An adaptive multiresolution Vortex Particle-Mesh method for the simulation of unbounded incompressible flows

Pierre Balty[a,*], Matthieu Duponcheel[a], Philippe Chatelain[a]

[a] *Institute of Mechanics, Materials and Civil Engineering, Université catholique de Louvain, Place du Levant 2, Louvain-la-Neuve, 1348, , Belgium*

Abstract

Simulations of incompressible convection-dominated flows, such as vortical flows or wake flows, require accurate and efficient numerical methods. Among the various approaches, the Vortex Particle-Mesh (VPM) method stands out for its ability to combine the strengths of mesh-based methods, such as efficient elliptic solvers and finite difference stencils, with the advantages of particle methods, which minimize numerical dispersion and dissipation errors. However, most VPM implementations consider a uniform grid. This can lead to a prohibitive computational cost, especially in three dimensions. This challenge can be addressed effectively through Adaptive Mesh Refinement (AMR), which significantly reduces simulation costs by dynamically adjusting the local grid resolution to meet the requirements of the underlying physics. In this work, we present a novel VPM method that leverages the benefits of AMR. Building upon `murphy` [1], a wavelet-based block-structured AMR framework, we propose a new approach to represent the particles that allows for their refinement or coarsening. We show that our method achieves high-order accuracy, grid adaptation, and CFL relaxation at the same time. We then demonstrate the scalability of our method from 128 to 16384 cores, and compare its performance with a uniform resolution framework for the simulation of aircraft wake vortices. We show that despite the increased complexity due to the overhead of the AMR operations, the resulting gains in computational efficiency and reductions in memory footprint are substantial.

Keywords: Vortex particle-mesh method, multiresolution analysis, adaptive mesh refinement, wavelet analysis, incompressible flows, vortex dynamics
2020 MSC: 68W10, 76D05, 76M23

1. Introduction

Vortical flows are at the heart of many science and engineering problems, from the wakes behind aircraft to the flow signature behind flying birds or swimming fish. Dominated by convection, such phenomena unfold over long times and large scales. This, hence, imposes a substantial amount of computational resources if one wants to simulate all the scales involved in their flow dynamics. It also requires enforcing free-space boundary conditions, which is challenging to achieve using traditional numerical approaches. In such a context, vortex methods [2, 3] stand out as an effective solution. They solve the velocity-vorticity formulation of the Navier-Stokes equation for incompressible flows, discretize the vorticity with vortex particles, and treat the advection term in a Lagrangian fashion. This mitigates the dispersion and diffusion errors and waives the usual *Courant-Friedrichs-Lewy* (CFL) constraints required for the stability of corresponding explicit

*Corresponding author
Email addresses: `pierre.balty@uclouvain.be` (Pierre Balty), `matthieu.duponcheel@uclouvain.be` (Matthieu Duponcheel), `philippe.chatelain@uclouvain.be` (Philippe Chatelain)

Eulerian discretizations. Additionally, the vorticity provides a compact representation of the entire flow field, naturally enabling the enforcement of unbounded boundary conditions. In recent years, hybrid techniques, known as Vortex-in-cell or Vortex Particle-Mesh (VPM) method, have taken over from traditional vortex methods. They retain the advantages of particle-based methods but combine them with a mesh to benefit from highly efficient Poisson solver and finite difference stencils. Particles and mesh exchange information through high-order interpolation kernels. The method was successfully applied to various bio-locomotion, wind energy, and aircraft wake cases [4, 5, 6, 7, 8, 9] and has been proven to be a fast and accurate alternative to pseudospectral solvers to simulate vortical flows [10, 11]. Despite its many advantages, the VPM method faces a fundamental challenge: its computational cost becomes prohibitive when it comes to $3D$, high Reynolds number flows which *de facto* exhibit strong disparity in their spatiotemporal scales. While recent efforts have focused on porting the method on Graphic Processing Units (GPU) and hybrid CPU-GPU architectures to leverage their computational power [12, 13, 14], others improved the method efficiency by adapting the density of computational elements to the physics that have to be captured. Our work falls within this category.

Initially, works have been done to improve the adaptivity of vortex methods using variable core size particles [15, 16, 17]. However, those approaches required an *a-priori* knowledge of the flow and were, therefore, not adaptive by nature. *Bergdorf et al.* [18] then introduced an Adaptive Global Mapping (AGM). The approach pertains to r-adaptivity in finite element methods, where the computational elements shift toward regions requiring higher resolution. AGM solves the governing partial differential equations (PDEs) in a reference space where all particles have a uniform core size. The solution is then mapped back to the physical space, where core sizes vary spatially based on physical requirements.

Beyond AGM, other studies leveraged the remeshing procedure introduced above and the grid in hybrid approaches to explore Adaptive Mesh Refinement (AMR) techniques initially developed for finite difference and finite element schemes. AMR methods dynamically adjust the local grid spacing to the physical scales, ensuring efficient resolution where needed. These approaches have the advantage of keeping a grid composed of regions with uniform and isotropic elements of different level, often exploited in VPM methods. Three prime prerequisites must be met: they must (1) detect the need for refinement and the opportunity to coarsen, (2) provide mechanisms to adapt the fields, and (3) handle the resolution jumps in all the involved numerical tools. Simultaneously, the computational overhead associated with the adaptation machinery must be minimized to preserve the effective gains made from reducing the number of computational elements.

Two approaches stand out for performing AMR: the patch-based adaptive mesh refinement (pAMR) technique and the octree-based method. The first, the pAMR approach, represents the domain as a union of nested overlapping grids of increasing resolution. The adaptation criteria are typically based on field features such as solution gradients and/or curvature, or estimates of truncation error, or a combination of both [19]. The actual refinement and coarsening and the evaluation of differential operators across resolution jumps are usually based on polynomial interpolation, and most methods achieve second-order accuracy in space. Several pAMR frameworks, such as `Chombo` [20], `SAMRAI` [21] or `AMReX` [22] have been recently developed, all targetting distributed architecture and building upon the original Berger-Oliger algorithm [23]. For incompressible flows, *Yu et al.* [24] proposed an adaptive lattice Green's function method that has been recently extended to account for immersed boundaries and new boundary conditions [25].

In contrast to patch-based approaches, octree-based methods represent the computational domain using a hierarchy of non-overlapping and recursively refined cells that are stored as octree leaves. As for pAMR approaches, several libraries have been designed for High performance computing (HPC) architecture [26, 27, 28]. Such a method has been further exploited in the finite volume code `Gerris` [29], later extended and renamed `Basilisk` [30]. Building on this framework, *van Hooft et al.* [31] introduced a new AMR incompressible flow solver achieving fourth-order accuracy in both space and time. *Rossinelli et. al.* [13], *Schornbaum and Rüde* [32], *Gillis and van Rees* [1] and *Jude et. al* [33] leveraged the octree approach to use the tree leaves not as a single cell but as a uniform-resolution block of fixed size. While this choice lowers the compression rate of the grid, it significantly enhances the computational performance, especially when working on GPUs.

The dyadic recursive structure of octree is naturally coupled with the wavelet-based multiresolution analysis [34, 1]. The wavelet transform is a very powerful tool that decomposes any signals into both space and scale. Wavelets are well-localized waveforms that present a fast decay at infinity and at least one vanishing moment (*i.e.*, zero mean). The scale decomposition is achieved by dilating or contracting the chosen analyzing wavelets before convolving them with the signal. In multiresolution analysis, the wavelets are used to generate two ensembles of nested, orthogonal subspaces that break down the signal into meaningful and negligible contributions. They can, thus, be employed to drive the adaptation of the grid. Wavelets have broad applications across various fields and are increasingly used in the CFD community, as illustrated in the review of *Schneider and Vasilyev* [35]. In such a context, *Engels et. al.* [36] combined a block-structured octree grid with bi-orthogonal wavelets to perform simulations of weakly compressible flows around moving bodies.

In the vortex method community, several works have leveraged pAMR techniques, and only a few have relied on octree-based approaches. In their paper on the AGM, *Bergdorf et al.* [18] also coupled a pAMR technique with particle methods. They solved the $2D$ incompressible Euler equations in a velocity-vorticity formulation with two levels of refinement, and were able to detect small structures in the vorticity field using a B-Spline-based high-pass filter. *Rasmussen et al.* [37] proposed a $2D$ multiresolution VPM method based on nested grids and included a Brinkman penalization. However, the nested grids were *a-priori* fixed and were not adapted to the flow characteristics. In $3D$, *Lonfils* [38] achieved patch-based grid refinement, high order accuracy, and CFL relaxation simultaneously, but the nested grids were also fixed *a priori*. *El Ossmani and Poncet* [39] studied, among other cases, the dynamics of a $3D$ ellipsoid vortex and the wake development behind a sphere using a multiresolution VPM method based on pAMR. The error estimation needed for the grid adaptation was driven by the periodic evaluation of $\alpha|\boldsymbol{\omega}| + \beta|\nabla \times \boldsymbol{\omega}|$, where $\boldsymbol{\omega}$ is the vorticity. The latter expression quantifies the smoothness of the flow. If it is larger than a user-defined threshold, the grid is refined. In $2D$, *Bergdorf et al.* [40] introduced the wavelet-based multiresolution analysis in hybrid particle-mesh methods and used it to dictate grid refinement. Building upon that work, *Rossinelli et al.* [13] developed a wavelet-based multiresolution VPM method that benefits from a block-structured octree-based approach to improve the method efficiency. The whole method was tailor-made for HPC architectures and showed substantial performance improvements while achieving equivalent accuracy compared to a uniform-mesh counterpart.

With the exception of the work by *Lonfils* [38], the other studies all lack a reliable and flexible procedure for the multiresolution interpolation of particle quantities on the mesh and vice-versa. Instead, they either impose a restrictive CFL-like constraint on the timestep, hence losing one of the advantages of the particle method, or they significantly expand the ghost layer of the different grids to ensure sufficient support for the interpolation near resolution jumps. The conventional interpolation kernel implicitly requires particles to have a size. When it comes to AMR VPM methods, one ideally requires the capability to refine or coarsen Lagrangian elements—a feature that, to the best of the authors' knowledge, has never been done before within AMR VPM methods.

In this work, we present a new $3D$ scalable, and efficient implementation of the VPM method that capitalizes on adaptive grid refinement techniques and supports the dynamic adaptation of the particles. Our solver is integrated within `murphy` [1], a wavelet-based multiresolution framework for scientific computing on $3D$ block-structured octree-based collocated grids. We propose a new mapping approach that connects the particles and enables their refinement or coarsening, thereby facilitating the interpolation between particles and a mesh of different resolutions. We employ a novel Multigrid approach with an FFT-based direct solver to solve the Poisson equation. Designed to leverage large-scale parallel architectures, the resulting software has been tested up to 16 384 CPUs.

This paper is organized as follows. The VPM method is briefly presented in Section 2. Then, the grid adaptation techniques employed, as well as the multiresolution hybrid particle-mesh framework, are discussed in Section 3. The rest of the paper is dedicated to the validation of the method in Section 5 and to the analysis of its performance in Section 6. This work is concluded in Section 7.

2. The Vortex Particle-Mesh mehod

The present contribution builds upon a corpus of works in the field of hybrid Vortex methods. We only present here the theory required to detail our multiresolution framework. A complete review of Vortex methods and their applications can be found in [41].

The Vortex Particle-Mesh method solves the velocity ($\boldsymbol{u}$)-vorticity ($\boldsymbol{\omega} = \nabla \times \boldsymbol{u}$) formulation of the incompressible ($\nabla \cdot \boldsymbol{u} = 0$) Navier-Stokes equations

$$\frac{D\boldsymbol{\omega}}{Dt} = (\boldsymbol{\omega} \cdot \nabla)\boldsymbol{u} + \nu \nabla^2 \boldsymbol{\omega}, \tag{1}$$

where $\frac{D}{Dt} \triangleq \left(\frac{\partial}{\partial t} + \boldsymbol{u} \cdot \nabla\right)$ denotes the Lagrangian derivatives and ν the kinematic viscosity. The velocity field is related to the vorticity through its Helmholtz decomposition, based on the stream and potential functions, $\boldsymbol{\Psi}$ and $\boldsymbol{\Phi}$, respectively:

$$\boldsymbol{u} = \nabla \times \boldsymbol{\Psi} + \nabla \boldsymbol{\Phi} \ , \tag{2}$$

typically with a gauge set on $\boldsymbol{\Psi}$ as $\nabla \cdot \boldsymbol{\Psi} = \boldsymbol{0}$. Due to the incompressibility of the flow, $\nabla^2 \boldsymbol{\Phi} = \nabla \cdot \boldsymbol{u} = 0$. In the absence of body in the flow, the potential component of the velocity, $\nabla \boldsymbol{\Phi}$, is hence often used to take a free-stream velocity $\boldsymbol{U}_\infty$ into account. On the other hand, the stream function is recovered from the vorticity based on a Poisson equation:

$$\nabla^2 \boldsymbol{\Psi} = -\boldsymbol{\omega} \ . \tag{3}$$

The vorticity field is discretized using Lagrangian particles, characterized by a material position $\boldsymbol{x}_p$, a volume V_p and a strength $\boldsymbol{\alpha}_p$. The strength represents the integral of $\boldsymbol{\omega}$ over the particle material volume: $\boldsymbol{\alpha}_p = \int_{V_p} \boldsymbol{\omega} dV_p$. The particles are convected by the local flow field, $\boldsymbol{u}_p$, and their strength is updated to account for the vorticity diffusion and the vortex stretching. The latter discretization results in a set of ordinary differential equations:

$$\frac{d}{dt}\begin{bmatrix}\boldsymbol{x}_p \\ \boldsymbol{\alpha}\end{bmatrix}_p = \begin{bmatrix}\boldsymbol{u} \\ \left((\boldsymbol{\omega} \cdot \nabla)\boldsymbol{u} + \nu \nabla^2 \boldsymbol{\omega}\right) V_p\end{bmatrix}_p \ . \tag{4}$$

The VPM framework takes advantage of a Cartesian mesh to improve its computational efficiency. Particle quantities are first interpolated onto the mesh, where finite differences are applied to compute the differential operators of the right-hand side of Eq. eq. (4) and where the velocity is recovered from the vorticity through the resolution of the Poisson equation, Equation (3). In a uniform resolution framework, the latter is efficiently solved using a Fourier-based solver such as those provided by `flups`, a versatile and efficient Fourier-based library of unbounded Poisson solvers [42, 43]. The updated quantities are then interpolated back onto the particles.

