, at $t = 6$ and $t = 12$, compared to the exact solution. At $t = 6$, which is the time of maximal stretching for this experiment, the exact solution was obtained by a front tracking method using 10 000 markers.



Kernel	Order of convergence
$\Lambda_{2,1}$	1.87
$\Lambda_{4,2}$	3.17
$\Lambda_{6,4}$	5.92

FIGURE 3. Refinement study for the 2D advection field (4.2). CFL value is equal to 12. Left picture: *black-circle curve*: kernel $\Lambda_{2,1}$; *red-square*: kernel $\Lambda_{4,2}$; *blue-triangle*: kernel $\Lambda_{6,4}$; dashed lines indicate slopes corresponding to second and fourth order convergence. Right table: average order of convergence for these RPM.

This study clearly shows the gain obtained in this case by using high order kernels, even though the theoretical order of convergence is limited by that of the dimensional splitting.

As already stressed, one distinctive feature of Semi Lagrangian Particles are that their stability and the spatial order of convergence is not constrained by a CFL condition. To illustrate this property, we repeated the above experiment with 256 points in each direction and a CFL number equal to 30. These grid resolution and time-step give a value of the constant M in (3.2) equal to 0.35. Figure 5 shows the zero level set at times 6 and 12. Although one cannot expect the same overall accuracy as in the previous experiment with such a large time-step, the interface appears to remain rather well captured.

We now turn to a 2D example where the kernel regularity seems to be more important than in the previous cases. We consider an expansion field given by

$$\mathbf{a}(x_1, x_2) = (x_1/r, x_2/r), r = \sqrt{x_1^2 + x_2^2}, \tag{4.3}$$

and the initial value of u is an axisymmetric function supported in an annulus:

$$u_0(r) = \begin{cases} C[(r - r_a)(r - r_b)]^4 \text{ if } r_a \le r \le r_b \ , \\ 0 \text{ otherwise.} \end{cases}$$

with $r_a = 0.1, r_b = 0.25$ and $C = [2/(r_a - r_b)]^8$.

The exact solution at time t is given by $u(r,t) = u_0(r-t)(r-t)/r$. Figure 6 shows the result of a refinement study for the kernels $\Lambda_{2,1}$, $\Lambda_{2,2}$, $\Lambda_{4,2}$ and $\Lambda_{4,4}$. In that case areas where the local CFL number cross integer values are aligned with the flow directions. As a result the kernel $\Lambda_{2,1}$ only gives first order convergence, while $\Lambda_{2,2}$ is close to the second order accuracy predicted by our analysis. Note that increasing the order and regularity of the kernel improves the convergence but does not allow to reach fourth order. In this example the second order time-splitting limits the observed overall accuracy.

We close this section with a 3D version case suggested in [17] and often used to validate level set methods. In this example the computational box is the unit cube and the initial condition is a sphere of radius 0.15 centered

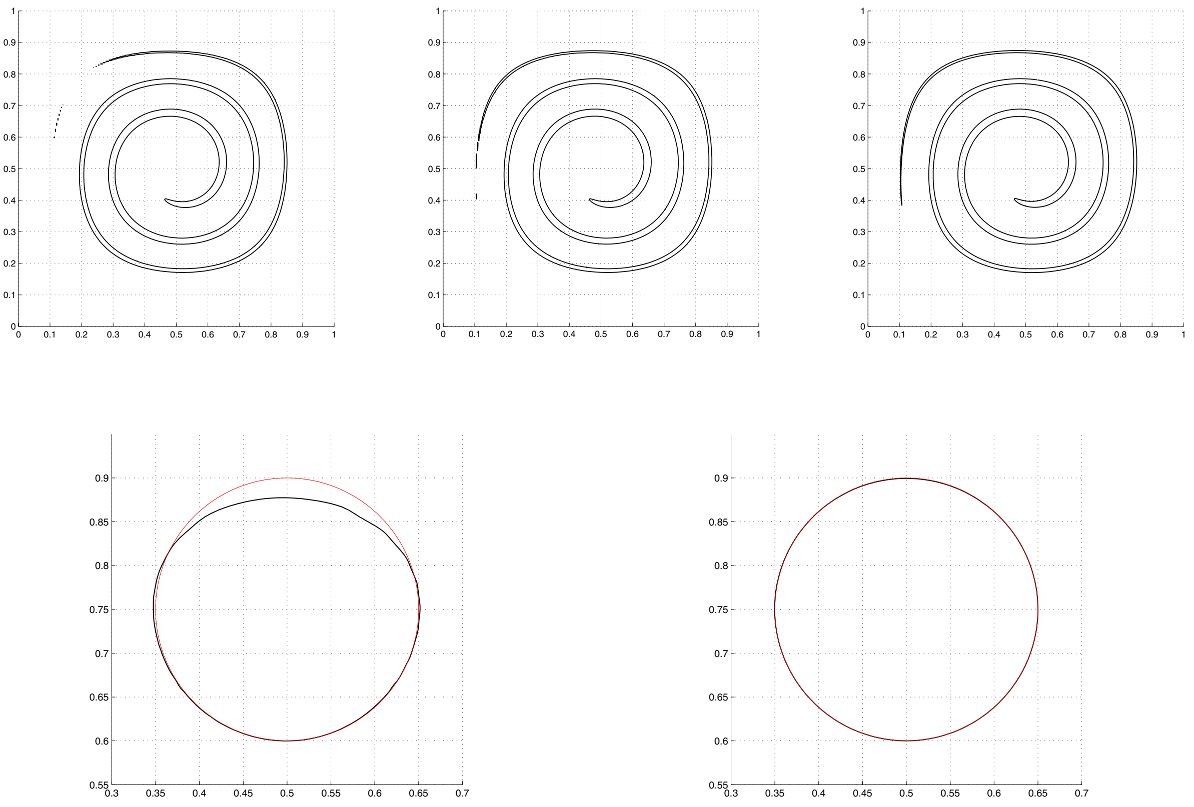


FIGURE 4. Interface $u = 0$ for the advection field (4.2) with $N_x = N_y = 256$. Top-left: $t = 6$, kernel $\Lambda_{2,1}$; top-middle: $t = 6$, kernel $\Lambda_{6,4}$; top-right: exact front-tracking solution; bottom-left: $t = 12$, kernel $\Lambda_{2,1}$; bottom-right: $t = 12$, kernel $\Lambda_{6,4}$. On the bottom pictures the red curve represents the exact (initial) interface at $t = 12$.

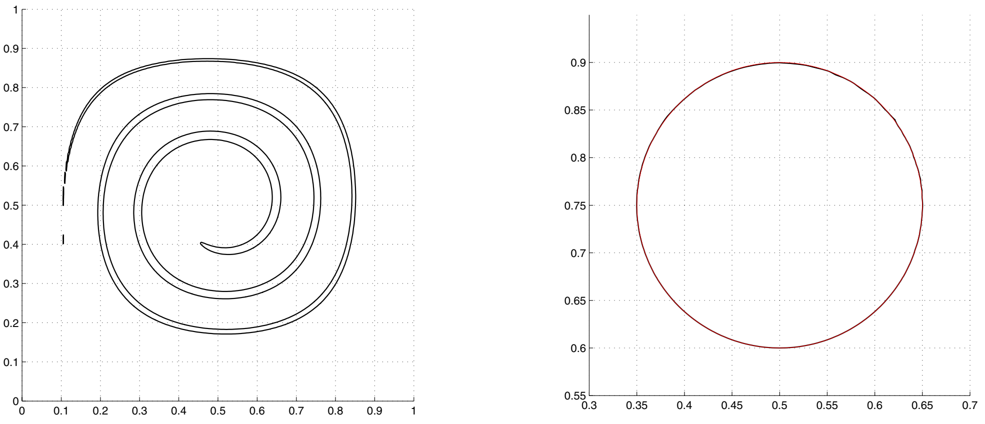
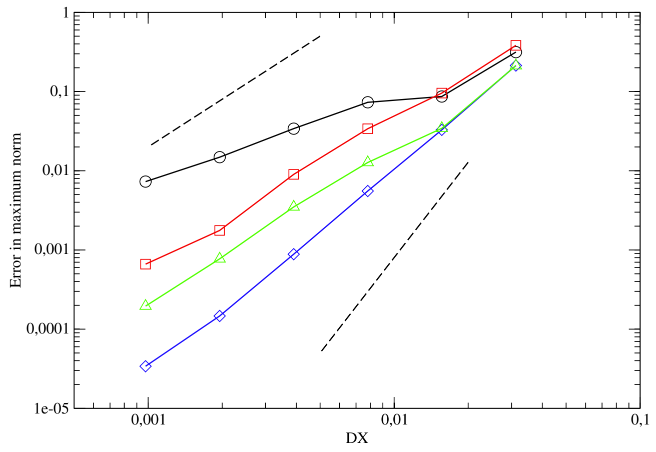


FIGURE 5. Same experiment as in Figure 4 with $Nx = Ny = 256$, the kernel $\Lambda_{6,4}$ and $CFL = 30$. Left picture: interface at $t = 6$; right picture: interface at $t = 12$ (red curve is the exact solution).