If left unchecked, Lagrangian distortion can lead to clustering of particles, or conversely depletion in some regions of the flow. This translates into a breakdown of the interpolation and quadrature operations behind Equation (4), and subsequently in the generation of spurious vortical structures [44, 45]. To ensure a homogeneous particle distribution and maintain the method accuracy over time, a redistribution scheme periodically replaces the distorted particle set with a new one aligned with an underlying grid [46, 16, 47], an operation hereafter referred to as remeshing or redistribution.

The particle-mesh interpolations and the particle redistribution rely on high-order interpolation kernels. This interpolation can be seen as a discrete convolution between the particle weights and an interpolation kernel:

$$\boldsymbol{\omega}_q = \sum_p \frac{1}{h^3} \boldsymbol{\alpha}_p W\left(\frac{\boldsymbol{x}_q - \boldsymbol{x}_p}{h}\right) \ , \tag{5}$$

where $\boldsymbol{\omega}_q$ is the vorticity lying at a location $\boldsymbol{x}_q$ on the mesh, h is the grid spacing, and W is the interpolation kernel. These kernels fulfill certain conditions to guarantee the method's accuracy. Among them, the

interpolating property, i.e. :

$$W(i) = \begin{cases} 1 & \text{for } i = 0 \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

ensures that the exact solution is algebraically conserved when the particle positions exactly coincide with the grid points. Preserving moments up to order 2 (i.e., the first 3 moments) is also important, as it guarantees the conservation of the integral of the vorticity, the linear impulse, and the angular impulse. This can be expressed as:

$$\sum_i \boldsymbol{x}_i^\alpha W\left(\frac{\boldsymbol{x}_p - \boldsymbol{x}_i}{h}\right) = \boldsymbol{x}_p^\alpha \quad \alpha = 0, \ldots, r-1 \,, \tag{7}$$

with r being the number of conserved moments. This is equivalent to a polynomial reproduction property. The $3D$ interpolation scheme consists in a composition of $1D$ schemes through a tensor product.

The present framework implements the M_4' and M_6^* kernels which are both interpolating. On the one hand, M_4' has a four-point support, an error of $O(h^3)$, and conserves the first three moments [48, 46, 3]. On the other hand, M_6^* has a six-point support, an error of $O(h^5)$, and conserves the first 5 moments [49, 11].

It has been shown that, as such, the Lagrangian discretization does not preserve the solenoidal property of the vorticity field [50]. Therefore, the vorticity field is periodically reprojected onto a solenoidal space thanks to a Helmholtz decomposition ($\boldsymbol{\omega}_{\text{new}} = \nabla\phi + \boldsymbol{\omega}$) [8, 47]. In practice, the scalar potential ϕ is computed by solving a Poisson equation, $\nabla^2\phi = -\nabla\cdot\boldsymbol{\omega}$, on the mesh, such that the solenoidal character of the resulting vorticity field $\boldsymbol{\omega}_{\text{new}}$ is enforced.

Finally, the Lagrangian treatment of the advection waives the usual CFL condition that restricts the time step size. In particle methods, the size of the time step is instead constrained by the flow strain to avoid crossing particle trajectories. This deformation criterion is translated using the 1-norm of the strain tensor, $\boldsymbol{S} = \frac{1}{2}[\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T]$:

$$\Delta t = \text{LCFL}_S \max_{1\le j\le 3}\left(\sum_{i=1}^{3} \|S_{ij}\|\right)^{-1} \,, \tag{8}$$

where LCFL_S is a user-controlled constant whose values are typically $0.15 \lesssim \text{LCFL}_S \lesssim 0.3$. The viscous operator also constrains the time step: this is embodied in the Fourier number, i.e., $r = \frac{\nu\Delta t}{h^2} < r_{\max}$, where $r_{\max}$ is given by the chosen time and spatial discretization scheme.

3. A 3D adaptive Multiresolution Vortex Particle-Mesh method

The multiresolution algorithm follows the same structure as the uniform one, with the addition of a dynamic mesh adaptation step. The introduced multiresolution mesh also imposes to handle interpolation between particles and the mesh of different resolutions.

The whole algorithm can be summarized as follows. The particles are first advected, and the information they carry is interpolated on the grid. On the grid, the velocity is recovered from the vorticity through the Poisson equation and the diffusion as well as the stretching of the vorticity are evaluated using finite difference schemes. Finally, the particle quantities are updated via a mesh-to-particle interpolation. The mesh adaptation is governed by the scale decomposition of the vorticity field and the latter is obtained through the wavelet multiresolution analysis. It is performed at a certain frequency which has to be chosen carefully in order to balance the need for resolution adjustments with the computational cost of modifying the grid. This frequency is constrained to be a multiple of the remeshing frequency, to ensure the alignment of particles with the mesh when adapting it. The whole algorithm is outlined in pseudo-code in Algorithms 1.

In this section, we first provide more details on the mesh adaptation and then introduce a novel representation of the particles that enables their refinement and coarsening. The latter are essential for interpolating particle quantities on the mesh and vice versa. We describe how we proceed to these interpolations, with

a particular emphasis on handling the interface between blocks of different resolutions. We conclude the section with a description of the Poisson solver used in this work.

Algorithm A: The vortex particle-mesh method

Initialization of the simulation
δt; `// time-step - user-defined`
$\omega_{p,0}$; `// Initial particles vorticity - user-defined`
time = 0; iter = 0;
while time $\leq t_{\max}$ **do**
  `RemapParticlesMPI`$(\boldsymbol{x}_{p,i}, \boldsymbol{\omega}_{p,i})$
  $\frac{d\omega_{p,i}}{dt}$, $\boldsymbol{u}_{p,i}$ = `ComputeRHS`$(\boldsymbol{x}_{p,i}, \boldsymbol{\omega}_{p,i})$
  $\boldsymbol{\omega}_{p,i+1}$, $x_{p,i+1}$ = `AdvanceInTime`$(\boldsymbol{x}_{p,i}, \boldsymbol{\omega}_{p,i}, \frac{d\omega_{p,i}}{dt}, \boldsymbol{u}_{p,i})$
  time = time + δt
  iter = iter + 1 '
  if `Modulo`$(iter, N_{remesh}) \equiv 0$ **then**
    $\boldsymbol{\omega}_m$ = `InterpolateFromParticlesToMesh`$(\boldsymbol{x}_{p,i+1}, \boldsymbol{\omega}_{p,i+1})$
    `DeleteParticles()`
    if `Modulo`$(iter, N_{reproj}) \equiv 0$ **then**
      `ReprojectVorticity`$(\boldsymbol{\omega}_m)$
    end
    if `Modulo`$(iter, N_{adapt}) \equiv 0$ **then**
      `AdaptMesh`$(\boldsymbol{\omega}_m)$
    end
    $\boldsymbol{x}_{p,i+1}$ = `CreateParticlesAlignedWithMesh()`
    $\boldsymbol{\omega}_{p,i+1}$ = `CopyMeshToParticles`$(\boldsymbol{\omega}_m, \boldsymbol{x}_{p,i+1})$
  end
end

Algorithm B: ComputeRHS

Data: $\boldsymbol{x}_p, \boldsymbol{\omega}_p$ `// Position and vorticity of the particles`
Result: $\frac{d\omega_p}{dt}$, $\boldsymbol{u}_p$
begin
  $\boldsymbol{\omega}_m$ = `InterpolateFromParticlesToMesh`$(\boldsymbol{x}_p, \boldsymbol{\omega}_p)$
  $\boldsymbol{u}_m, \boldsymbol{\Psi}_m$ = `ComputeVelocityFromVorticity`$(\boldsymbol{\omega}_m)$
  $\frac{d\omega_m}{dt}$ = `ComputeDiffusion`$(\boldsymbol{\omega}_m)$ + `ComputeStretching`$(\boldsymbol{\omega}_m, \boldsymbol{u}_m)$
  $\frac{d\omega_p}{dt}$, $\boldsymbol{u}_p$ = `InterpolateFromMeshToParticles`$(\boldsymbol{x}_p, \frac{d\omega_m}{dt}, \boldsymbol{u}_m)$
end

Algorithms 1: Overview of the vortex particle-mesh method: **Algorithm A** details the typical temporal loop of VPM method, while **Algorithm B** summaries the computation of the right-hand side of Equation (4). Operations in red represent operations needed only in a uniform framework, those in orange represent the operations only present in AMR framework, and the ones in blue represent the operations that are different in AMR frameworks compared to uniform implementations.

3.1. Mesh Adaptation

`murphy` [1] is a scalable multiresolution framework for scientific computing on 3D block-structured collocated grids and constitutes the foundation of our method. It decomposes the grid following an octree-based

approach and relies on a distributed implementation of the wavelet-based multiresolution theory to break down grid information into meaningful and negligible contributions.

In this section, we provide a concise overview of the wavelet-based multiresolution analysis, with particular emphasis on the wavelets implemented within `murphy`. We then describe the grid decomposition procedure and conclude the adaptation strategy.

3.1.1. The wavelet-based multiresolution analysis within `murphy`

This section provides a brief introduction to the multiresolution formalism implemented in `murphy`; we refer to *Gillis et al.* [1] for a more thorough description.

The scale decomposition of fields within `murphy` is performed using lifted interpolating wavelets [51]. To briefly present their formalism, we first go through the conventional orthogonal multiresolution analysis, followed by the biorthogonal multiresolution analysis in second. From there, we outline their main properties, which are obtained thanks to the lifting scheme.

Orthogonal multiresolution analysis. The wavelet-based multiresolution analysis [34, 52, 53] decomposes the space $L_2(\mathbb{R})$ into a sequence of nested, orthogonal subspaces indexed by $L \in \mathbb{Z}$:

$$\cdots \subset V^{L-1} \subset V^L \subset V^{L+1} \subset \cdots .$$

Due to the orthogonality, the intersection between spaces, $\cap_{L\in\mathbb{Z}} V^L$, is empty. The difference between sucessive spaces, V^L and V^{L+1}, defines the detail space W^L which is the orthogonal complement of V^L in V^{L+1}:

$$W^L \perp V^L, \text{ so that } V^{L+1} = V^L \oplus W^L.$$

At each level L, orthonormal bases for both V^L and W^L are constructed by dilating and translating a unique scaling function $\varphi(x)$ and wavelet function $\psi(x)$, respectively. The basis functions are defined as

$$\varphi_k^L(x) = \sqrt{2^L}\varphi(2^L x - k), \quad \psi_k^L(x) = \sqrt{2^L}\psi(2^L x - k),$$

where $\{\varphi_k^L\}$ and $\{\psi_k^L\}$ form orthonomal bases of V^L and W^L, respectively. Orthonormality means $\left\langle \varphi_k^L(x)\ ,\ \varphi_i^L(x) \right\rangle = \delta_{i,k}\ \forall\, \{k,i\} \in \mathbb{Z}$, with the inner product defined by $\langle f(x)\ ,\ g(x)\rangle = \int_{-\infty}^{\infty} f(x)g(x)\ \mathrm{d}x$.

Any function $f(x) \in L_2(\mathbb{R})$ can be projected onto these bases, yielding the scaling coefficients λ_k^L and detail coefficients γ_k^L:

$$\lambda_k^L \triangleq \left\langle f(x)\ ,\ \varphi_k^L(x) \right\rangle \qquad \text{and} \qquad \gamma_m^L \triangleq \left\langle f(x)\ ,\ \psi_m^L(x) \right\rangle\ . \tag{9}$$

With $V^L = V^{L-1} \oplus W^{L-1}$, the projection of $f(x)$ onto level L can be written recursively in terms of the previous level via the refinement relation:

$$P^L[f](x) \triangleq \sum_j \lambda_j^L\ \varphi_j^L(x) = \sum_k \lambda_k^{L-1}\ \varphi_k^{L-1}(x)\ +\ \sum_m \gamma_m^{L-1}\ \psi_m^{L-1}(x)\ . \tag{10}$$

Biorthogonal wavelets. The multiresolution analysis of `murphy` relies on a broader class of wavelet functions, known as interpolating wavelets, which relax the orthonormality requirement by employing biorthogonality [54, 52]. In such a case, the space V^L is paired with a dual space $\tilde{V}^L$, spanned by its own basis functions $\tilde{\varphi}_k^L$ such that

$$\left\langle \varphi_i^L(x)\ ,\ \tilde{\varphi}_k^L(x) \right\rangle = \delta_{i,k} \quad \text{with} \quad \{i,k\} \in \mathbb{Z}.$$

Each space V^L and $\tilde{V}^L$ has a corresponding wavelet complements W^L and $\tilde{W}^L$, respectively. The primal and dual scaling and wavelet functions –φ, $\tilde{\varphi}$,ψ, $\tilde{\psi}$ – form bases of their respective subspaces and satisfy

$$\left\langle \tilde{\varphi}_i^L(x)\ ,\ \psi_k^L(x) \right\rangle = \left\langle \tilde{\psi}_i^L(x)\ ,\ \varphi_k^L(x) \right\rangle = 0$$

$$\left\langle \tilde{\varphi}_i^L(x)\ ,\ \varphi_k^L(x)\right\rangle = \left\langle \tilde{\psi}_i^L(x)\ ,\ \psi_k^L(x)\right\rangle = \delta_{i,k}$$

The detail and scaling coefficients in the biorthogonal formalism are computed using the dual functions:

$$\lambda_k^L \triangleq \left\langle f(x)\ ,\ \tilde{\varphi}_k^L(x)\right\rangle \quad \text{and} \quad \gamma_m^L \triangleq \left\langle f(x)\ ,\ \tilde{\psi}_m^L(x)\right\rangle$$

The refinement relation remains structurally unchanged:

$$P^L[f](x) = \sum_j \lambda_j^L\ \varphi_j^L(x) = \sum_k \lambda_k^{L-1}\ \varphi_k^{L-1}(x)\ +\ \sum_m \gamma_m^{L-1}\ \psi_m^{L-1}(x).$$

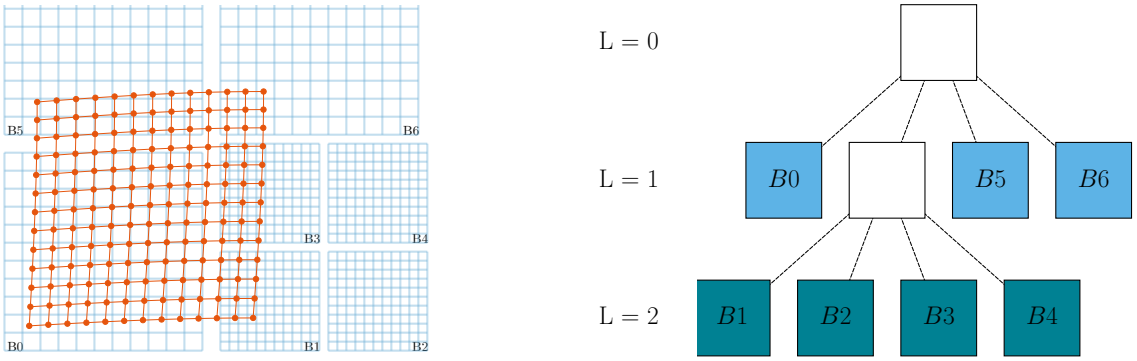
Lifted interpolating wavelets. In `murphy`, interpolating wavelets have been extended to conserve moments through the use of the lifting scheme [55]. First introduced in [51], interpolating wavelets build a polynomial interpolant from coarse-level values. The detail coefficients represent the deviation of the fine-level values from this polynomial, and hence drive the adaptation of the grid. As such, `murphy` provides a collection of linear filters to decompose any function into scaling coefficients, representing retained information, and detail coefficients, representing discardable information. Those wavelets are classified by their degree of interpolation N_w, and by the number $\tilde{N}_w$ of moments they conserve across levels. By construction, an interpolating wavelet of degree N_w guarantees to reproduce polynomials of order $N_w - 1$ exactly. The wavelets are indexed $N_w.\tilde{N}_w$, depending on their properties. Non-lifted wavelets, that do not conserve any moment, will hence be indexed as $N_w.0$. For further details about the wavelets and their implementation within `murphy`, we refer the reader to [1].

3.1.2. Grid decomposition

The grid is decomposed into structured cubic blocks that are organized in an octree-based fashion. Each block is a uniform Cartesian grid with a fixed number of grid points N_b^3. The domain is actually represented as a forest, i.e. an ensemble of trees, where a single tree occupies a cubic subdomain which also corresponds to the octree's root level, i.e. level $L = 0$. The blocks are the leaves of the trees and constitute the smallest computational units. The resulting grid spacing on a block depends on its level L in the tree and can be computed as $h = \frac{1}{2^L N_b}$. Each level is thus twice finer than the previous one. The fields are discretized using a node-centered data layout. A simple example of a $2D$ representation of the domain both as a quadtree and as a physical grid is given in Figure 1. All the octree meta-data are handled by the `p4est` library [28] while the block adaptation and the operations between blocks are implemented within the `murphy` solver. The size of the grid in a block N_b can be chosen arbitrarily at compilation time.

To adapt the grid, the detail coefficients are first computed on all the blocks using the wavelet filters. Specifically, the detail coefficients γ^{L-1} of each block b at level L are computed through the wavelet transform, and the maximum value $\|\gamma^{L-1}\|_\infty^b$ of each block is retained. Based on this value, the blocks are either coarsened or refined. Due to the octree nature of the grid and the fixed number of points per block, the coarsening of blocks happens if and only if all the blocks within the same tree node satisfy $\|\gamma^{L-1}\|_\infty^b \leq \epsilon_c$, with ϵ_c being the compression threshold. With this approach, increasing the number of points per block, N_b, lessens the granularity of the grid and consequently reduces the compression rates of a given signal. On the other hand, the refinement of a block happens if its maximum detail coefficient exceeds the refinement criterion, $\|\gamma^{L-1}\|_\infty^b \geq \epsilon_r$, with $\epsilon_r > \epsilon_c$. This policy allows the user to bound the maximum detail coefficients on the grid, and guarantee that the refinement threshold is nowhere exceeded. The block at level $L = 0$ being the root of the tree, it can not be coarsened. To refine it, we apply the same procedure as the other leaf, meaning that the computed details coefficient belongs to a virtual coarser level $L = -1$.

In such a strategy, the refinement threshold, ϵ_r, controls the overall accuracy of the simulation by limiting the maximum value of the detail coefficient admissible on the grid. In contrast, the coarsening limit, ϵ_c, determines the grid compression rate by fixing the minimum value of the detail coefficients that are preserved. To ensure the stability in the grid adaptation, the ratio between these two thresholds must be higher than $\epsilon_r/\epsilon_c > 2^{N_w}$. This is required by the wavelet framework, where the detail coefficients scale as $\|\gamma\|_\infty \propto h^{N_w}$, with



(a) Physical representation of a two dimensional grid. The mesh is depicted via the blue lines (⌗) while the (—•—) represents the ijk-particles owned by the block $B0$. Blocks $B5$ and $B6$ are only partially displayed.

(b) Quadtree representation of the grid. The colored blocks correspond to leaves that effectively own the data. Blocks of different colors have different local resolutions.

Figure 1: 2D example of a multiresolution grid. The physical grid is represented on the left, while the quadtree representation of the same grid is provided on the right.

N_w the wavelet interpolation order. Since coarsening a block increases its detail coefficients by a factor of 2^{N_w}, a ratio below this threshold would cause the coefficients of a coarsened block to exceed the refinement limit, triggering immediate re-refinement. Such behavior leads to oscillations, or "flip-flops", in grid adaptation. Conversely, setting ϵ_r/ϵ_c significantly higher than 2^{N_w} can lead to a non-unique grid, where the final state depends on the initial grid topology. In such cases, both a block and its coarsened version may satisfy the condition $\epsilon_c < \|\gamma\|^b_\infty < \epsilon_r$. As a result, starting the simulation from the coarsest grid or the finest grid, even with identical initial conditions, can lead to different grid configurations.

Gillis et al. [1] provide more details on the refinement and coarsening and shows that their implementation achieves the expected order of convergence for wavelets from 2.0 to 6.2.

One should note that `murphy` enforces a 2:1 resolution ratio constraint on adjacent blocks during the adaptation. Therefore, the maximum level difference between two neighboring blocks will be 1.

3.2. Multiresolution particles

The multiresolution grid introduced above presents a key challenge for hybrid particle-mesh methods: handling particles and their interactions with a mesh of varying resolution. Particles are discrete points characterized by a position and a volume corresponding to the size of the underlying grid cells. They carry physical quantities and are initially aligned with the grid. However, as they move across the domain, they may encounter regions with grid resolutions that differ from their original size, either finer or coarser. To enable the use of classical interpolation schemes, we dynamically adapt the Lagrangian elements to match the resolution of the mesh and vice versa. We therefore introduce a novel particle representation, which relies on a grid-like connectivity that enables their refinement and coarsening.

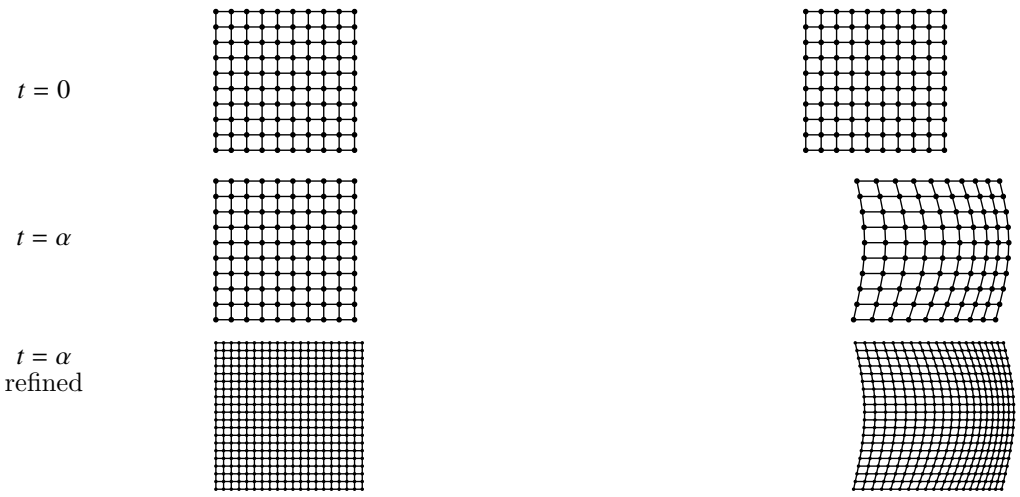
The particles are initially distributed across `murphy` blocks, with each block owning its subset of particles at its local resolution. In the standard VPM method, particles do not interact directly and they are, hence, typically stored as a simple list of elements: x_p, with p being the index of the particle. However, in our approach, each particle is indexed by three coordinates (i, j, k), corresponding to the indices of its initial underlying grid point: $x_{i,j,k}$. This indexing generates a connectivity between the particles, allowing their neighbors to be identified at any time during the simulation. This connectivity also implicitly stores the

reference, i.e., undeformed, configuration of the particles, and enables us to represent any particle fields in both Lagrangian (material), $\mathbf{X} = (X, Y, Z, t)$, and Eulerian (spatial), $\mathbf{x} = (x, y, z, t)$, frames of reference. The mapping between these two configurations is given by $\mathbf{x}(\mathbf{X}, t)$. The material coordinates are computed using the particle's $3D$ indexing (i, j, k), the origin of the block, and its local grid spacing, while the spatial positions are stored and updated in time using the simulation's time integrator.

Using the wavelet tools provided by `murphy`, these spatial coordinates, $\mathbf{x}(\mathbf{X}, t)$, can be refined or coarsened in the reference space $\mathbf{X}(t)$. Since particle positions remain regularly spaced in the material frame, wavelet filters can be directly applied. These wavelets are also essential for retrieving coarse or fine representations of Lagrangian fields, ensuring that both the particle positions and their associated quantities are adapted consistently with the multiresolution grid. This simultaneous adaptation of the particles' positions and fields enables the dynamic adaptation of any Lagrangian field. A visualization of the particles in both frames of reference is provided in Figure 2.

Two conditions are necessary to preserve the validity of this procedure. First, the particle set must remain homogeneous to avoid large deformation of the mapping, which is ensured thanks to the redistribution of the particles. The mapping is hence reset at each remeshing step. Then, the adaption of the particle fields requires ghost particles. Here again, we leverage the algorithm of `murphy` to recompute the ghost particles, as the ijk connectivity enables particle fields to be handled in the same manner as any Cartesian field. Such a ghost particles reconstruction is not a local operation and rely on MPI communications.

The interpolating wavelets, modified by the lifting step, conserve some moments but lose their interpolating trait. When adapting the positions of the particles, we need this interpolating property. Hence, we adopt the policy of always using non-lifted interpolating wavelets when refining or coarsening the particles' positions. The effect of using lifted or non-lifted wavelets for the fields carried by the particles is discussed in Section 5.



(a) $\mathbf{X}(X, Y, Z, t)$ - Evolution of the Material coordinates of the particles in the Lagrangian frame of reference.

(b) $\mathbf{x}(x, y, z, t)$ - Evolution of the Spatial coordinates of the particles in the Eulerian frame of reference.

Figure 2: 2D example of the multiresolution particles. The initial time and an arbitrary time $t = \alpha > 0$ are represented, as well as the refined particle representation at the same $t = \alpha$.

As such, interpreting node-centered particles as carrying integral quantities leads to discrepancies at resolution jumps, i.e. block interfaces. Indeed, relying only on "valid" particles leads to a volume miss,

as illustrated in Figure 3. Since most vortex methods rely on the assumption $\boldsymbol{\omega}_p V_p = \boldsymbol{\alpha}_p$, we choose to conceptually move away from the interpretation of the particles as carrier of integral quantities $\boldsymbol{\alpha}_p$ and instead consider them as pointwise information $\boldsymbol{\omega}_p$ at the particle position, as already done in [56] where they used point particles to handle the simulation of sharp interfaces. This interpretation relaxes the emphasis on exact conservation of moments and focuses instead on the local accuracy of the interpolated field. This solves the conceptual issue of the definition of the volume of the particles at the block interface, and, even though the conservation of moments in not guaranteed, the targeted convergence of the method in space implies the convergence of the computed moments for a vanishing grid spacing.

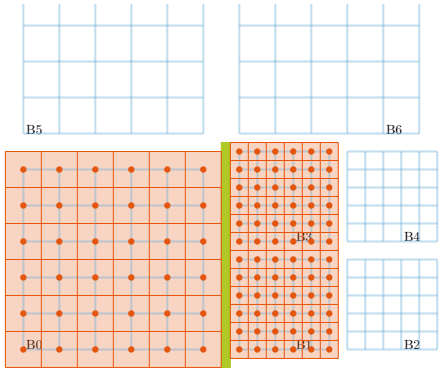


Figure 3: 2*D* Example of the volume miss that appears when using particles that carry integral strength. Particles (●) are represented with their volume in the same shaded color, while the volume that is not covered by particles is represented in (■)

3.3. Particles-Mesh interpolation

Particles are advected without any constraint: they can cross block interfaces and end up on blocks of different resolutions. The interpolation between Lagrangian and Eulerian fields must consequently handle resolution jumps. Here, we consider a strategy where the interpolated field adapts itself to its target field resolution. For example, when interpolating a Lagrangian field on an Eulerian block, we adapt the Lagrangian field to match the Eulerian resolution. Similarly, the Eulerian field is adapted when interpolating an Eulerian source on a Lagrangian target. This procedure is illustrated on Figure 4. The classical interpolation or remeshing schemes presented in Section 2 are then applied to the adapted fields with matching resolution.