Kernel	Order of convergence
$\Lambda_{2,1}$	1.08
$\Lambda_{2,2}$	1.83
$\Lambda_{4,2}$	2.01
$\Lambda_{4,4}$	2.52

FIGURE 6. Refinement study for the radial test case (4.3). Left picture: *black-circle curve*: kernel $\Lambda_{2,1}$; *red-square*: kernel $\Lambda_{2,2}$; *blue-diamond*: kernel $\Lambda_{4,2}$; *green-triangle*: $\Lambda_{4,4}$; dashed lines indicate slopes corresponding to second and fourth order convergence. Right table: average order of convergence for these kernels. The CFL number is equal to 4.

around the point with coordinates $(0.35, 0.35, 0.35)$. This sphere is advected with the velocity field

$$a_1(x_1, x_2, x_3) = 2\,\sin^2(\pi x_1)\sin(2\pi x_2)\sin(2\pi x_3), \tag{4.4}$$

$$a_2(x_1, x_2, x_3) = -\,\sin(2\pi x_1)\sin^2(\pi x_2)\sin(2\pi x_3), \tag{4.5}$$

$$a_3(x_1, x_2, x_3) = -\,\sin(2\pi x_1)\sin(2\pi x_2)\sin^2(\pi x_3). \tag{4.6}$$

This motion wraps the sphere into thinner and thinner surfaces difficult to capture. In this case we want also to illustrate the fact that RPM only need particles in the support of the advected quantity. To capture the sphere we use a color function (1 inside the sphere, 0 outside). At the end of the remeshing step, particles are created only when their strength is above 0.001. In this experiment, we use 256 grid points in each direction. The CFL number is equal to 30, which corresponds to the same value of M as in the 2D case (4.2). The number of particles roughly varies between 1.5%, at $t = 0$, and 10%, at $t = 4$, of the total number of grid points inside the computational box. Due to the cut-off used in the remeshing step, the total mass was not algebraically conserved. However, in the present experiments the deviation did not exceed 0.1% of the total mass. Figure 7 shows the evolution of the volume inside the interface (this volume should remain constant at all times) obtained with the kernels $\Lambda_{2,1}$, $\Lambda_{4,2}$ and $\Lambda_{8,4}$. Figure 8 show the interface $u = 0.5$ at time $t = 4$, with the kernels $\Lambda_{2,1}$ and $\Lambda_{8,4}$. These results confirm the gain in accuracy that can be obtained by using high order kernels. The next section will discuss the computational complexity associated with these kernels.

5. Remeshed particle methods and software engineering

One challenge in High Performance Computing is to develop algorithms that are able to take advantage of clusters made of emerging exascale hardware. These clusters are hybrid architectures which combine many heterogeneous distributed processors and accelerators. A natural idea is to match models, algorithms and physical scales resolved by these algorithms to the specific features of the processors involved in these architectures.

A typical example is the case of turbulent transport in incompressible flows. Hybrid algorithms can be designed to resolve in an optimal fashion the minimal scales present in the flow and in the advected scalar [15]. Moreover in these hybrid approaches the different algorithms can have different parallel scalability and it may be

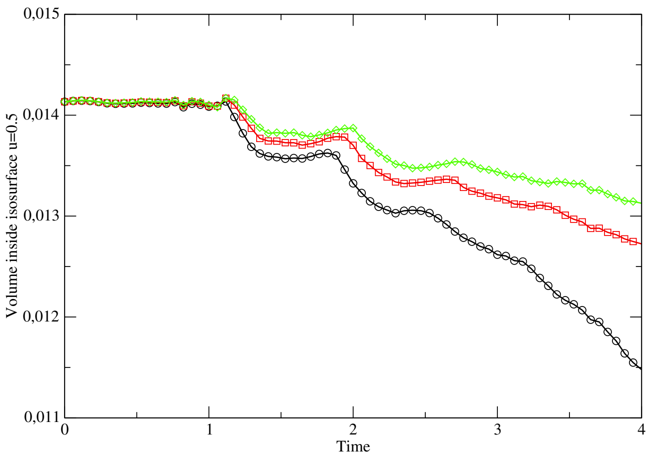


FIGURE 7. Volumes inside the interface $u = 0.5$ for the test case (4.4)−(4.6) with $CFL = 30$ and 256 grid points in each direction. *Green-circle curve*: kernel $\Lambda_{2,1}$; *red-square curve*: kernel $\Lambda_{4,2}$; green-diamond curve: kernel $\Lambda_{8,4}$.

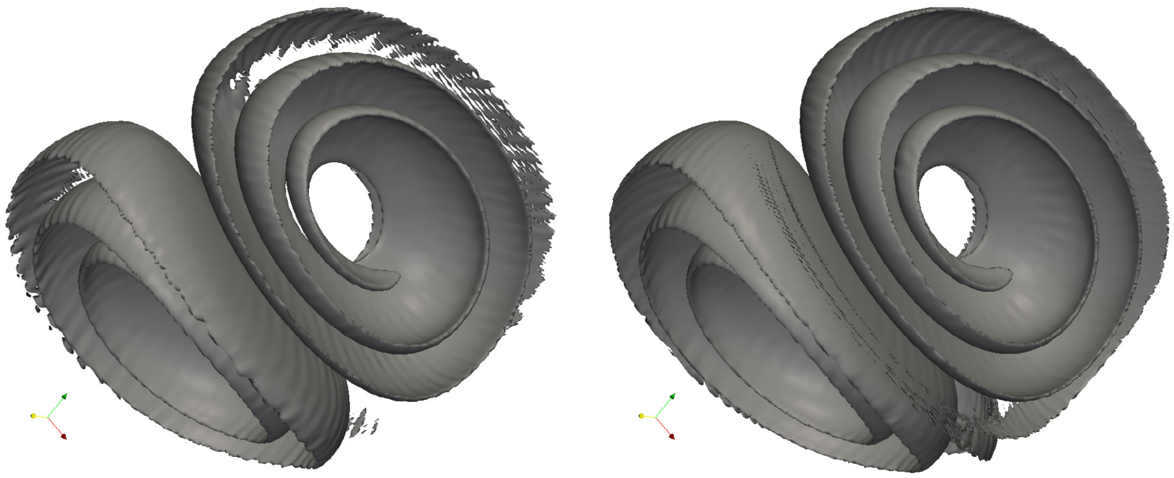
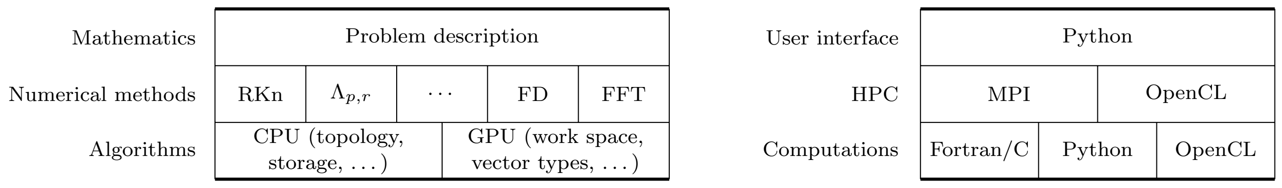


FIGURE 8. Isosurface $u = 0.5$ at $t = 4.0$ for the test case of the sphere in the flow (4.4)−(4.6) with $N = 256$ and $CFL = 30$. Left picture: kernel $\Lambda_{2,1}$; right picture: kernel $\Lambda_{8,4}$.

advantageous to distribute these algorithms to different type of processors. The local nature of RPM make these methods well suited for highly parallel processors, like GPUs, and one may envision simulations of transport at high Schmidt numbers in turbulent flows (that is when the scalar smallest scales extend much beyond those of the flows) at very high resolution at a cost that does not exceed that of the flow solver at a much lower resolution.

To be able to implement hybrid algorithms on hybrid architectures, one needs to develop high level frameworks and libraries with a high level description which allows to distribute different solvers and grids to different parts of the clusters in a seamless fashion.



(a) Functional view (b) Programming view

FIGURE 9. Application layout.

In this section we first outline the general approach followed for this framework. We then go into more details for a specific branch of this framework, namely the implementation on GPU of RPM. We detail the performance of the different kernels associated to the different parts of the algorithm and the overall performance of the GPU implementation of RPM. We in particular focus on the complexity of the RPM using various high order kernels that have been derived and analyzed in the previous section.

5.1. Library description

In this library we use object oriented programming techniques to reach a high level of modularity, with a strong focus on usability and flexibility. The goal is to enable the user to launch indifferently sequential, parallel or hybrid numerical simulations. We use Python as an abstraction framework.

Our library currently provides two levels of parallelism, MPI and OpenCL. However, thanks to the modularity it would also be possible to implement other parallelism paradigms such as task parallelism.

In order to achieve good portability, the computational frameworks are written using OpenCL C whenever it will enable good performances on the target architecture. OpenCL is an open standard for parallel programming of heterogeneous systems [21]. It provides application programming interfaces to address hybrid platforms containing many CPUs and GPUs and a programming language based on C99 to write instructions executed concurrently on the OpenCL devices. This framework is used through the PyOpenCL python interface to OpenCL [11].

In OpenCL applications, the program, that executes on the host system, must send OpenCL kernels to available devices in a command queue. The execution model of the application defines the devices and queues management. A given kernel contains an executable code that concurrently runs on device compute units which are called work-items. A memory model needs to be explicitly defined to manage data layout in the memory hierarchy. Details of these models, such as the number of work-items in a work-group or the memory access patterns, have a very strong impact on the program efficiency.

The high level of abstraction of Python allows to conceal from the user the parallel paradigms and the low level implementations of the numerical algorithms. It will participate in our goal to develop a library that is portable to various kind of hybrid architectures of modern computers.

Figure 9a shows the global design of the library from a functional point of view using concepts from [14]. In this work, a methodology is proposed for the design of object oriented codes. It relies on a distinction of components in different fields: abstract mathematical concepts are independent of their implementation and of the numerical methods. In order to use the library, the user only needs to describe his problem with high level components – mathematical operators, problem variables and physical domain description. The resolution is performed using default options *(Mathematics level)*. The user chooses the available numerical methods *(Numerical methods level)* and algorithms *(Algorithms level)*, or develop his own modules, depending on his needs. Thanks to the flexibility and modularity of Python, new modules are readily available in the library *(User interface level)* through its programming layers, illustrated on Figure 9b. In order to add his modules written in a common programming language *(Computations level)*, the user needs to develop a small Python interface. The module may use a parallel paradigm *(HPC level)*. This design of the library allowed us to easily verify and optimize all the kernels presented below.

5.2. Implementation on GPU

The literature contains several GPU implementations of mesh-free particle methods. In the context of Smoothed Particles Hydrodynamics (SPH) for gas dynamics, [30] presents an implementation on multi-GPUs cluster for 32 millions of particles distributed on 4 GPUs. A mesh-free solver for incompressible flows using a Fast Multipole Method accelerated on a large number GPU is presented in [32]. These authors report reference a 0.5 petaflop/s computation for 2048^3 particles distributed on 2048 cards for homogeneous isotropic turbulence simulations.

The advection equation is a key component in the simulation of gas dynamics. In [31], the authors show that porting the resolution of Knudsen gases by particle methods to the GPGPU results in an important speedup in the simulations. Remeshed vortex methods have also been ported on GPU within the Parallel Particle Mesh library [28]. The implementation presented in [26] is devoted to two-dimensional incompressible flows in vorticity formulation, using a multigrid finite-difference solver for the Poisson equation and the $\Lambda_{2,1}$ remeshing formula for the advection. This reference reports performances in single precision floating point from 25 frames/iterations per seconds (FPS) with 1024^2 particles – for their *fastest solver*, Euler time stepping and coarse multigrid – to 3 FPS – for their *most accurate*, fourth order Runge−Kutta time stepping and accurate multigrid. In [24], penalization was added to deal with complex boundaries and a FFT solver was used instead of the multi-grid solver. The hand coded interpolation between the Cartesian grid and the particles was dropped in favor of OpenGL shaders with improved performances. For 1024^2 particles the full Navier–Stokes solver reached 22 FPS in single precision, which represents a speedup against a similar CPU solver of about 30.

As particle-mesh interpolations are core functions in RPM, some works have been devoted to these operations. In [3, 25] an implementation based on OpenCL Images objects, allowed to obtain, for a 4096×2048 problem size, a maximum speedup of 45 against a single-threaded CPU implementation, in double precision, and 150 in single precision. The speedup drops to 2.8 when comparing to a multi-threaded CPU version (9.4 in single precision). In [3], an efficient data structure enables interesting performances on GPU for tensorial particle-mesh interpolations in 2D and 3D. This reference reports that linear interpolation exhibits lower speedups than higher order schemes and that 3D interpolations are more efficient than their 2D counterparts.

The method presented in this article differs from the previous works by the fact that we use a dimensional splitting and higher order remeshing formulas. The dimensional splitting reduces 2D and 3D problems to a collection of 1D problems. The complexity will therefore remain linear with the stencil width, but this strategy has some implications that we will see below in the memory management.

In this work, we intend to produce an efficient GPU implementation of the particle advection and remeshing steps that will be used, in future work, as part of a fluid solver. The following results will demonstrate that high order remeshing formulas can be implemented in an efficient way. As in the previous section, particles are advanced with a fourth order Runge−Kutta method. The velocity field is given at grid points at the beginning of each time-step and is linearly interpolated at particle locations corresponding to the intermediate Runge−Kutta sub-steps. We use a second order time-splitting which for each time-step alternates advection successively along the directions x, y, z, y, x. The time-step of the advection is $\Delta t/2$ in the x and y directions and Δt in the z direction.

5.2.1. *From method to algorithm*

In order to better understand the library performances and behavior with respect to the number of particles, we analyze the computational complexity of the algorithm. The complexity C is given for one advection and one remeshing step for N^d particles, where N is the number of particles in one direction and d is the number of directions. C depends on several parameters: c, the number of scalar quantities or vector components carried by particles; the order of the dimensional splitting (second order in our case); N_s, the remeshing stencil width;

and p, the polynomial degree of the formula:

$$C(N_s,p,c) = (C_A + C_R(N_s,p,c))(2d-1)N^d = O\left(N^d\right) \text{ [Operations]}, \tag{5.1}$$

where C_A and C_R represent the advection and remeshing complexity for one particle. Advection with a fourth order Runge−Kutta scheme and linear velocity interpolation leads to the following estimate

$$C_A = \underbrace{4\times 2+3}_{\text{positions computation}} + \underbrace{3\times 9}_{\text{linear interpolations}} + \underbrace{5}_{\text{weights combination}} = 43 \text{ [Operations]}. \tag{5.2}$$

The complexity of remeshing algorithms takes into account the polynomial evaluations, performed with the Horner's rule rearrangements. The total amount of operations corresponds to N_s polynomials of degree p, c linear combinations of the N_s elements and the N_s grid point indices.

$$C_R(N_s,p,c) = \underbrace{N_s(2p+1)}_{\text{Horner's rule}} + \underbrace{2N_sc}_{\text{linear combinations}} + \underbrace{3+N_s}_{\text{grid indices}} = 2N_s(p+c+1)+3 \text{ [Operations]}. \tag{5.3}$$

Similarly, we study the memory access complexity. We obtain the following evaluations

$$M_A = (\underbrace{3\times 2+1}_{\text{read velocity}} + \underbrace{1}_{\text{write particle position}})P = 8P \text{ [Bytes]}, \tag{5.4}$$

$$M_R(N_s,c) = (\underbrace{1}_{\text{read particle position}} + \underbrace{(1+N_s)c}_{\text{read and write particle quantity}})P = ((1+N_s)c+1)P \text{ [Bytes]}, \tag{5.5}$$

$$D(c) = \underbrace{2cP}_{\text{read and write}} \text{ [Bytes]}, \tag{5.6}$$

$$M(N_s,c) = (D(c)+M_A+M_R(N_s,c))(2d-1)N^d = P((1+N_s)c+5)(2d-1)N^d \text{ [Bytes]}, \tag{5.7}$$

where P equals to the size of a floating point number in bytes (4 bytes for single precision or 8 bytes for double precision). $D(c)$ denotes the memory complexity of a copy or a transposition. In this expression, reading and writing access are supposed to be equivalent.

These complexities do not take into account the algorithms optimization for targeted platforms. For instance, vectorized access or *fma* operations are not considered here.

As already mentioned semi-Lagrangian particle methods can be viewed as an alternative to forward semi-Lagrangian methods developped in [9]. The use of explicit remeshing kernels significantly increase the parallel efficiency of the method. In the method described in [9], one needs to solve for each 1D problem a periodic tridiagonal system. [33] compares various implementations for solving tridiagonal systems on GPU. They show that a cyclic reduction algorithm requires at least $\log_2 N-2$ algorithmic steps and $5N$ accesses to global memory. For the $\Lambda_{4,2}$ formula, with stencils using 6 points, the remeshing algorithm only requires 1 algorithmic step, and $2N$ accesses to global memory. For problems with at least 64 particles in one direction, the semi-Lagrangian particle methods proposed in this paper require minimal access to global and shared memories.