With this policy, the quantities interpolated close to a resolution jump receive contributions from two different levels, and we can estimate the expected error. The interpolation of a point information q from p particles to a mesh point m is computed as

$$q_m = \sum_p q_p W\left(\frac{x_m - x_p}{h_m}\right) + \sum_{p\prime} q_{p\prime} W\left(\frac{x_m - x_{p\prime}}{h_m}\right), \tag{11}$$

where $[\cdot]\prime$ denotes quantities that have been adapted before the interpolation, h_m is the mesh spacing, and W is the kernel function. $q_{p\prime}$, the position of the newly created particles, can be decomposed as $q_{p\prime} = q_p + O(h_m^{N_w})$, where N_w is the interpolation degree of the wavelet used for the adaptation. Decomposing $x_{p\prime}$ as $x_{p\prime} = x_p + \epsilon_x$ with $\epsilon_x \sim O\left(h_m^{N_w}\right)$, we can rewrite the second term of Eq. eq. (11) as

$$\sum_{p\prime} q_{p\prime} W\left(\frac{x_m - x_{p\prime}}{h_m}\right) = \sum_{p\prime} \left[q_p + O(h_m^{N_w})\right] W\left(\frac{x_m - x_p - \epsilon_x}{h_m}\right).$$

Further expanding the last factor leads to

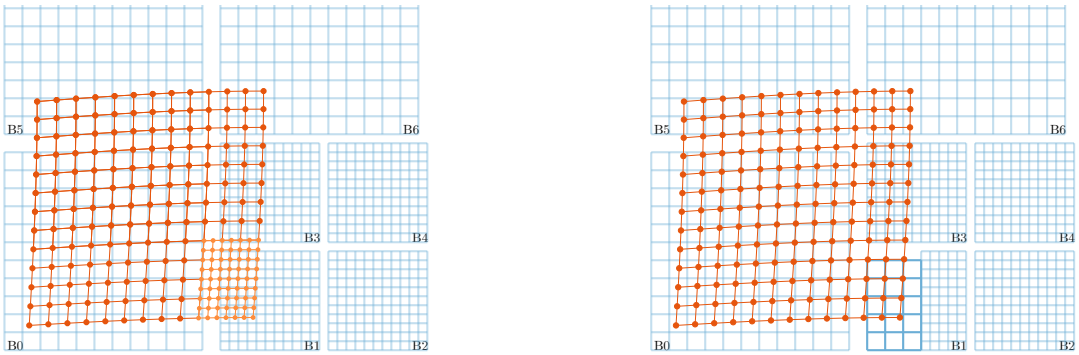
$$W\left(\frac{x_m - x_p - \epsilon_x}{h_m}\right) = W\left(\frac{x_m - x_p}{h_m}\right) + \underbrace{\left(\frac{\epsilon_x}{h_m}\right)}_{O\left(h_m^{N_w-1}\right)} \nabla W\left(\frac{x_m - x_p}{h_m}\right) + O\left(\left(\frac{\epsilon_x}{h_m}\right)^2\right).$$

The expected error on the interpolation has then multiple contributions:

$$q_m = \underbrace{\sum_p q_p W\left(\frac{x_m - x_p}{h_m}\right)}_{C_1 O\left(h_m^{N_k}\right)} + \underbrace{\sum_{p'} q_{p'} W\left(\frac{x_m - x_{p'}}{h_m}\right)}_{C_2 O\left(h_m^{N_k}\right) + C_3 O\left(h_m^{N_w-1}\right) + C_4 O\left(h_m^{N_w}\right)}, \tag{12}$$

where N_k is the order of the interpolating kernels, M_4' and M_6^*, and C_i are constants. Two contributions come from the interpolation kernel , i.e. $C_1 O\left(h_m^{N_k}\right)$ and $C_2 O\left(h_m^{N_k}\right)$, and two others originate from the adaptation of the source field: one from the adaptation of the particle's positions, i.e. $C_3 O\left(h_m^{N_w-1}\right)$, and one from the adaptation of the particle's fields, i.e. $C_4 O\left(h_m^{N_w}\right)$. The predominant one will depend on the considered problem.

To simplify the implementation, the interpolation is restricted to a maximum level difference of one between the source and the target. This constraint introduces a block-CFL-like condition that limits the total displacement of particles between successive remeshings to be smaller than the size of their original block. To ensure that this condition is satisfied for all the blocks within the computational domain, the maximum allowed displacement between the remeshing operations is set to the size of the smallest block within the grid. Combined with the 2:1 adaptation policy, this constraint ensures the level difference between a block and a particle never exceeds one. In practice, this constraint has a limited impact on the simulations. *Gillis et al.* showed that for the transport of smooth functions, the adaptation frequency must be chosen such that the signal travels at most 2 fine-level blocks to ensure the framework accuracy. To have the particles aligned with the grid during the adaptation, we impose the number of iterations between two grid adaptations to be a multiple of the number of timesteps between successive remeshings. The latter condition implies that the maximum particle displacement between remeshings must always be smaller than a single fine-level block size, which is equivalent to the block-CFL-like constraint we introduced.



(a) Example of the interpolation from the particles to the mesh.

(b) Example of the interpolation from the mesh to the particles.

Figure 4: 2D example of particles-mesh interpolation. When performing a multilevel interpolation, the source field is adapted to the resolution of the target one. Here, the interpolations between particles coming from the block $B0$ and the mesh of the block $B1$ are detailed. The mesh is depicted via the blue lines (#) while the (—•—) represents the ijk-particles owned by the block $B0$.

3.4. The unbounded multigrid Poisson solver

To solve the Poisson equation, we use an in-house multigrid Poisson solver, tailor-made for `murphy` application, as described in details in [57]. The solver builds the whole grid hierarchy needed by a multigrid approach and achieves flexibility in terms of boundary conditions, including unbounded conditions, as well as performance by using `flups`, our Fourier-based Poisson solver [58, 43], as a base solver on the coarsest grid. This novel unbounded multigrid Poisson solver handles any combination of unbounded and semi-unbounded boundary conditions and benefits from high-order compact stencils to reduce the amount of communication while improving the solver's accuracy.

4. Implementation

The main implementation challenge of the framework is to preserve and manage efficiently the particle connectivity, i.e., the local ijk ordering needed to adapt the particles via the wavelets tools provided by `murphy`, when the particles move. After remeshing, the connectivity is naturally obtained by storing the ijk-particles as a 3D-array on each block. However, it becomes more complex when they cross block interfaces. In a typical VPM framework, the particles moving outside the boundary of the subdomain of an MPI rank are transferred to the MPI rank owning the neighboring subdomain. However, this transfer of particles is cumbersome and inefficient when the particles are stored as 3D-arrays as it would require a collection of partially filled arrays or a list of particles with additional explicit connectivity information. Therefore, we decided to keep the particles stored on their original block in one 3D-array. Compared to the usual VPM framework, this choice alleviates the need for a remapping of the particles across the MPI ranks but adds MPI communications when it comes to particle-mesh interpolations close to block interfaces. In the following paragraph, we detail how we register the particles that will be communicated and how we perform the interpolations between the particles and the mesh.

4.1. Bounding box of the particles

After the particle displacements, we must identify the particles that influence neighboring blocks during particles-mesh interpolations and thus require MPI communication. To do this, we build a map of different particle statuses: either a particle influences only neighboring blocks, either its support spans both its original block and neighboring blocks, or it only impacts its block of origin. Once the map has been built, each particle has a status assigned. From those statuses, we compute the indices of the particles that will be sent to each neighboring block, creating a bounding box of indices.

4.2. Particle-to-mesh interpolation

When interpolating the particles on the mesh, we must communicate the contribution of the local particles onto the neighboring blocks. Therefore, once the bounding box of indices has been computed, we adapt the particles to the neighbor resolution and interpolate them on the neighboring block mesh. When another MPI process owns the adjacent block, the rank owning the particles performs these operations locally and interpolates the particles' information in a memory buffer that serves as a send buffer. To overlap MPI communications with computations, we first interpolate the particles influencing blocks of other MPI processes into a temporary send buffer. This buffer is then sent. We perform an MPI communication per neighboring blocks. While waiting for the end of the communications, we perform the local interpolation of the particle's field. We then accumulate the received buffer information in our local data as soon as a communication has completed. Such an implementation allows us to keep the waiting time for MPI communication completion to a minimum.

4.3. Mesh-to-particle interpolation

An interpolation from the mesh to the particles is always performed after a particle-to-mesh interpolation. As a result, for this interpolation operation back to the particles, we already know which particles are influenced by each local mesh point, as this information was determined during the prior particle-to-mesh step. This alleviates the need to recompute the bounding box for this operation. In the mesh-to-particle interpolation, the mesh data is adapted to match the resolution of the target particles. To maximize communication-computation overlap, we first send all required mesh point data to neighboring blocks. While waiting for the completion of the communications, we interpolate the local mesh points onto our particles. Then, as soon as neighboring mesh data is received, we adapt it to the resolution of the corresponding particles if needed and we accumulate their contributions to the relevant particle fields.

5. Validation

In this section, we present the numerical validation of our multiresolution particle framework based on three aspects: the convergence of the particle-mesh interpolations themselves, the behavior of the method when solving linear and non-linear advection problems, and the comparison of the method accuracy against a uniform resolution incompressible flow simulation tool. In all cases, the time integration is performed using a low-storage third-order Runge-Kutta method [59].

5.1. Interpolation

The interpolation test measures the convergence of the error when quantities are interpolated from the particles to the mesh and vice versa across a jump of resolution. The interpolation is done using the M'_4 and M^*_6 kernels, with the wavelets 4.0, 4.2, 6.0 and 6.2.

For these test cases, we choose to solve the advection equation of a scalar ϕ in a solenoidal velocity field ($\nabla \cdot \mathbf{u} = 0$):

$$\frac{D\phi}{Dt} = 0. \tag{13}$$

Because the accuracy of interpolation kernels is sensitive to particle distortion, non-linear velocity fields are used to ensure that particle positions undergo non-linear deformation from an initially regular distribution. The number of unknowns per block has been fixed to $N_b^3 = 24^3 = 13824$ unknowns for all cases.

5.1.1. Particle-to-mesh

In the case of the particle-to-mesh interpolation, we deform a compact Gaussian blob centered in $\{x_c, y_c, z_c\}$ and defined as

$$\phi_{init}(\mathbf{x}) = \begin{cases} \exp\left(\frac{-(r(\mathbf{x})/\sigma)^2}{1-(r(\mathbf{x})/(\sigma))^2}\right) & \text{for } r(\mathbf{x})/(\sigma) < 1 \\ 0 & \text{elsewhere} \end{cases} \tag{14}$$

with $r(\mathbf{x}) = \sqrt{(x-x_c)^2 + (y-y_c)^2 + (z-z_c)^2}$. The velocity field to distort the particle set is that of a stagnation flow, where the particle set is stretched in two directions and compressed in the last one. The analytical velocity can be expressed as

$$\mathbf{u}(\mathbf{x}) = \{kx, ky, -2kz\},$$

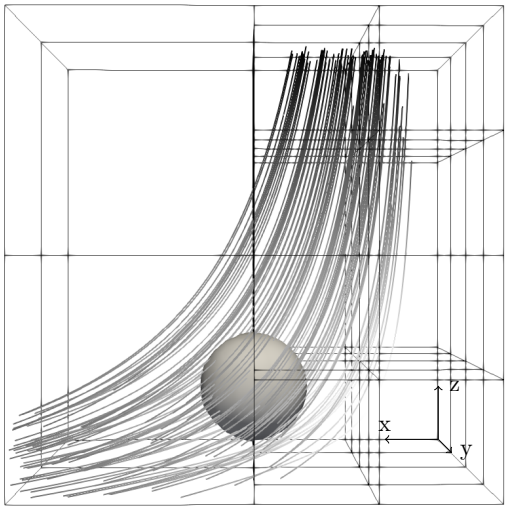
where k is the strain rate of the flow, arbitrarily defined. The positions of the particles are imposed using the analytical expression of their trajectories:

$$\mathbf{x} = \{x_0 \exp(kt), y_0 \exp(kt), z_0 \exp(-2kt)\},$$

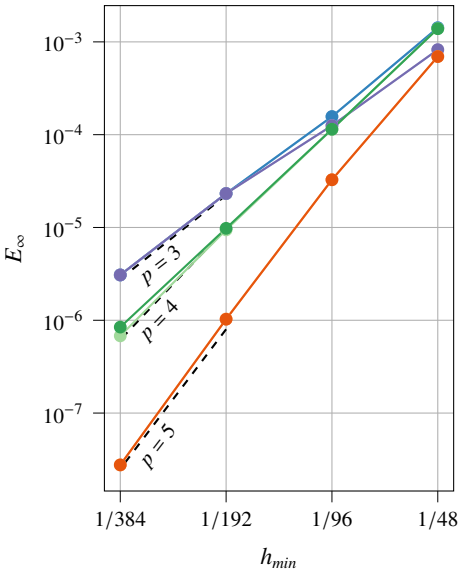
to avoid any error due to the time integration. The Gaussian blob with $\sigma = 0.1L$, ϕ_{init}, in its initial undeformed configuration of the particles is placed in a domain of dimension $\{L, L, L\}$, at $\boldsymbol{x}_c = (0.5L, 0.6L, 0.2L)$, and spans blocks of two different levels, as illustrated in Figure 5a.

The particles are advected up to $t = 0.0015t_0$, with $t_0 = 1/k$. The error of the particle-to-mesh interpolation is computed by comparing the values obtained on the mesh with the initial condition deformed with the inverse mapping of the flow:

$$E_\infty = \max |\phi_{init}(x\exp(-kt), y\exp(-kt), z\exp(2kt)) - \phi_{\text{mesh}}(x, y, z)| \tag{15}$$



(a) Initial condition of the stagnation flow case. The streamlines are colored according to the velocity magnitude: the darker, the higher the velocity. The block boundaries are shown in black. The Gaussian blob is located at the resolution jump.



(b) Convergence of the particle-to-mesh interpolation error of the M'_4 kernel with the wavelets 4.0 ( ), 4.2 ( ), 6.0 ( ) and 6.2 ( ) as well as the M^*_6 kernel with the wavelets 4.0 ( ), 4.2 ( ), 6.0 ( ) and 6.2 ( ) in function of the minimum grid spacing.

Figure 5: Stagnation flow: advection of a compact scalar Gaussian field in a non-linear velocity field. Both the initial condition and the obtained convergence of the interpolation are displayed.

In Figure 5b, we observe that the convergence of the error is bounded by the minimum between the order of the wavelet and the order of the interpolation kernel. The M'_4 kernel hence converges with order 3, no matter if it is combined with a wavelet of order 4 or 6. The prevalent term of the error is the contribution of the interpolation kernel and all the lines are superimposed. The M^*_6 kernel converges with order 4 with the wavelets 4.0 and 4.2, the wavelet approximation being here the predominant source of error. Finally, the M^*_6 kernel with the wavelets 6.0 and 6.2 converges with order 5, the error, this time, being driven by the interpolation kernel.

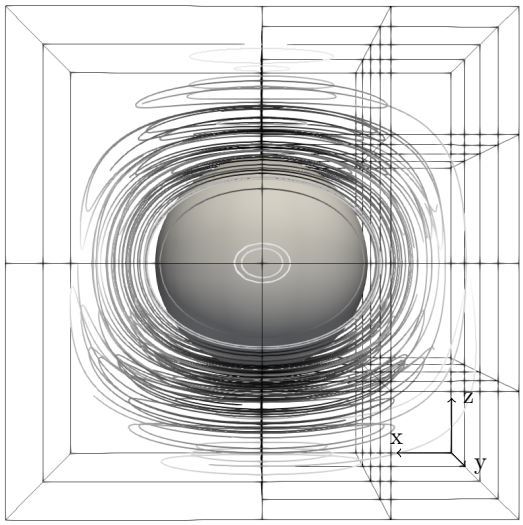
5.1.2. Mesh-to-particle

To study the mesh-to-particle interpolation, we used a 3D periodic and incompressible flow field introduced by *LeVeque* [60] and extensively used in the level-set literature [61]. Being a challenging scale separation problem, it has also been chosen by *Gillis et al.* [1] to validate the ability of their wavelet-based adaptation to track the need for computational resources. The velocity field is defined as:

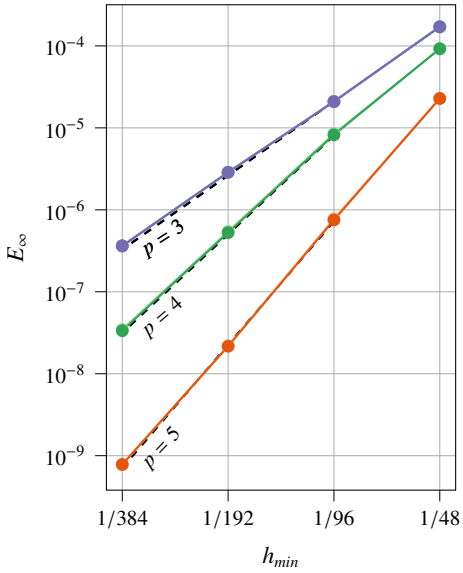
$$\mathbf{u}(\mathbf{x}) = \begin{cases} 2U\sin^2(\pi x)\ \sin(2\pi y)\ \sin(2\pi z) \\ -U\sin(2\pi x)\ \sin^2(\pi y)\ \sin(2\pi z) \\ -U\sin(2\pi x)\ \sin(2\pi y)\ \sin^2(\pi z) \end{cases} \tag{16}$$

and its components are modulated by $\cos(\pi t/3)$ [61, 1]. A Gaussian blob, with $\sigma = 0.2L$, is reused as the analytical solution and centered in the middle of a unit cube domain of size $\{L, L, L\}$. It is shown in Figure 6a with the streamlines of the flow.

In this test case, only the 'forward' advection is considered, up to $t = 0.0125t_0$ with $t_0 = L/U$. The set of particles is distorted, and the undeformed Gaussian blob on the mesh is interpolated onto it. We then compare the resulting particle field to the analytical solution. Looking at the convergence of the error as a function of the grid spacing in Figure 6b, the same conclusion as in the particle-to-mesh case can be drawn. The error is bounded by the minimum between the order of the wavelet and the order of the interpolation kernel.



(a) Illustration of the deforming flow used in the mesh-to-particle convergence study. The streamlines are colored according to the velocity magnitude: the darker, the higher the velocity. The block boundaries are shown in black. The Gaussian blob is located at the resolution jump.



(b) Convergence of the mesh-to-particle interpolation error of the M'_4 kernel with the wavelets 4.0 ( ), 4.2 ( ), 6.0 ( ) and 6.2 ( ) as well as the M^*_6 kernel with the wavelets 4.0 ( ), 4.2 ( ), 6.0 ( ) and 6.2 ( ) in function of the minimum grid spacing.

Figure 6: Advection of a compact scalar Gaussian field in a non-linear velocity field. Both the initial condition and the obtained convergence of the interpolation are displayed.

5.2. Convergence analysis for the advection equation

Now that the cross-level interpolation has been validated, this section focuses on validating our hybrid AMR particle-mesh method for the resolution of the advection equation, Equation (13), for longer time.

As highlighted in *Gillis et al.* [1], the convergence analysis of a AMR method is non-trivial. For a non-adapting grid case, the convergence study consists in showing the maximum error E_∞ as a function of the grid spacing h. However, the local grid spacing varies in time in the AMR case, and there is no direct control over the minimum grid spacing since it is the result of the grid adaptation. Furthermore, it cannot be ensured that the maximum error E_∞ on a multi-level grid is located on a region with the finest resolution. Therefore, following the procedure proposed in [1], we will perform the convergence analysis by relating the error made at the end of a simulation with the refinement threshold ϵ_r. We will show that ϵ_r can successfully control the error and even be used to bound the maximum error made in a simulation. The two cases proposed in the following subsections are similar to those presented in Section 5 of [1].

5.2.1. Linear advection of a Gaussian blob

We first test our adaptive method by transporting a Gaussian blob in a uniform velocity field within a domain of size $\{L \times L \times 2L\}$. The Gaussian blob is defined as:

$$\phi(x, y, z) = \exp\left(\frac{-r^2}{\sigma^2}\right) \qquad \text{where } r^2 = (x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2. \tag{17}$$

The uniform velocity field is defined to produce a linear translation of the blob in all three spatial directions and the final time is chosen so that the blob travels half the domain length along the z-axis. Its initial position is set to ensure symmetry between the initial and final locations relative to the domain center. All numerical parameters for this test case are listed in Table 1.

Parameter	**Value**
Domain size	$\{L \times L \times 2L\}$
Velocity	$\boldsymbol{u} = \{U_x, U_y, U\}$
	$U_x = U(1 - \cos(\pi/3 - \sqrt{2}))/\sin(\sqrt{2})$
	$U_y = U\cot(\sqrt{2})$
Final time	$t_f = L/U$
Initial position	$\{0.5(L - t_f\, U_x), 0.5(L - t_f\, U_y), 0.5L\}$
Blob size	$\sigma = L/15$

Table 1: Numerical parameters for the linear advection test case.

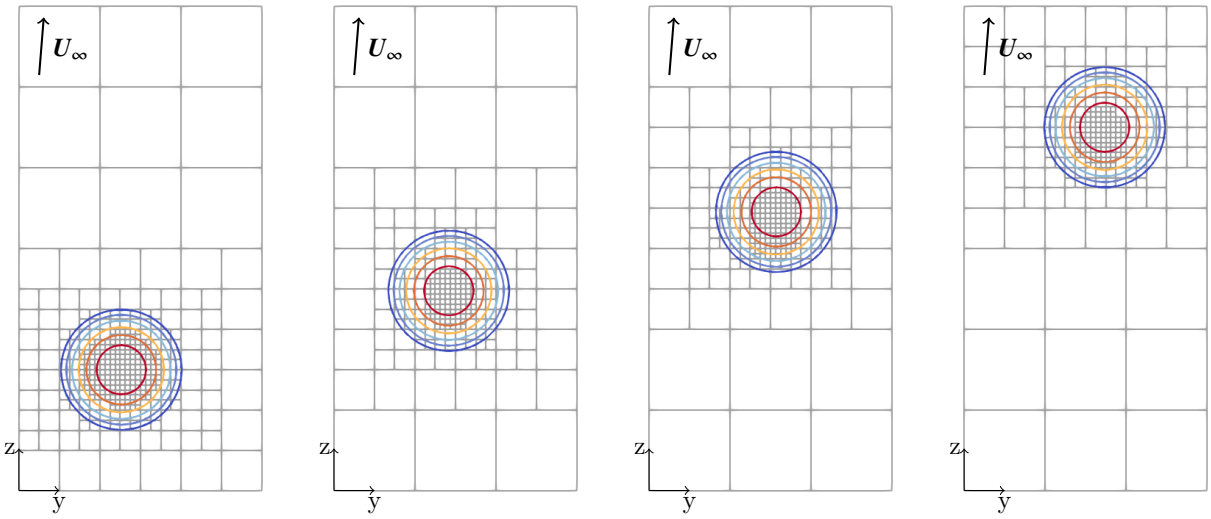
In this case, the timestep is chosen to satisfy $CFL = 0.5$ based on the finest grid resolution. The particles are remeshed every three timesteps, and adaptation is performed every six. Although a CFL constraint is not required for particle methods, it here restricts the distance the information travels between successive adaptations. In our case, the information travels across 1/8th of the finest blocks between two adaptations. Using this configuration, we vary the refinement threshold ϵ_r while maintaining the ratio of the refinement over the coarsening threshold constant and equal to $\epsilon_r/\epsilon_c = 100$. Two wavelets, the 4.0 and 4.2, combined with the M'_4 and M^*_6 kernels are considered.

The Figure 7 shows a 2D slice of the domain at four different times obtained with the simulation using the M^*_6 kernel combined with the wavelet 4.0, for $\epsilon_r = 10^{-6}$ and $\epsilon_c = 10^{-8}$. We can observe a front/back asymmetry in the mesh topology. The moving field triggers the refinement right in front of the blob, while coarsening is initiated further behind the blob. Such an asymmetry can be reduced by decreasing the refinement to coarsening ratio, which results in a more aggressive adaptation.

Figure 8 shows the evolution of the simulation diagnostics over time for wavelets 4.0 and 4.2 combined with the M'_4 and M^*_6 kernels. Figures 8a and 8b show that decreasing the refinement threshold effectively reduces the error amplitude. Comparing the interpolation kernels, the error produced by M^*_6 is two orders of magnitude smaller than that of M'_4.

Figure 8c depicts the time history of the maximum detail coefficient of the grid. It confirms that the refinement threshold bounds the maximum detail coefficient. The global evolution of this maximum detail coefficient presents prominent spikes that occur with adaptation events. Between successive adaptations, the amplitude of the detail coefficients progressively increases. When adapting the grid, blocks containing detail coefficients exceeding the refinement threshold are refined, causing a sudden drop in the detail coefficient maximum value, explaining the observed spikes. The location of the maximum is not necessarily on the finest grid but can vary in space as blocks are refined or coarsened.

Figure 8d shows the evolution of the number of blocks with time. Interestingly, the adaptation strategy depends on the interpolation kernel, with the M^*_6 case resulting in a noteworthy reduction in unknowns compared to M'_4, especially for low refinement thresholds. The reduced impact of the wavelet on the adaptation strategy is explained by the fact that the signal provided to the wavelet decomposition algorithm



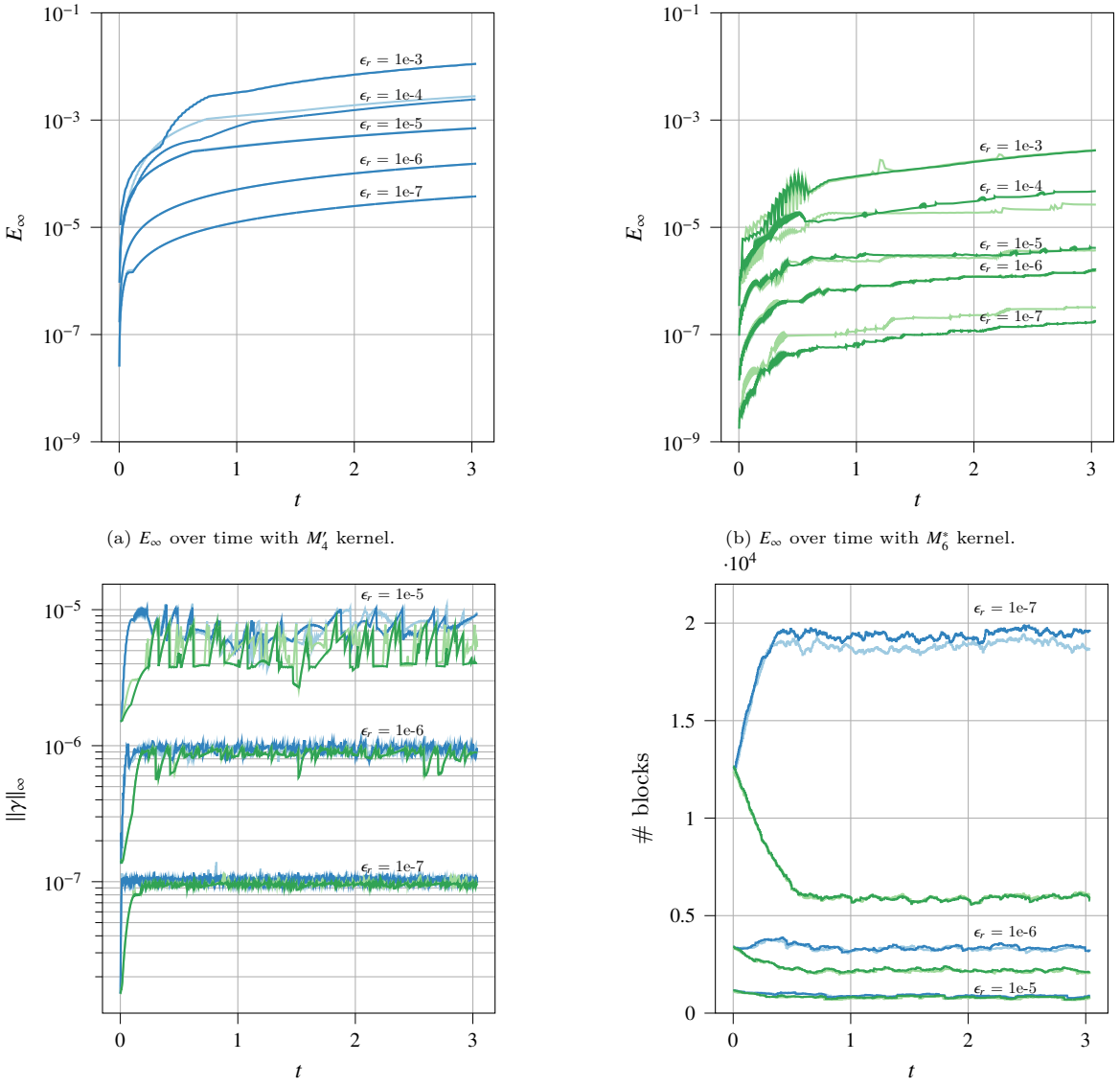
(a) $t = 0$ (b) $t = t_f/3$ (c) $t = 2t_f/3$ (d) $t = t_f$

Figure 7: 2D Slice in the XZ-plane of the Gaussian blob at its center. The simulation has been done with the M_6^* kernel, the wavelet 4.0 and $\epsilon_r = 10^{-6}$ with $\epsilon_r/\epsilon_c = 100$. Isolevels of the blob at $[10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-2}, 10^{-1}]$ are represented.

has been first interpolated from the particles to the mesh. The interpolation acts as a first smoother that interferes with the subsequent wavelet analysis. Therefore, using lifted or non-lifted wavelets does not produce significant differences in the grid topology: their two curves are almost superimposed for a given refinement threshold. This was not the case in [1] where they observed more compressed grids when using lifted wavelets. In both cases, the fine regions of the initial condition are overestimated, due to the non-optimal ϵ_r/ϵ_c ratio. In the case of the M_6^* kernel, the Gaussian blob first moves across these well-refined regions: there is no creation of new blocks in front of the blob. Behind the blob, the blocks are progressively coarsened, which explains the reduction in the number of blocks at the beginning of the simulation. Once the blob reaches the refined region's limits, the blocks' generation is triggered again. From there, the number of blocks remains constant. On the other hand, the M_4' kernel triggers the creation of detail coefficients, a behavior already observed in [49]. Therefore, at the beginning of the simulation, the number of blocks in the grid first increases before reaching a plateau. A lower refinement threshold enhances the wavelet sensitivity to these small numerical artifacts, amplifying the differences in the grid topology between the two kernels.