5.2.2. From algorithm to GPU implementation

The choice of the OpenCL execution model constrained us to an in-order OpenCL kernels call sequence. The main bottleneck of this model is the host-device data transfer rate. The data should stay on the device as long as possible. However, host-device transfers are mandatory for regular checkpointing and shared computations between CPU and GPUs or multiple GPUs. The index space for calling OpenCL kernels follows the intrinsic dimension decomposition of the method: each work-group is in charge of a 1D problem. The kernels performances are strongly connected to the layout of work-items in each work-groups. Section 5.3 will detail the different strategies for the work-items layout.

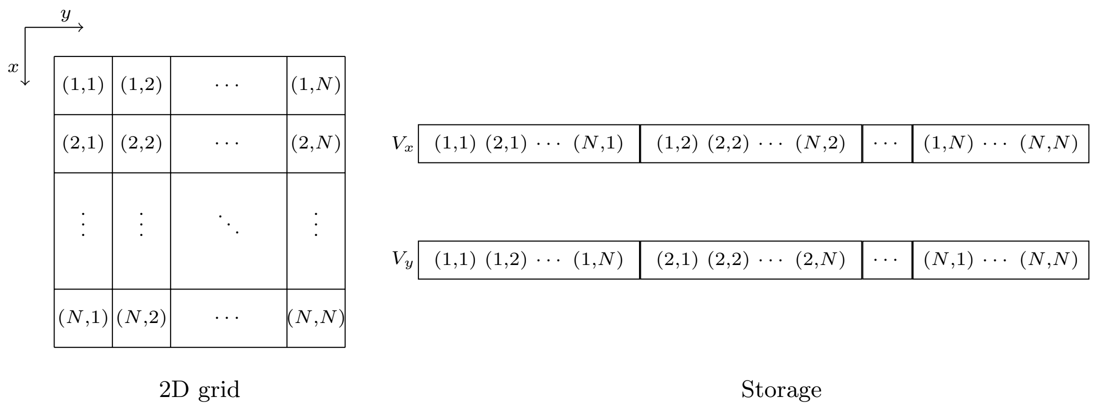


FIGURE 10. Velocity field storage.

The computation of one splitting step requires only the velocity component and the 1D coordinate of the particles in the corresponding direction. The velocity field needs to be stored as structured arrays. The maximum bandwidth for accessing global memory is reached for contiguous data. Arrays are thus stored contiguously in the direction of use. Figure 10 illustrates the storage of the velocity components with respect to the grid layout for a 2D problem. This setup imposes the advected quantities to be transposed when alternating splitting directions.

Algorithm 1 summarizes the GPU implementation, where each part of the inner loop is performed using an OpenCL kernel.

Algorithm 1: Particle method.

```
while $t \leqslant T_{end}$ do
    Computation of velocity field $a_G$ {on grid G} ;
    {Solve 1D problems in each direction};
    foreach splitting direction do
        Initialize particles ;
        Compute particle positions {from corresponding velocity component} ;
        Compute particle remeshing {in corresponding direction to grid G};
    $t \leftarrow t + dt$
```

5.3. Description of the OpenCL kernels

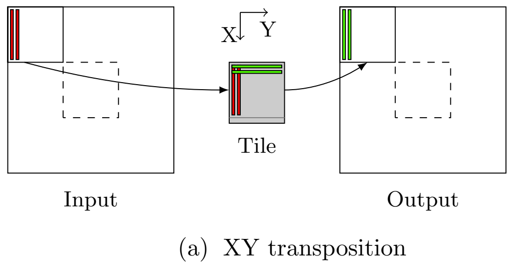
All our tests were performed on a NVIDIA Tesla K20m. The main specifications of this device are given in Table 2. Obviously, the global and local memory sizes and work-group size set some limits in our performance analysis. Each kernel used in Algorithm 1 must be studied separately to obtain the most efficient execution model characteristics such as *e.g.* work-group size, work-item workload and memory levels usage. We considered single and double precision implementations of the various remeshing kernels considered in the previous sections. Depending on the type of kernels, different optimizations – concerning OpenCL vector types, built-in functions, temporary private variables, bank conflicts avoidance, ... – were tested.

5.3.1. Memory management related to method and hardware

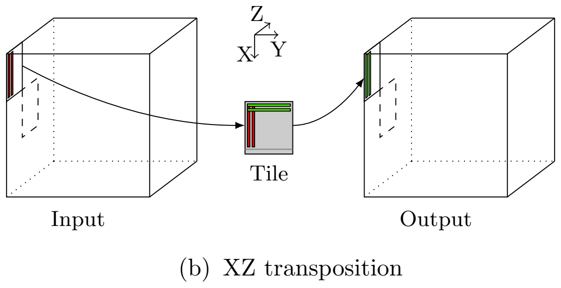
From the data on the grid, we update the advected quantities on particles. Data initialization depends on the alternating splitting direction in which advection and remeshing are performed. Computational efficiency requires the data to be contiguous in the working direction. Therefore, one needs to change the data layout in memory for each direction. Data initialization consists in transpositions except in the cases of unchanged

TABLE 2. NVIDIA Tesla K20m main features.

Device	OpenCL version	1.1	Global memory	Size	4 GB
	Compute units	13		Bandwidth	208 GB/s
	Maximum work-items	[1024,1024,64]		Bus width	320 bit
	Work-group size	1024		Max. allocation	1 GB
	Clock rate	705 MHz	Local memory	Size	48 KB
Host	Connection type	PCIe 2.1 $\times 16$	Processing power	Double Precision	3.52 TFlops
	Bandwidth	8 GB/s		Single Precision	1.17 TFlops



(a) XY transposition



(b) XZ transposition

FIGURE 11. Transposition kernels.

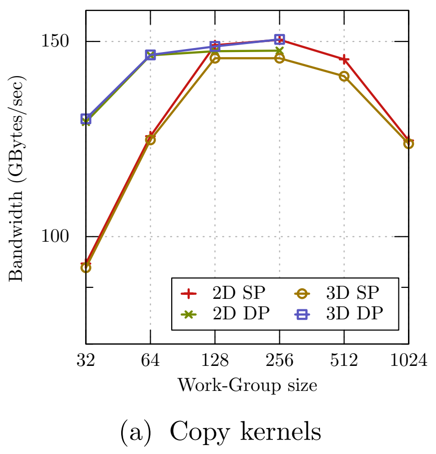
successive directions where a simple copy is sufficient. For a 2D problem, efficient transposition algorithms on GPU have been designed in [27]. This algorithm reaches 96 percent performance of the simple copy bandwidth for a 2048^2 square matrix in single precision. We implemented this algorithm and extended it to 3D. In order to achieve good performances, global memory accesses should be contiguous, fit the bus width and avoid memory camping. Memory camping occurs when successive work-groups operate on the same global memory page. It affects global efficiency of the algorithm and should be avoided as much as possible. Each work-group is handling a subset of the data, called a tile, which is stored in local memory. The tile is transposed locally, requiring bank conflict free access.

We denote by XY transposition the transposition operating on the first and second directions in which the data are contiguous. Similarly, XZ transposition operates on the first and third directions.

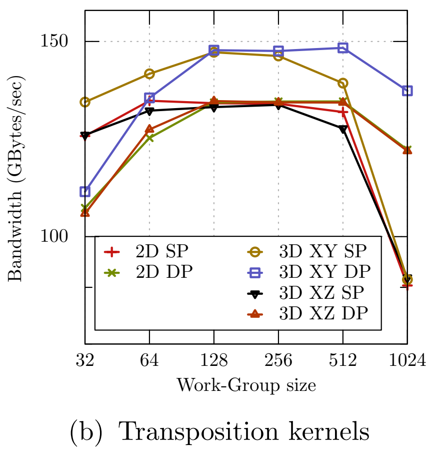
The XY transposition follows [27] except that, in 3D, it is performed on every slices along Z direction. Figure 11a illustrates this transformation. Each work-group loads a tile (grey square) into local memory (red). Each work-item transposes its own subset of data by writing data to their correct positions (green). Contiguous access in global memory is ensured by use of the tile and the bus width is filled up by wrapping elements into OpenCL vector types. Memory camping is avoided by roaming the global memory arrays diagonally – in dashed lines in the Figure. Finally a one row padding is used in tiles to avoid bank conflicts.

Similarly, XZ transpositions are performed along slices in Y direction. The work of [27] is extended to deal with 3D data. Using the same conventions as previously, Figure 11b sketches this transformation. To avoid memory camping, the global memory is roamed in XZ planes. Because of the larger strides when going along Z direction than in Y, we used a cubic tile to reduce the number of large strides.