Figure 9 reveals the convergence of the errors made at the end of the simulations and compares them to uniform resolution simulations. Looking at the maximum error E_∞ in function of the minimum grid spacing h_{min} in Figure 9a, we can observe a degradation in the convergence order compared to the order obtained in Section 5.1. This deterioration is due to the accumulation of errors in the multiple remeshings of the particles.

The multiresolution framework employing M_4' is as accurate as its uniform counterpart and has the same convergence behavior, which is impressive given the number of unknowns is divided by a factor of approximately 55, see Figure 10. It is not the case when using M_6^*. In this second case, two phenomena coexist and influence the convergence: the error is either located in the bulk of blocks or at the resolution jumps. For high refinement threshold values, the prevalent error is generated in the bulk of blocks, and the multiresolution framework matches the uniform results. When reducing the refinement threshold, the grid topology changes and spans up to 4 levels of refinement. Errors occur at resolution jumps, which slightly reduces the method order.



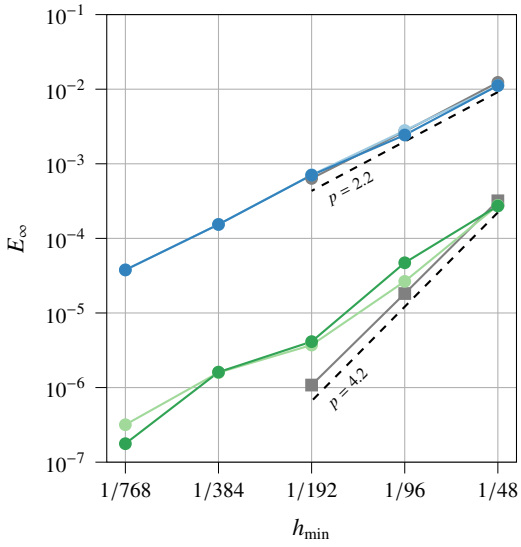
(a) E_∞ over time with M'_4 kernel.

(b) E_∞ over time with M^*_6 kernel.

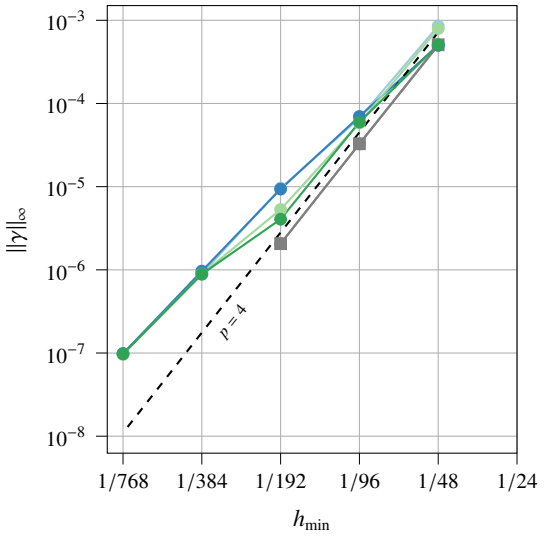
(c) $\|\gamma\|_\infty$ over time.

(d) Number of blocks over time.

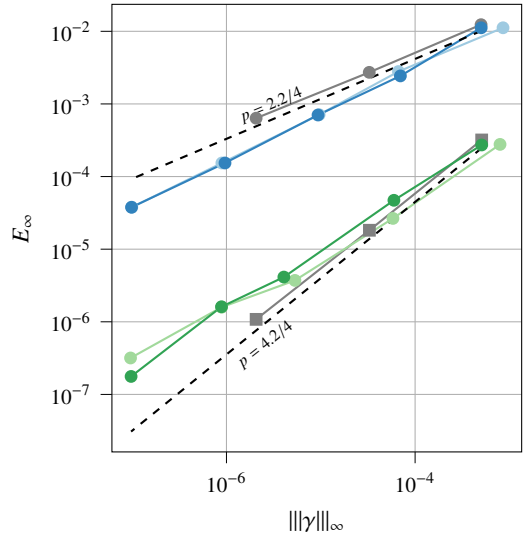
Figure 8: Linear advection of a Gaussian blob – Impact of ϵ_r, with $\epsilon_r/\epsilon_c = 100$, on the time evolution of the maximum error, the maximum detail coefficient and the number of blocks using the M'_4 combined with wavelet 4.0 ( ) and 4.2 ( ) and using M^*_6 combined with wavelets 4.0 ( ) and 4.2 ( ).



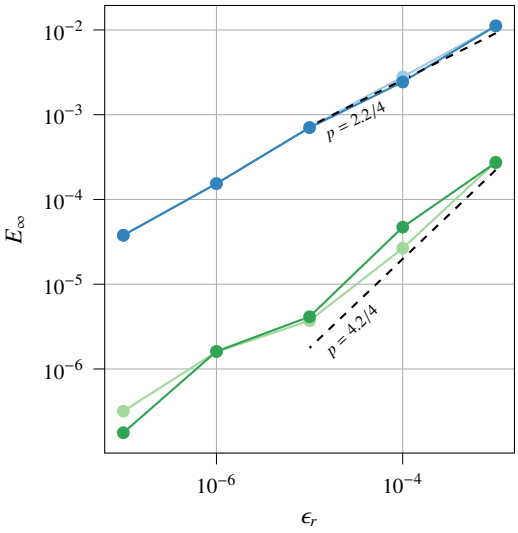
(a) Convergence of E_∞ as a function of the minimum grid spacing $h_{\min}$.



(b) Convergence of $\|\gamma\|_\infty$ as a function of the minimum grid spacing $h_{\min}$.



(c) Convergence of E_∞ as a function of $\|\gamma\|_\infty$.



(d) Convergence of E_∞ as a function of ϵ_r.

Figure 9: Linear advection of a Gaussian blob – Convergence study of the AMR framework combining the M'_4 kernel with the wavelet 4.0 ( ) and the 4.2 ( ), the AMR framework combining the M^*_6 kernel with the wavelet 4.0 ( ) and the 4.2 ( ), and the uniform resolution framework using the M'_4 ( ) and the M^*_6 ( ) kernels.

Plotting the resulting maximum error as a function of ϵ_r in Figure 9d validates that the refinement threshold successfully controls the error. As explained in [1], this convergence rate results from the combination of different components. On the one hand, ϵ_r bounds the maximum detail coefficient, $\|\gamma\|_\infty$, through the adaptation policy. Following the wavelet theory, the maximum detail coefficient associated with the projection of a given smooth function onto a level with spacing h scale as $\|\gamma\|_\infty \propto h^{N_w}$, where here $N_w = 4$ is the polynomial interpolation order of the wavelets 4.0 and 4.2. This is confirmed in Figure 9b by the curve showing the maximum detail coefficient computed during uniform grid simulations. On the other hand, Figure 9a shows us that the convergence order of the errors of the frameworks are respectively $E_\infty = O(h^{2.2})$ for that using M'_4 and $E_\infty = O(h^{4.2})$ for that using M^*_6. Putting it all together, the expected convergence rate is $E_\infty \propto \|\gamma\|_\infty^{N_k/N_w}$, with N_k the resulting order of the interpolation kernel and N_w the order of the wavelet, as depicted in Figure 9c.

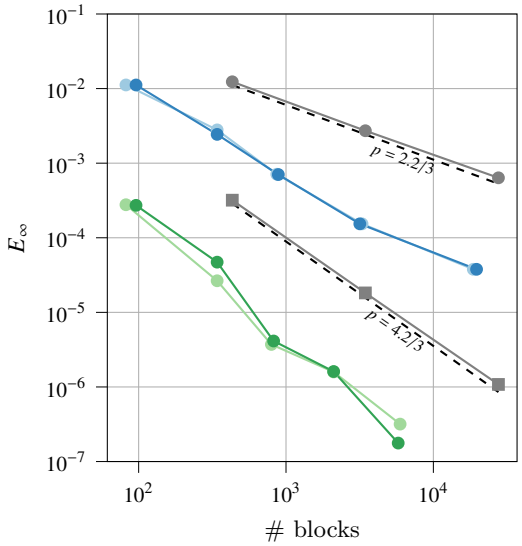


Figure 10: Linear advection of a Gaussian blob – E_∞ as a function of the number of blocks with the AMR framework combining M'_4 kernel with the wavelet 4.0 ( ) and the 4.2 ( ), the AMR framework combining the M^*_6 kernel with the wavelet 4.0 ( ) and the 4.2 ( ), and the uniform resolution framework using the M'_4 ( ) and the M^*_6 ( ) kernels.

Finally, the maximum error made in uniform simulations as a function of the total number of blocks (Figure 10) presents a slope equal to the order of the spatial discretization divided by three. This is explained by the fact that the number of blocks in a uniform grid is proportional to $\frac{1}{h^3}$ while the error is proportional to the grid spacing to the power of the interpolation order. For the AMR simulations, we can see that the slopes are steeper, meaning that the computational gains made over the uniform resolution simulations increase with the expected accuracy. The gains will however be highly dependent on the scale separation of the studied problem. Again, for a given number of blocks, using the M^*_6 kernel significantly reduces the error amplitude compared to the M'_4 kernel.

5.2.2. Non-Linear advection of a Gaussian blob

The ability of the refinement threshold, ϵ_r to control the error on the solution of non-linear problem is studied using the same velocity field as in Section 5.1.2. The solution is advected up to time $t = 0.5$. Since there is no analytical solution, we compare our results with a uniform resolution simulation performed at the finest level as a reference solution. For the multiresolution cases, we used two wavelets, 4.0 and 4.2, two interpolation kernels, M'_4 and M^*_6, and considered four different refinement thresholds $\epsilon_r = \{1; 10^{-1}; 10^{-2}; 10^{-3}\}$. We set the time step as $\Delta t = \frac{1/2h_f}{U}$, with h_f the instantaneous finest grid spacing and U fixed to 1. When a simulation reaches the final time, the grid is refined to a uniform resolution grid of level 5 with the chosen

wavelet. The maximum error between the uniform resolution simulation and the resulting grid is computed. Particles are remeshed every five timesteps and the grid is adapted every ten timesteps.

The results are shown in Figure 11. Figures 11a and 11c demonstrate the convergence of our frameworks and the ability to control the error using ϵ_r which provides an estimate for the error. As already observed, the convergence rate relating those two quantities matches the theory: $E_\infty \propto \epsilon_r^{N_k/N_w}$. Figures 11b and 11d show the total number of blocks at the final time. This metric validates the observations made in the previous test case but for a more challenging problem.

5.3. Comparison of the AMR and uniform resolution Navier-Stokes solvers

Finally, we deploy our multiresolution particles in a complete VPM solver for the Navier-Stokes equations and compare its accuracy to its equivalent in uniform resolution. The comparison is performed on the test case of aircraft trailing vortices, as previously used in [8]. A finite-span wing that generates lift sheds a vortex sheet, which rolls up into a vortex pair. These vortices are subject to instabilities, among which the well-known long-wavelength Crow instability [62], which governs the long-time decay of the pair. More complex systems, with multiple vortex pairs, can be generated by the roll-up and also undergo shorter, rapidly growing instabilities with medium wavelength. In the four-vortex system presented here, the secondary vortex pair is weaker and counter-rotating compared to the leading pair. Those can, for example, be initiated by a negative load on the horizontal tail plane. These secondary vortices are unstable, and this quickly results in large-amplitude deformations and stretching of the secondary vortices into the so-called Ω-loops, which are eventually wrapped around the primary vortices. This finally leads to reconnections of unequal strength vortices and accelerates their decay.

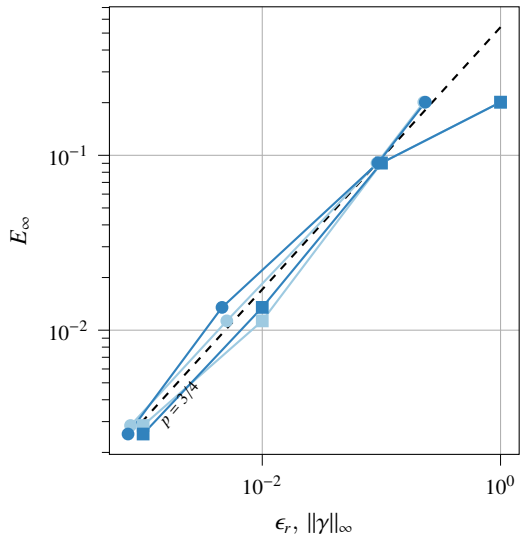
This problem presents a definite sparsity in its vorticity distribution and is particularly well suited for adaptive mesh refinement simulations since the vortices remain relatively compact and move within a large domain.

The problem is solved in a time-developing manner, starting from an initial condition that represents the four vortices. The cores of the initial vortices are prescribed to follow a low-order algebraic profile:

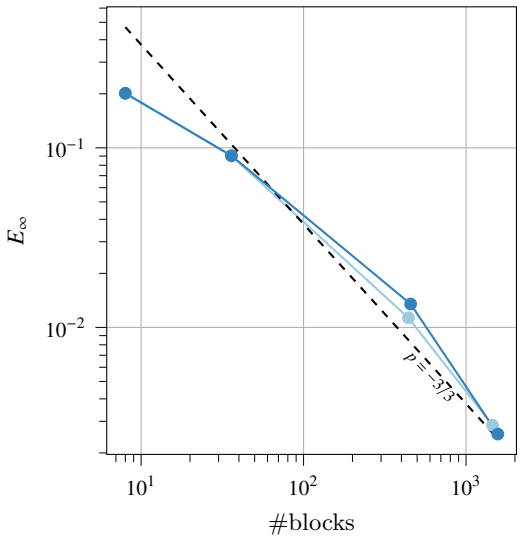
$$\omega(r) = \frac{1}{2\pi\sigma^2}\left(\frac{1}{(1+(r/\sigma)^2}\right)^2 , \tag{18}$$

with σ being the core size. The distance between the vortices is noted b. The core sizes are equal to $\sigma_1/b_1 = 0.075$ and $\sigma_2/b_1 = 0.005$. The subscripts 1 and 2 correspond to the primary and secondary vortices, respectively. The distance between the secondary vortices is $b_2/b_1 = 0.3$ and the ratio of circulations is $\Gamma_2/\Gamma_1 = -0.3$. The equivalent vortex pair can be expressed through the parameters $\Gamma_0 = \Gamma_1 + \Gamma_2$ and $b_0 = \frac{\Gamma_1 b_1 + \Gamma_2 b_2}{\Gamma_0}$. Those are then used for the non-dimensionalization, for instance with $t_0 = \frac{2\pi b_0^2}{\Gamma_0}$. The Reynolds number is equal to $Re = \Gamma_0/\nu = 3500$. The domain is periodic in the $z-$direction and unbounded in the $x-$ and $y-$directions. The length of the domain in the axial direction maximizes the growth rate of the instability: $L_z/b_0 = 0.76$ and the other directions are fixed to $L_x/L_z = 5$ and $L_y/L_z = 7$. To trigger the instability, the center lines of the vortices are perturbed by a sine function of amplitude $10^{-6}\, b_1$ in the x-direction. The primary vortices are in opposition of phase compared to the secondary vortices.