Performances of the kernels associated to copy and transposition are presented on Figures 12a and 12b. These results concern fully optimized kernels and the best configurations. From Figure 12a we conclude that using 256 work-items in work-groups lead to performances close to 72 percent of the theoretical bandwidth. The remaining gap can be explained by the presence of the NVIDIA Error Correction Code (ECC) that checks and eventually corrects bit errors in data storage and transmissions. The performances should increase when disabling ECC



(a) Copy kernels



(b) Transposition kernels

FIGURE 12. Performances of Copy, XY and XZ transpositions kernels for 2D and 3D, single (SP) and double precision (DP).

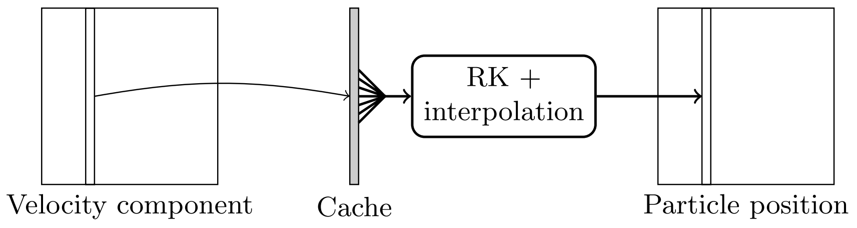


FIGURE 13. Advection algorithm.

because it uses a significant part of the bandwidth and 12.5 percent of the memory[2] to work. One can notice that the performance of the XY transposition is better in 3D than in 2D, very close to the performance of the copies for work-group size up to 512. This is due to smaller stride when moving work-group diagonally in 3D. In the optimal configuration, with respect to the work-group size, the XZ transposition exhibits performances similar to the 2D XY transposition.

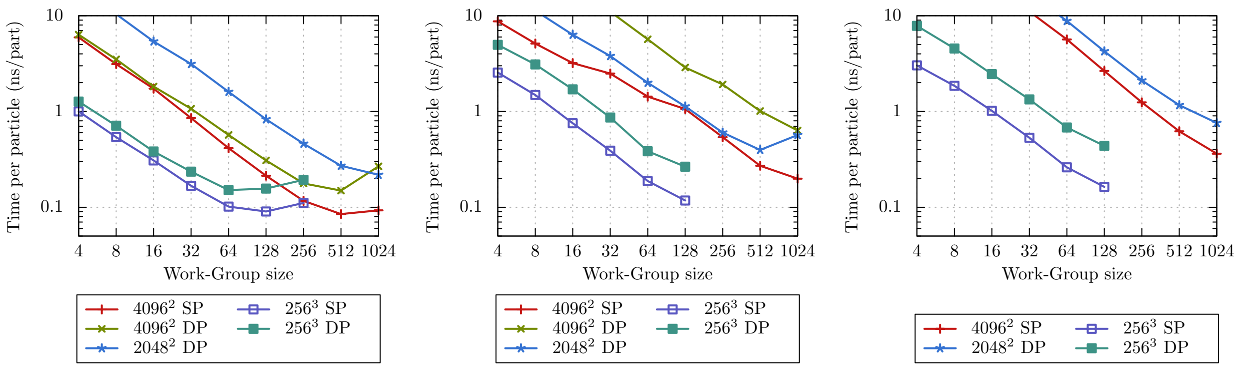
5.3.2. Advection

This kernel computes the particles positions by means of a Runge−Kutta scheme and a 2D or 3D linear interpolation of velocities on particles from known grid values. As suggested by (5.4), the fourth order Runge−Kutta method requires 8 reads in the velocity field. We bring it to a single read using local memory as a cache as illustrated on Figure 13. After filling up the cache, each work-group performs the computations from this local buffer. Results are directly stored contiguously in global memory. Using local memory, (5.4) becomes:

$$M_{A,\mathrm{GPU}} = 2P \text{ [Bytes]}. \tag{5.8}$$

Benchmarks for this kernel are shown on Figure 14a. The maximum work-group size allowed on the device is 1024, as given in Table 2. Performances increase with the work-group size until they reach a plateau around 128 work-items in 3D and 512 in 2D. Note that constraints on the global memory size and the bus width limit 3D cases to small work-group sizes.

[2]From `http://www.nvidia.com/object/tesla-servers.html`



(a) Advection kernels (RK4) (b) Remeshing kernels ($\Lambda_{2,1}$) (c) Monolithic kernels (RK4 + $\Lambda_{2,1}$)

Figure 14. Benchmarks for computational kernels in single (SP) and double precision (DP).

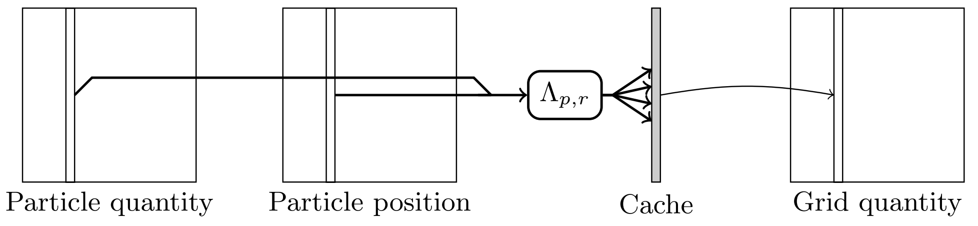


Figure 15. Remeshing algorithm.

5.3.3. Remeshing

Remeshing particles on the grid involves particle positions and strengths that are read once in global memory. Each work-group computes new quantities on the grid in a local buffer to avoid multiple writes in global memory, as shown on Figure 15. Again, the number of global memory accesses decreases and (5.5) becomes:

$$M_{R,\mathrm{GPU}} = (2c+1)P \text{ [Bytes]}. \tag{5.9}$$

In case of large velocity gradients, it may happen that several particles have the same remeshing points. To avoid concurrent writings, computational load is organized so that any pair of work-items do not remesh contiguous particles.

As for the advection benchmarks, Figure 14b shows that performances increase with the work-group size until it reaches a plateau.

5.3.4. Monolithic advection and remeshing

This kernel combines the advection and the remeshing computations in a single kernel. The particle positions are computed directly by the kernel, which reduces global memory occupation. The number of lunched OpenCL kernels is also reduced. However, it requires more local memory, as illustrated on Figure 16. For this kernel, the global memory complexity, given by (5.7) is reduced to:

$$M_{\mathrm{GPU,Monolithic}}(N_s, c) = P(2c+1)(2d-1)N^d \text{ [Bytes]} \tag{5.10}$$

instead of:

$$M_{\mathrm{GPU}}(N_s, c) = P(2c+3)(2d-1)N^d \text{ [Bytes]} \tag{5.11}$$

for separate kernels.

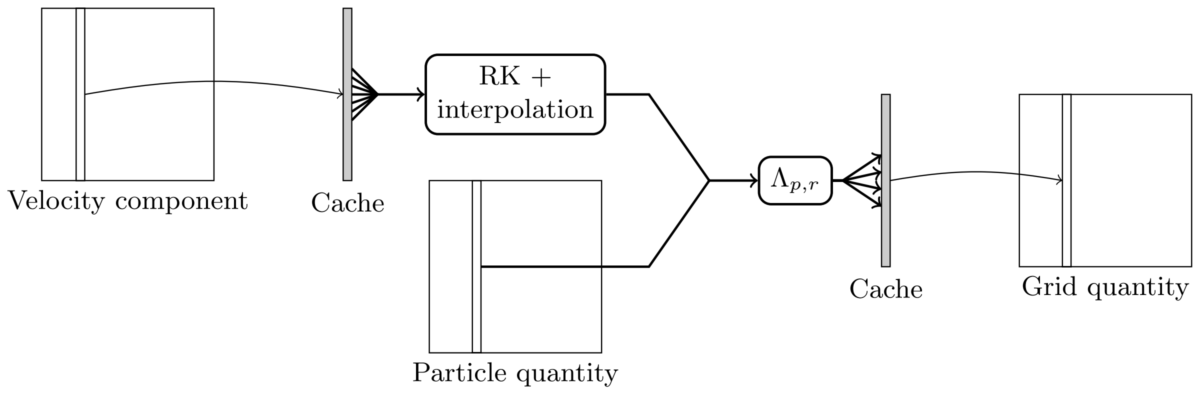


FIGURE 16. Monolithic advection and remeshing algorithm.

Again similar performances are obtained for monolithic kernel where results are close to the sum of the separate ones. Because of local memory limitations to 48 kBytes, we were not able to compute the 4096^2 double precision case (which would require 64 kBytes).

5.4. Performances

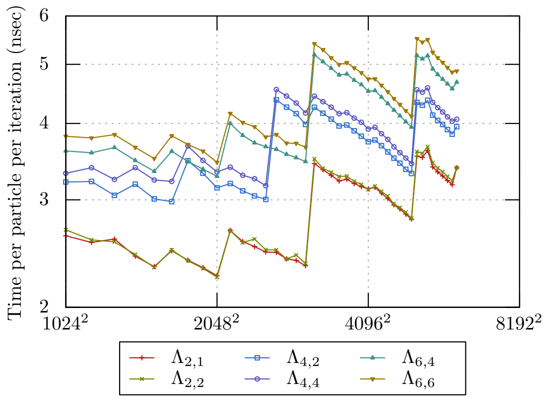
The whole code is profiled with the best configurations determined by the previous benchmarks. In this section, for a sake of simplicity we have restricted the discussion to calculations with double precision arithmetic.

Figure 17 gives the global performances for 2D and 3D problems and different remeshing formulas. The measured time includes kernel execution and OpenCL host code such as queue management, kernels launching, profiling events, ... Figure 17a shows that the time needed to compute one step for one particle is nearly constant, especially for small problems. This is consistent with (5.1), since the complexity per particle and per iteration, $(C_A + C_R(N_s, p, c))(2d - 1)$, does not depend on the problem size. The variations are due to built-in mechanisms of the device depending on the memory occupancy. In 3D, Figure 17a shows that, except for the 64^3 resolution, the computational time is also constant.

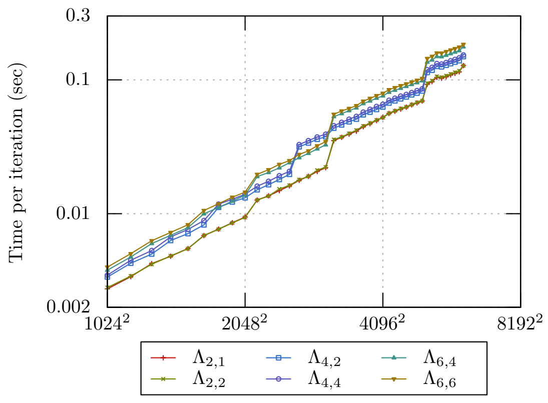
Similarly, the global complexity, $C(N_s, p, c)$, in (5.1) grows linearly with the problem size, N^d. This linearity is observed for the time per iteration on Figures 17b and 17d. For the 256^3 and 4096^2 resolutions, performances range from 10 to 20 FPS.

We provide on Figure 18 profiling details for the largest cases, corresponding to the 256^3 and 4096^2 particles. As expected, most of the time is devoted to particle remeshing. Remeshing times are grouped with respect to the stencil width. Table 3 exhibits the remeshing computational time per particle averaged – from data on Figures 17a and 17c – on different problem sizes together with the arithmetic complexity – from (5.3) – and the stencil width for the different remeshing formulas. In the *Ratio* columns, we compute the costs using the $\Lambda_{2,1}$ formula as a reference. We observe that the cost of remeshing depends mostly on the stencil width and only slightly on the polynomial computational complexity. The reason is that memory accesses are a bottleneck for remeshing kernels and this confirms the efficiency of GPU to accelerate algorithms with high computational intensity. The number of arithmetic operations plays only a marginal role in performances. Note that if the traditional tensor-product formulas were used in 3D, instead of the directional splitting used here, the ratio between the kernels $\Lambda_{6,6}$ and $\Lambda_{2,1}$ would be of order 8 instead of 2.

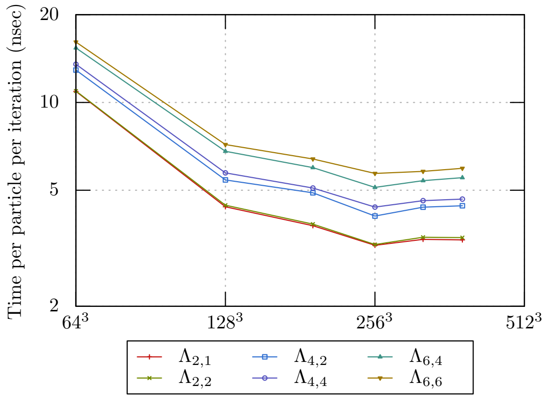
The dimensional splitting is also advantageous from the point of view of memory access. The number of operations required for a tensorial formula is of order N_s^3 instead of $5N_s$ for the splitting. In a tensor-product remeshing algorithm, $2N_s^3$ particles would need to be transferred to and from the global memory. Cache optimization allows to reduce the number of transfers per particle, but due to the limited size of the shared memory, this reduction is limited. For instance, with the GPU described in Table 2, one can only load 12^3 particles in



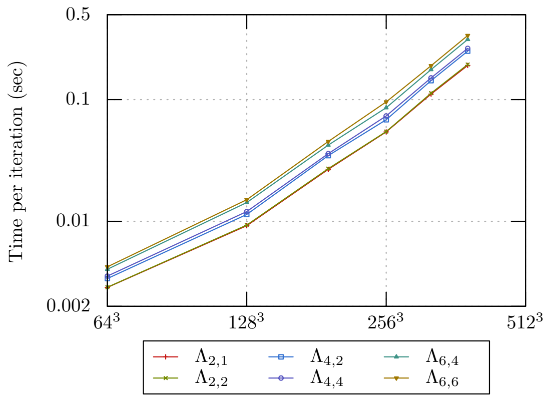
(a) Time per particle and per iteration (2D)



(b) Time per iteration (2D)



(c) Time per particle and per iteration (3D)



(d) Time per iteration (3D)

FIGURE 17. Overall performances for different problem sizes in double precision. (lower is better). Times are averaged over 20 independent executions.

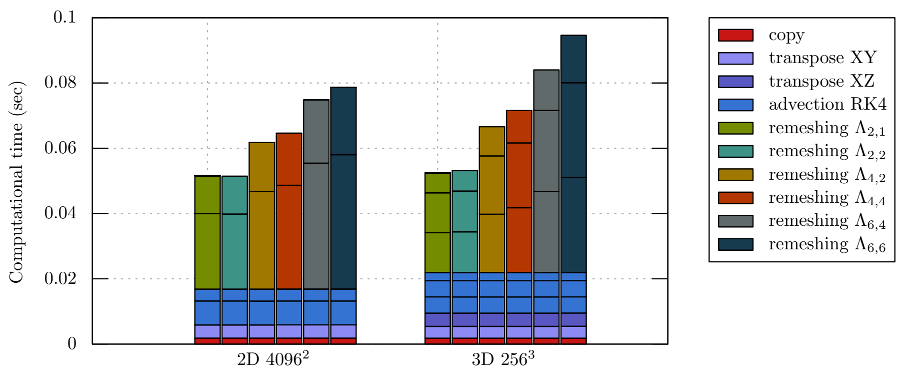


FIGURE 18. Code profiling of one time step for different problem sizes in double precision.

TABLE 3. Influence of remeshing formulas on performances. Times are averaged among the different runs reported in Figure 17b for the 2D cases and Figure 17c for the 3D cases.

	Arithmetics		Stencil		2D Time		3D Time	
Formula	$C_R(N_s, p, c)$	Ratio	Points	Ratio	Average	Ratio	Average	Ratio
$\Lambda_{2,1}$	43	1	4	1	3.13	1.00	3.88	1.00
$\Lambda_{2,2}$	59	1.37	4	1	3.17	1.01	3.99	1.03
$\Lambda_{4,2}$	87	2.02	6	1.5	4.22	1.35	5.81	1.50
$\Lambda_{4,4}$	135	3.14	6	1.5	4.49	1.43	6.28	1.62
$\Lambda_{6,4}$	179	4.16	8	2	5.22	1.67	7.88	2.03
$\Lambda_{6,6}$	243	5.65	8	2	5.62	1.79	8.78	2.26

shared memory. This limitation would cause the number of transfers with global memory performed at each particle location to increase. This would significantly impact the overall performance of the remeshing step.

Our benchmarks showed that the computational time remains linear with the stencil width. Increasing the order of the numerical method enables us to leverage the processing power of the GPU. From an applicative point of view, the accuracy of the results is improved either by increasing the problem size or increasing the numerical order of the method while keeping the problem size constant, which allows to relax the constraints due to the global memory size on the GPU. Note that the remeshing formulas involve algebraic fractions for the polynomials coefficients whose numerator and denominator are increasing with the order. In the present implementation, we kept this coefficients under their rational form but this could introduce numerical errors if high order kernels are used.