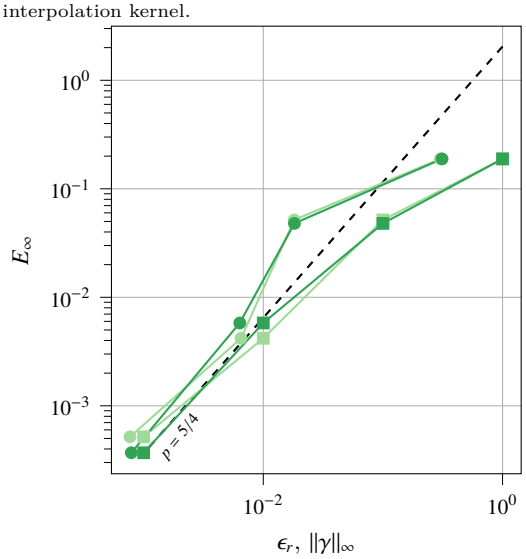
In the uniform resolution case, the grid size is fixed to 1280×1792×256, resulting in 587 million unknowns. In the multiresolution case, we used blocks of size 32^3 and fixed $\epsilon_r = 10^{-2}$ and $\epsilon_r/\epsilon_c = 2^6$, the optimal refinement to coarsening ratio for the wavelet 6.0. In such a configuration, we need to restrict the maximum allowable refinement level of our AMR simulation to 3, ensuring that the uniform and multiresolution have the same finest grid spacing. The maximum timestep is fixed and equal to $\Delta t = 3.3 \times 10^{-4} t_0$. In both cases, four-order finite differences are used for operations on the grid, the interpolations are made using M_6^* and the time integration is performed using a third order low storage Runge-Kutta [59]. In the AMR simulation, the wavelet 4.0 is combined with the M_6^* interpolation kernel, and the wavelet 6.0 is used for the grid adaptation and the ghost reconstruction, to enable the use of fourth-order finite difference schemes on the grid.



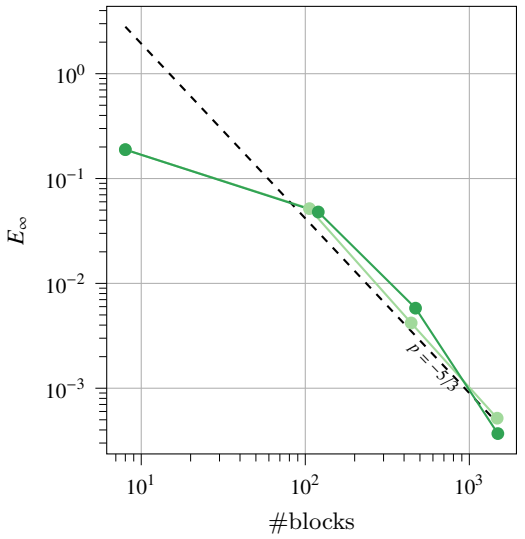
(a) Convergence of E_∞ as a function of ϵ_r ( ) and as a function of $\|\gamma\|_\infty$( ). The wavelets 4.0 and 4.2 are combined with the M'_4 interpolation kernel.



(b) Convergence of E_∞ as a function of the number of blocks. The wavelets 4.0 and 4.2 are combined with the M'_4 interpolation kernel.

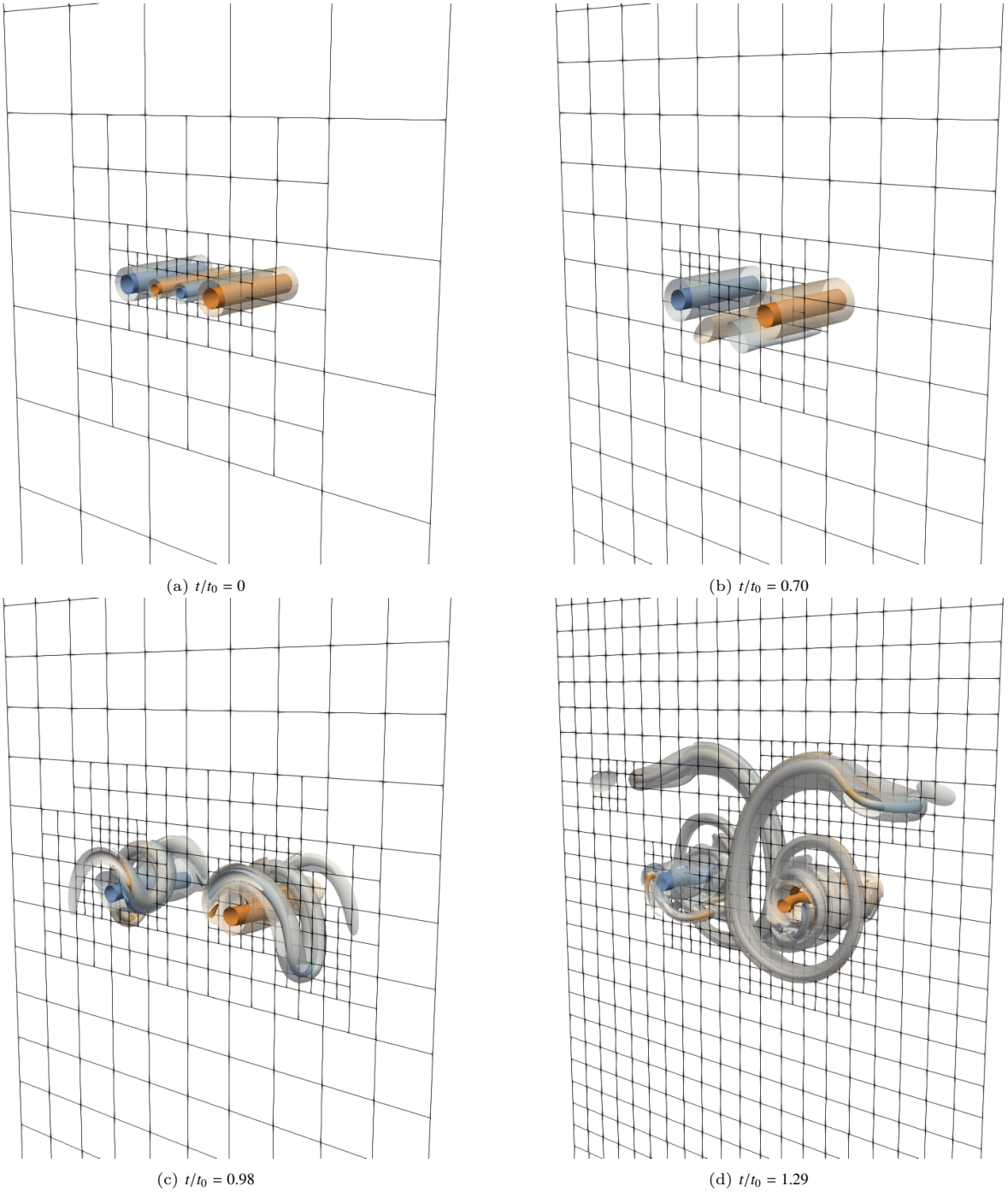


(c) Connvergence of E_∞ as a function of ϵ_r ( ) and as a function of $\|\gamma\|_\infty$ ( ) with $\epsilon_r/\epsilon_c = 100$. The wavelets 4.0 and 4.2 are combined with the M^*_6 interpolation kernel.



(d) Convergence of E_∞ as a function of the number of blocks. The wavelets 4.0 and 4.2 are combined with the M^*_6 interpolation kernel.

Figure 11: Wavelet-based convergence analysis: advection of a compact scalar Gaussian field in a non-linear velocity field. $\epsilon_r/\epsilon_c = 100$ in all cases.



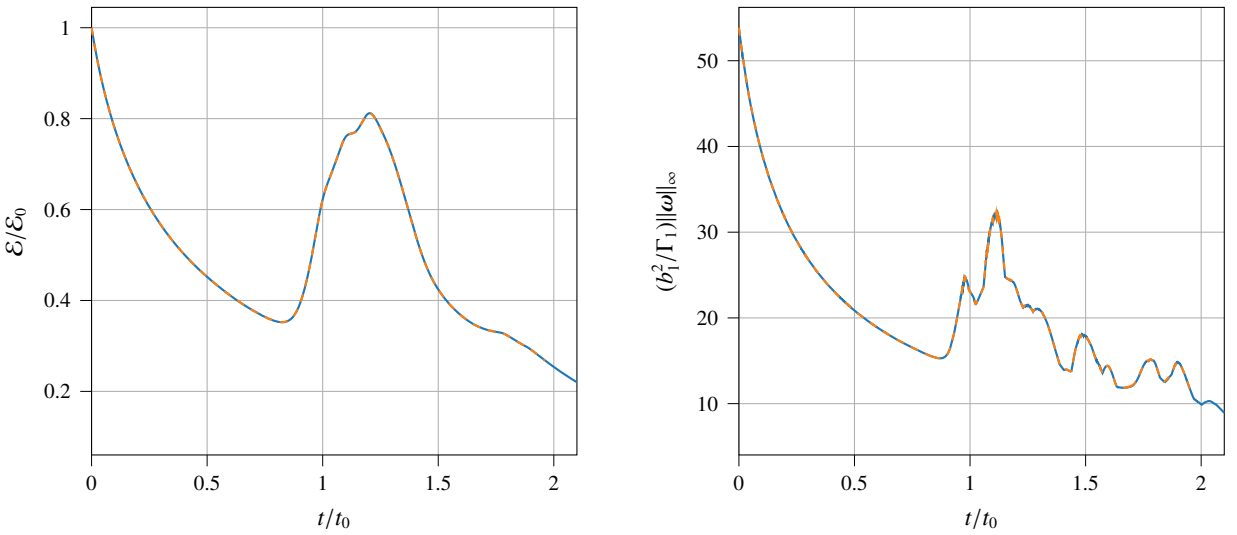
(a) $t/t_0 = 0$ (b) $t/t_0 = 0.70$

(c) $t/t_0 = 0.98$ (d) $t/t_0 = 1.29$

Figure 12: Iso-surfaces evolution of the four-vortex system. The opaque surface represents $|\boldsymbol{\omega}| = 10\frac{\Gamma_1}{b_1^2}$, and the transparent one to $|\boldsymbol{\omega}| = 10\frac{\Gamma_1}{b_1^2}$. The surface colors show the magnitude of ω_z . The evolution of the grid is also shown via a xy-slice located at $L_z/4$.

Figure 12 illustrates the evolution of the vortex system along with the adaptive refinement of the AMR grid. Initially, both vortex pairs are captured within fine-level blocks that then follow the wrapping motion of the secondary vortex around the main ones.

The computational gains through a reduced number of unknowns are affected because of our unbounded Poisson solver. Specifically, the FFT-based coarse-grid solver requires that all blocks along an unbounded boundary reside at the coarsest level of the multiresolution grid. Consequently, when a boundary block is refined, the entire grid must be adapted accordingly, increasing the minimum level of the grid by one level to ensure the convergence of the elliptic solver. At the beginning of the simulation, the grid spans up to four refinement levels, yielding a 34-fold reduction in unknowns compared to a uniform grid. As the simulation progresses, refinements of the coarsest level occur as the secondary tubes are transported close to the domain boundaries. This reduces the effective range of refinement to just two levels, corresponding to a reduction factor of approximately four. Despite this limitation, the solver still demonstrates a strong ability to efficiently adapt its resolution to the flow characteristics. Moreover, the AMR approach does not compromise accuracy, as demonstrated by the evolution of global diagnostics such as the enstrophy, $\mathcal{E} = \int \frac{\omega\cdot\omega}{2} dV$, in Figure 13a. The uniform and multiresolution simulations yield nearly superimposed results, with a maximum relative difference of only 0.07% of the enstrophy peak value. A similar agreement is observed in local diagnostics such as the $L-\infty$ norm of the vorticity field, $\|\boldsymbol{\omega}\|_\infty$, where the maximum difference is only 0.2% of the peak vorticity, see Figure 13b.



(a) Evolution of the enstrophy, $\mathcal{E} = \int \frac{\omega\cdot\omega}{2} dV$, as a function of time

(b) Evolution of the $L-\infty$ norm of the vorticity, $\|\boldsymbol{\omega}\|_\infty$, as a function of time

Figure 13: Comparison of global and local diagnostics obtained from a uniform (—) and from a multiresolution (- -) simulation.

6. Performance of the incompressible Navier-Stokes solver

While the reduction of the number of unknowns in the AMR framework is substantial, this comes at an increased computational cost by unknown as compared to the uniform resolution solver. This section aims to identify the time spent in each operation of our new framework, study the parallel performance of our implementation, and assess the actual computational gains of this new approach compared to its uniform counterpart. All the simulations were performed on the Lumi-C partition, a EuroHPC facility hosted in Finland. Each of its CPU nodes has two AMD EPYC 7763 CPUs with a total of 128 cores per node, which are connected with a 200 Gb/s `slingshot`-11 network.

6.1. Breakdown of the time-to-solution

Figure 14b presents the average time spent in the different steps of the VPM temporal loop. The timings are taken from the first 1000 timesteps of a simulation of the collision of two vortex rings at $Re_\Gamma = 10000$. The simulation ran for 5.6 hours on 1536 MPI ranks, which brings the total cost of this part of the simulation to 8.6 kCPUh.