Unlike in [3], for an equivalent number of particles our 3D and 2D implementations exhibit similar performances, as they both rely on 1D problems. The global efficiency of our implementations could be further improved by gathering 1D problems to better fit the device characteristics. To conclude this section and give an idea of the compared efficiency of GPU and multi-threaded CPU implementations of the particle method, let us mention that the algorithm just described yields speedups ranging between 20 and 25, depending on the remeshing formula, against the optimized Fortran MPI code used in [15] running on 8 `Xeon E5-2640` cores.

6. Conclusion

In this paper we have discussed and analyzed remeshed particle methods, from the point of view of forward semi-Lagrangian methods. We have given a systematic consistency analysis of these methods, based on the regularity and moment properties of the remeshing kernels. These consistency results are supplemented by a stability analysis valid for a class of second order and fourth order methods, under a Lagrangian CFL condition which is independent of the grid size.

Remeshed particle methods are designed to deal with advection dominated problems and we have outlined a general computational framework in which they can be combined with other numerical methods for hybrid calculations. In this context, GPU are ideally suited to the local nature of RPM. We have presented a strategy to optimize the efficiency of RPM on GPU and discussed the complexity of theses algorithms in function of the kernel accuracy. Our results show in particular that, for a given number of conserved moments, increasing the smoothness of the kernels can improve the accuracy of the RPM without added computational complexity. The performance of the algorithms for a linear transport equation in double precision ranges between 10 and 20 FPS for grid resolutions of 4096^2 in 2D or 256^3 in 3D resolutions and kernels with order of accuracy varying between 1 and 6.

Although the focus in this paper was the implementation on GPU of particle methods, the framework we have described is designed with hybrid architectures in mind. Multi-CPUs and multi-GPUs implementations of semi-Lagrangian particle methods are the topic of ongoing work.

Appendix A

For a sake of completeness, we give here the analytical formulas for the kernels used in this work.

$$\Lambda_{2,1}(x) = \begin{cases} 1 - \frac{5}{2}|x|^2 + \frac{3}{2}|x|^3 & 0 \leqslant |x| < 1 \\ 2 - 4|x| + \frac{5}{2}|x|^2 - \frac{1}{2}|x|^3 & 1 \leqslant |x| < 2 \\ 0 & 2 \leqslant |x| \end{cases}$$

$$\Lambda_{2,2}(x) = \begin{cases} 1 - |x|^2 - \frac{9}{2}|x|^3 + \frac{15}{2}|x|^4 - 3|x|^5 & 0 \leqslant |x| < 1 \\ -4 + 18|x| - 29|x|^2 + \frac{43}{2}|x|^3 - \frac{15}{2}|x|^4 + |x|^5 & 1 \leqslant |x| < 2 \\ 0 & 2 \leqslant |x| \end{cases}$$

$$\Lambda_{4,2}(x) = \begin{cases} 1 - \frac{5}{4}|x|^2 - \frac{35}{12}|x|^3 + \frac{21}{4}|x|^4 - \frac{25}{12}|x|^5 & 0 \leqslant |x| < 1 \\ -4 + \frac{75}{4}|x| - \frac{245}{8}|x|^2 + \frac{545}{24}|x|^3 - \frac{63}{8}|x|^4 + \frac{25}{24}|x|^5 & 1 \leqslant |x| < 2 \\ 18 - \frac{153}{4}|x| + \frac{255}{8}|x|^2 - \frac{313}{24}|x|^3 + \frac{21}{8}|x|^4 - \frac{5}{24}|x|^5 & 2 \leqslant |x| < 3 \\ 0 & 3 \leqslant |x| \end{cases}$$

$$\Lambda_{4,4}(x) = \begin{cases} 1 - \frac{5}{4}|x|^2 + \frac{1}{4}|x|^4 - \frac{100}{3}|x|^5 + \frac{455}{4}|x|^6 - \frac{295}{2}|x|^7 + \frac{345}{4}|x|^8 - \frac{115}{6}|x|^9 & 0 \leqslant |x| < 1 \\ -199 + \frac{5485}{4}|x| - \frac{32975}{8}|x|^2 + \frac{28425}{4}|x|^3 - \frac{61953}{8}|x|^4 + \frac{33175}{6}|x|^5 & \\ \qquad - \frac{20685}{8}|x|^6 + \frac{3055}{4}|x|^7 - \frac{1035}{8}|x|^8 + \frac{115}{12}|x|^9 & 1 \leqslant |x| < 2 \\ 5913 - \frac{89235}{4}|x| + \frac{297585}{8}|x|^2 - \frac{143895}{4}|x|^3 + \frac{177871}{8}|x|^4 - \frac{54641}{6}|x|^5 & \\ \qquad + \frac{19775}{8}|x|^6 - \frac{1715}{4}|x|^7 + \frac{345}{8}|x|^8 - \frac{23}{12}|x|^9 & 2 \leqslant |x| < 3 \\ 0 & 3 \leqslant |x| \end{cases}$$

$$\Lambda_{6,4}(x) = \begin{cases} 1 - \frac{49}{36}|x|^2 + \frac{7}{18}|x|^4 - \frac{3521}{144}|x|^5 + \frac{12029}{144}|x|^6 - \frac{15617}{144}|x|^7 + \frac{1015}{16}|x|^8 - \frac{1015}{72}|x|^9 & 0 \leqslant |x| < 1 \\ -\frac{877}{5} + \frac{72583}{60}|x| - \frac{145467}{40}|x|^2 + \frac{18809}{3}|x|^3 - \frac{54663}{8}|x|^4 + \frac{390327}{80}|x|^5 & \\ \qquad - \frac{182549}{80}|x|^6 + \frac{161777}{240}|x|^7 - \frac{1827}{16}|x|^8 + \frac{203}{24}|x|^9 & 1 \leqslant |x| < 2 \\ 8695 - \frac{656131}{20}|x| + \frac{3938809}{72}|x|^2 - \frac{158725}{3}|x|^3 + \frac{2354569}{72}|x|^4 - \frac{9644621}{720}|x|^5 & \\ \qquad + \frac{523589}{144}|x|^6 - \frac{454097}{720}|x|^7 + \frac{1015}{16}|x|^8 - \frac{203}{72}|x|^9 & 2 \leqslant |x| < 3 \\ -\frac{142528}{5} + \frac{375344}{5}|x| - \frac{3942344}{45}|x|^2 + \frac{178394}{3}|x|^3 - \frac{931315}{36}|x|^4 + \frac{5385983}{720}|x|^5 & \\ \qquad - \frac{1035149}{720}|x|^6 + \frac{127511}{720}|x|^7 - \frac{203}{16}|x|^8 + \frac{29}{72}|x|^9 & 3 \leqslant |x| < 4 \\ 0 & 4 \leqslant |x| \end{cases}$$

$$\Lambda_{6,6}(x) = \begin{cases} 1 - \frac{49}{36}|x|^2 + \frac{7}{18}|x|^4 - \frac{1}{36}|x|^6 - \frac{46109}{144}|x|^7 + \frac{81361}{48}|x|^8 - \frac{544705}{144}|x|^9 + \frac{655039}{144}|x|^{10} & \\ \qquad - \frac{223531}{72}|x|^{11} + \frac{81991}{72}|x|^{12} - \frac{6307}{36}|x|^{13}, & 0 \leqslant |x| < 1 \\ -\frac{44291}{5} + \frac{1745121}{20}|x| - \frac{15711339}{40}|x|^2 + \frac{32087377}{30}|x|^3 - \frac{7860503}{4}|x|^4 + \frac{38576524}{15}|x|^5 & \\ \qquad - \frac{24659323}{10}|x|^6 + \frac{84181657}{48}|x|^7 - \frac{74009313}{80}|x|^8 + \frac{17159513}{48}|x|^9 & \\ \qquad - \frac{7870247}{80}|x|^{10} + \frac{438263}{24}|x|^{11} - \frac{81991}{40}|x|^{12} + \frac{6307}{60}|x|^{13}, & 1 \leqslant |x| < 2 \\ 3905497 - \frac{424679647}{20}|x| + \frac{3822627865}{72}|x|^2 - \frac{2424839767}{30}|x|^3 + \frac{3009271097}{36}|x|^4 & \\ \qquad - \frac{930168127}{15}|x|^5 + \frac{305535494}{9}|x|^6 - \frac{9998313437}{720}|x|^7 + \frac{203720335}{48}|x|^8 - \frac{137843153}{144}|x|^9 & \\ \qquad + \frac{22300663}{144}|x|^{10} - \frac{6126883}{360}|x|^{11} + \frac{81991}{72}|x|^{12} - \frac{6307}{180}|x|^{13}, & 2 \leqslant |x| < 3 \\ -\frac{255622144}{5} + \frac{971097344}{5}|x| - \frac{15295867328}{45}|x|^2 + \frac{5442932656}{15}|x|^3 - \frac{2372571796}{9}|x|^4 & \\ \qquad + \frac{2064517469}{15}|x|^5 - \frac{9563054381}{180}|x|^6 + \frac{2210666335}{144}|x|^7 - \frac{796980541}{240}|x|^8 & \\ \qquad + \frac{76474979}{144}|x|^9 - \frac{43946287}{720}|x|^{10} + \frac{343721}{72}|x|^{11} - \frac{81991}{360}|x|^{12} + \frac{901}{180}|x|^{13} & 3 \leqslant |x| < 4 \\ 0 & 4 \leqslant |x| \end{cases}$$

$$\Lambda_{8,4}(x) = \begin{cases} 1 - \frac{205}{144}x^2 + \frac{91}{192}x^4 - \frac{6181}{320}x^5 + \frac{6337}{96}x^6 - \frac{2745}{32}x^7 + \frac{28909}{576}x^8 - \frac{3569}{320}x^9 & 0 \leqslant |x| < 1 \\ -154 + \frac{12757}{12}x - \frac{230123}{72}x^2 + \frac{264481}{48}x^3 - \frac{576499}{96}x^4 + \frac{686147}{160}x^5 & \\ \qquad - \frac{96277}{48}x^6 + \frac{14221}{24}x^7 - \frac{28909}{288}x^8 + \frac{3569}{480}x^9 & 1 \leqslant |x| < 2 \\ \frac{68776}{7} - \frac{1038011}{28}x + \frac{31157515}{504}x^2 - \frac{956669}{16}x^3 + \frac{3548009}{96}x^4 - \frac{2422263}{160}x^5 & \\ \qquad + \frac{197255}{48}x^6 - \frac{19959}{28}x^7 + \frac{144545}{2016}x^8 - \frac{3569}{1120}x^9 & 2 \leqslant |x| < 3 \\ -56375 + \frac{8314091}{56}x - \frac{49901303}{288}x^2 + \frac{3763529}{32}x^3 - \frac{19648027}{384}x^4 + \frac{9469163}{640}x^5 & \\ \qquad - \frac{545977}{192}x^6 + \frac{156927}{448}x^7 - \frac{28909}{1152}x^8 + \frac{3569}{4480}x^9 & 3 \leqslant |x| < 4 \\ \frac{439375}{7} - \frac{64188125}{504}x + \frac{231125375}{2016}x^2 - \frac{17306975}{288}x^3 + \frac{7761805}{384}x^4 - \frac{2895587}{640}x^5 & \\ \qquad + \frac{129391}{192}x^6 - \frac{259715}{4032}x^7 + \frac{28909}{8064}x^8 - \frac{3569}{40320}x^9 & 4 \leqslant |x| < 5 \\ 0 & 5 \leqslant |x| \end{cases}$$

Acknowledgements. The authors are grateful to M. Bergdorf, D. Rossinnelli and P. Koumoutsakos for enlightening discussions on GPU implementation and the derivation of high order kernels. The first author gratefully acknowledges support from Institut Universitaire de France. This research has been partially supported by the Agence Nationale pour la Recherche (ANR) under Contracts No. ANR-2010-COSI-0009 and ANR-2010-JCJC-091601.

References

[1] M. Bergdorf, G.-H. Cottet and P. Koumoutsakos, Multilevel adaptive particle methods for convection-diffusion equations. *SIAM Multiscale Model. Simul.* **4** (2005) 328–357.

[2] M. Bergdorf and P. Koumoutsakos, A lagrangian particle-wavelet method. *SIAM Multiscale Model. Simul.* **5** (2006) 980–995.

[3] F. Büyükkeçeci, O. Awile and I. Sbalzarini, A portable opencl implementation of generic particle-mesh and mesh-particle interpolation in 2d and 3d. *Parallel Comput.* **39** (2013) 94–111.

[4] A. Chorin, Numerical study of slightly viscous flow. *J. Fluid Mech.* **57** (1973) 785–796.

[5] C. Cocle, G. Winckelmans and G. Daeninck, Combining the vortex-in-cell and parallel fast multipole methods for efficient domain decomposition simulations. *J. Comput. Phys.* **227** (2008) 9091–9120.

[6] C. Cotter, J. Frank and S. Reich, The remapped particle-mesh semi-lagrangian advection scheme. *Q. J. Meteorol. Soc.* **133** (2007) 251–260.

[7] G.-H. Cottet and P. Koumoutsakos, *Vortex methods*. Cambridge University Press (2000).

[8] G.-H. Cottet and L. Weynans, Particle methods revisited: a class of high order finite-difference methods. *C.R. Math.* **343** (2006) 51–56.

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

[10] R. Hockney and J. Eastwood, *Simulation Using Particles*. Inst. Phys. Publ. (1988).

[11] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov and A. Fasih, PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation. *Parallel Comput.* **38** (2012) 157–174.

[12] P. Koumoutsakos, Inviscid axisymmetrization of an elliptical vortex. *J. Comput. Phys.* **138** (1997) 821–857.

[13] P. Koumoutsakos and A. Leonard, High resolution simulation of the flow around an impulsively started cylinder using vortex methods. *J. Fluid Mech.* **296** (1995) 1–38.

[14] S. Labbé, J. Laminie and V. Louvet, Méthodologie et environnement de développement orientés objets: de l'analyse mathématique à la programmation. *MATAPLI* **70** (2003) 79–92.

[15] J.-B. Lagaert, G Balarac and G.-H. Cottet, Hybrid spectral particle method for turbulent transport of passive scalar. *J. Comput. Phys.* **260** (2014) 127–142.

[16] A. Leonard. Computing three-dimensional incompressible flows with vortex elements. *Annu. Rev. Fluid Mech.* **17** (1985) 523–559.

[17] R.J. LeVeque, High-resolution conservative algorithms for advection in incompressible flow. *SIAM J. Numer. Anal.* **33** (1996) 627–665.

[18] A. Magni and G.-H. Cottet, Accurate, non-oscillatory, remeshing schemes for particle methods. *J. Comput. Phys.* **231** (2012) 152–172.

[19] J. Monaghan, Extrapolating B splines for interpolation. *J. Comput. Phys.* **60** (1985) 253–262.

[20] J. Monaghan, An introduction to SPH. *Comput. Phys. Commun.* **48** (1988) 89–96.

[21] A. Munshi, The OpenCL Specification. *Khronos OpenCL Working Group* (2011).

[22] M. Ould-Salihi, G.-H. Cottet and M. El Hamraoui, Blending finite-difference and vortex methods for incompressible flow computations. *SIAM J. Sci. Comput.* **22** (2000) 1655–1674.

[23] T. Respaud and E. Sonnendruücker, Analysis of a new class of forward semi-lagrangian schemes for the 1d Vlasov-Poisson equations. *Numer. Math.* **118** (2011) 329–366.

[24] D. Rossinelli, M. Bergdorf, G.H. Cottet and P. Koumoutsakos, GPU accelerated simulations of bluff body flows using vortex methods. *J. Comput. Phys.* **229** (2010) 3316–3333.

[25] D. Rossinelli, C. Conti and P. Koumoutsakos, Mesh-particle interpolations on graphics processing units and multicorecentral processing units. *Philosophical Transactions of the Royal Society A: Mathematical, Phys. Engrg. Sci.* **369** (2011) 2164–2175.

[26] D. Rossinelli and P. Koumoutsakos, Vortex methods for incompressible flow simulations on the GPU. *Visual Comput.* **24** (2008) 699–708.

[27] G. Ruetsch and P. Micikevicius, Optimizing matrix transpose in cuda. *NVIDIA CUDA SDK Application Note* (2009).

[28] I. Sbalzarini, J. Walther, M. Bergdorf, S. Hieber, E. Kotsalis and P. Koumoutsakos, PPM–a highly efficient parallel particle–mesh library for the simulation of continuum systems. *J. Comput. Phys.* **215** (2006) 566–588.

[29] I. Schoenberg, Contribution to the problem of approximation of equidistant data by analytic functions. *Q. Appl. Math.* **4** (1946) 45–99.

[30] D. Valdez-Balderas, J. Dominguez, B. Rogers and A. Crespo, Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-gpu clusters. *J. Parallel Distrib. Comput.* **73** (2012) 1483–1493.

[31] F. De Vuyst and F. Salvarani, GPU-accelerated numerical simulations of the knudsen gas on time- dependent domains. *Comput. Phys. Commun.* **184** (2013) 532–536.

[32] R. Yokota, L. Barba, T. Narumi and K. Yasuoka, Petascale turbulence simulation using a highly parallel fast multipole method. *Comput. Phys. Commun.* **184** (2013) 445–455.

[33] Y. Zhang, J. Cohen and J.D. Owens, Fast tridiagonal solvers on the GPU. *SIGPLAN Not.* **45** (2010) 127–136.