The two rings have their principal axis aligned with $\hat{\mathbf{e}}_z$ and their geometry is characterized by the following variables:

$$r(x,y) = \sqrt{(x-x_r)^2 + (y-y_r)^2} \qquad\qquad s = \sqrt{(z-z_r)^2 + (r-R)^2},$$

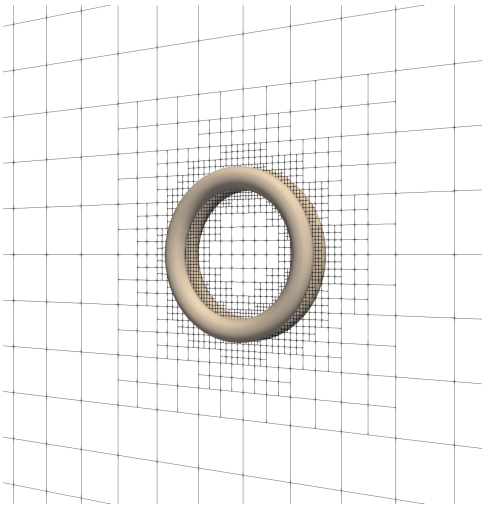
where s is the distance of any arbitrary point in the domain with respect to the ring center-line of radius R. The core of the rings is prescribed to follow a compact profile:

$$\frac{\omega_\theta(s)\pi\sigma^2}{\Gamma} = \begin{cases} C\left(\beta^2\right)\exp\left(-\frac{(s/\sigma)^2}{(1-(s/\sigma_0)^2)}\right) & \text{for } \rho_2 < 1 \\ 0 & \text{elsewhere.} \end{cases}$$

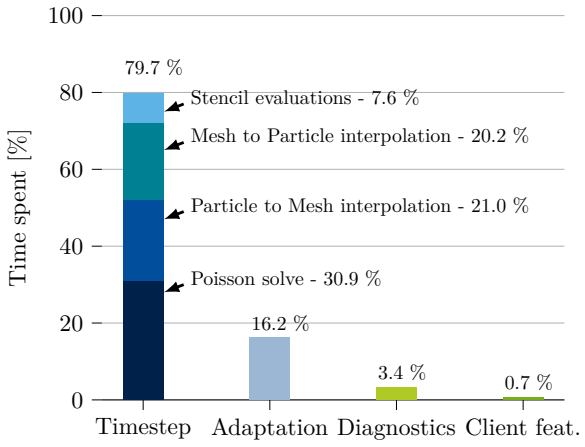
σ controls to the compactness of the distribution and σ_0 is the radius of the vorticity support. $\beta = \frac{\sigma}{\sigma_0}$ is a measure of the compact aspect of the ring. In this case, $\beta = 1$, which implies $\sigma = \sigma_0$. $C\left(\beta^2\right)$ is a parameter computed such that the total circulation is Γ and, here, $C\left(\beta^2\right) = 2.4774$. The rings are initially separated by a distance of $R + \sigma$, and their center, x_c, y_c, z_c, is located in the middle of the computational domain: $\{x_c, y_c, z_c\} = \{L_x/2, L_y/2, L_z/2 \pm (R+\sigma)/2\}$. The rings have opposite sign circulations, and their induced velocity pushes them towards the collision. The final step of the simulation is presented in Figure 14a. Time integration is carried out using a third-order low-storage Runge-Kutta scheme. Interpolations between the mesh and the particles are performed with the M_6^* kernel combined the wavelet 4.0 while the grid is adapted with the wavelet 6.0. Particles remeshing occurs every 5 timesteps, mesh adapted every 10 timesteps and the reprojection is performed every 20 timesteps. The domain is fully unbounded. While the physical analysis of the case is beyond the scope of this paper and is left as a topic for future work, the case is representative of the code usage and is therefore of great interest to characterize the time-to-solution of the solver.

The domain is initialized with a mean of 17,5 24^3-blocks (240k unknowns) per rank, and it slightly increases to reach 19 blocks (260k unknowns) per rank at most. The operations that advance the solution in time (gathered under the label "Timestep" in Figure 14b) are the most time-consuming. In particular, the interpolations and the resolution of the Poisson equation are the most expensive, accounting for 41% and 30% of the total computational time, respectively. The interpolation schemes thus constitute the largest share of the time-to-solution. This increased cost is primarily due to the use of the M_6^* kernel, which has a large support and therefore incurs a high computational overhead. In addition, these newly implemented schemes require both adapting the interpolated fields across resolution jumps and communicating the particles contributions to the neighboring block using MPI, thereby increasing their cost. This behavior contrasts with the uniform VPM framework, where the elliptic Poisson solve typically dominates the computational time [8]. In the AMR method, the computational cost per degree of freedom of the Poisson solver is also more important. The multiresolution nature of the grid requires elliptic solvers that handle highly localized sources, such as multigrid solvers, instead of more efficient FFT-based algorithms. These multigrid solvers inherently incur a higher cost per unknowns than the latter [63], thereby increasing the Poisson solver's computational cost. The grid adaptation, performed every ten timesteps, represents 16 % of the total. The other operations, such as the stencil evaluations needed to compute flow diagnostics or the remeshing of the particles and the reprojection of the vorticity onto a solenoidal space (gathered under the "Client features" label in Figure 14b) have a minor footprint on the time-to-solution of the solver.

Although these timings are sensitive to the simulated case and the grid configuration, the trend should essentially remain the same. Depending on the grid topology, some load imbalance may occur. The resolution of the Poisson equation is sensitive to the distribution of blocks per level. This metric relates the volume of fine resolution blocks compared to the volume of coarse resolution blocks. An uneven distribution of blocks per level will increase the load imbalance when solving the Poisson equation. The performance of



(a) Illustration of the grid topology and physics simulated in a representative production run.



(b) Breakdown of the mean time spent by each MPI process during the operations performed in a production simulation.

Figure 14: Example of production run: study of a vortex rings collision at Reynolds 10000.

the particle-mesh interpolation scheme also suffers from load imbalance. Since interpolations are computationally intensive, especially when particles cross interfaces of blocks with different resolutions, an uneven distribution of workload across MPI ranks can lead to idle processes waiting for the others to complete their tasks. The introduction of MPI communication within the scheme also introduces synchronization points, which further amplify the negative effects of poor load balancing.

6.2. Parallel efficiency

The second investigation consists in studying the weak scaling of the solver, i.e. when the computational resources are increased along with the size of the problem. The parallel efficiency of our framework is assessed from 1 to 128 nodes of the Lumi cluster, i.e. from 128 to 16384 MPI processes. The test case executes 60 timesteps of the four-vortex system presented in Section 5. The domain size, $S(N_n)$ is adjusted according to the number of nodes N_n to maintain the mean number of blocks per core constant: $S(N_n) = \{7, 5, 2 \times N_n\}$. The domain is initialized with an average of 11.4 blocks per core, which leads to 150k unknowns per core since each block owns 24^3 unknowns. We used the wavelet 4.0 combined with the M'_4 kernel for the particles mesh interpolation, while the wavelet 6.0 is employed for the grid adaptation and the computation of the ghost point on the mesh. The finite difference stencils are of order 4. The time integration is performed using a third-order low-storage Runge-Kutta scheme. The remeshing is performed every 3 timesteps, the grid is adapted every 6 timesteps, and we reproject the vorticity field in a solenoidal space every 12 timesteps. From a technical perspective, the code has been compiled with the *HPE Cray Compiling Environment* 17.0.1 for the MPI implementation, and the `libfabric` 15.2.0 from the vendor has been used for the `slingshot`-based interconnect.

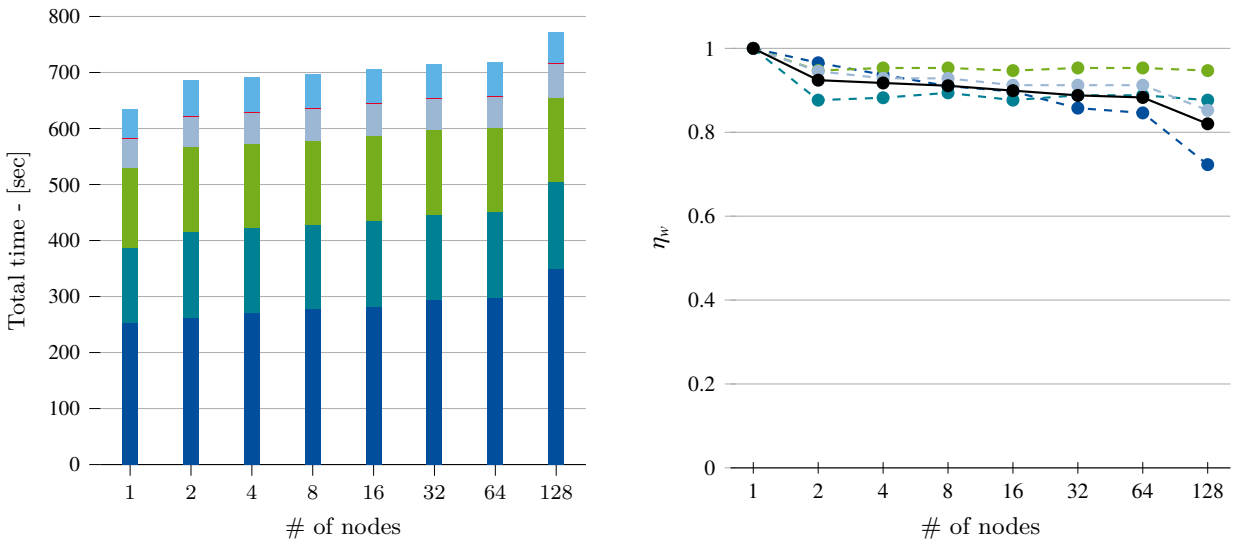
In light of the observations of the previous section, it is not surprising that the weak scaling behavior of the solver, shown in Figure 15, is largely dominated by the behavior of the interpolations, the Poisson solver, and the adaptation. The timings in Figure 15a show proportions close to those of Figure 14b for a small number of nodes. For higher node numbers, the time spent in the Poisson solver slightly increases and is followed by a sudden peak from 64 to 128 nodes (i.e. from 8192 to 16384 MPI processes). This performance drop is attributed to the domain's large aspect ratio combined with the large partition that results in high MPI traffic of irregularly sized messages. These conditions trigger a change in the `libfabric` communication protocol from the fast `hardware` mode to the slower but more robust `software` mode, due

to increased complexity in message matching. The interpolations exhibit an excellent scaling behavior and their timing remains constant. Based on these timings, the weak scaling efficiency is computed as

$$\eta_w(N_c) = \frac{T(N_{\text{ref}}, S(N_{\text{ref}}))}{T(N_c, S_c(N_c))}\,, \tag{19}$$

where $T(N, S(N))$ represents the time-to-solution of a simulation with a domain size $S(N)$ performed on N cores. Here, $N_{\text{ref}} = 128$, i.e., 1 node.

The weak scaling efficiency split by operations is illustrated in Figure 15b, and the curves confirm the analysis of the timings. The Poisson solver presents the worst weak efficiency, which is still above 80% for a problem size 64 times bigger, but drops to 70% for a problem size 128 times bigger due to the change in `libfabric` protocol. In contrast, the efficiency of the interpolations is almost perfect. Indeed, the interpolations, rely only on local communications that imply neighboring processes and present a great overlap of the communications with the computations while the Poisson solve requires global communication schemes. With 60 steps, the adaptation is not really stressed as the physics remains localized in the center of the computational domain. Yet, we do not expect the adaptation to significantly degrade the framework's efficiency. The first drop that appears from one to two nodes in all the curves results from the fact that the communication relies on fast memory transfer on one node, while on two nodes, some communications must go through the interconnect. Overall, the global efficiency of the framework remains above 80% for a problem that is 128 times bigger.



(a) Weak scaling: Total time spent in the Poisson solver( ), the particle-to-mesh interpolation( ), the mesh-to-particle interpolation( ), the adaptation( ), and the rest of the operations ( ).

(b) Weak scaling: efficiency η_w of the full simulation( ), the particle-to-mesh interpolation( ), the mesh-to-particle interpolation( ), the Poisson solver( ), and the adaptation( ).

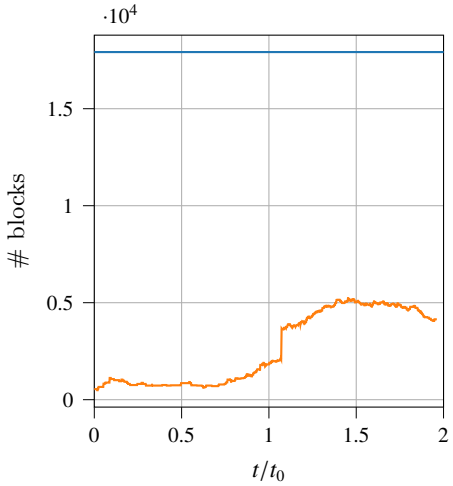
Figure 15: Parallel efficiencies of the Navier-Stokes solver.

6.3. Comparison with the uniform framework

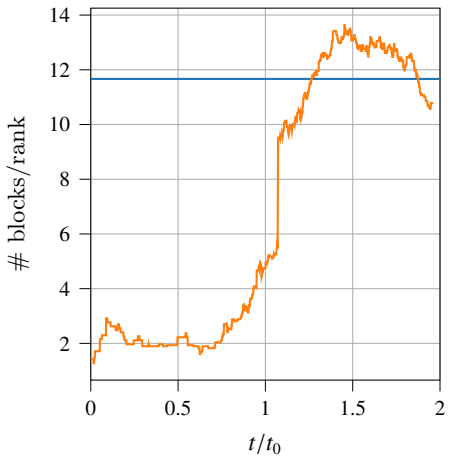
Finally, we exemplify the actual computational gains one can expect by directly comparing the AMR and the uniform frameworks, with the already mentioned caveat that the results of such a comparison are intrinsically dependent on the topology of the simulated problem. We here analyse the timings of the comparison between the AMR and uniform frameworks on the four-vortex case of Section 5.3.

On the one hand, the uniform resolution was initialized with a resolution of $1280 \times 1792 \times 256$ (587 million unknowns) distributed across 17920 blocks. It was executed with 1536 MPI processes, resulting in 11.6 blocks (382k unknowns) per rank. The simulation took 12.5 hours of wall-clock time, corresponding to a total computational cost of 19,200 CPUh.

On the other hand, the AMR simulation achieved a similar initial resolution with only 526 blocks, a 34-fold reduction in block count and thus resource usage. As discussed in Section 5.3 and shown in Figure 16a, the number of blocks toward the end of the AMR simulation was reduced by a factor of 4 compared to the uniform case. To maintain a similar blocks-per-rank ratio by the end of the simulation, we chose to run the AMR simulation with 384 MPI ranks, see Figure 16b.



(a) Evolution of the total number of blocks during the simulation of a four vortex system.



(b) Evolution of the mean number of blocks per rank during the simulation of a four vortex system.

Figure 16: Comparison of the number of blocks of a uniform (—) and a multiresolution (—) simulation.

Although the AMR simulation took 42.4 hours, 2.7 times longer than its uniform counterpart, it required only 16,287.4 CPUh in total. Overall, reducing the number of MPI ranks by a factor of 4 led to a 3.4-fold increase in wall time, but still reduced the overall computational cost by 20 %, all while preserving the same level of accuracy and dividing the memory footprint by approximately four.

In its current implementation, `murphy` constrains the load balancing by enforcing that all leaf blocks derived from the same parent remain assigned to a single MPI process. This constraint leads to a suboptimal workload distribution across ranks, which can be improved by increasing the number of blocks per rank. The benefits of the AMR framework could hence be further enhanced by running the simulation in two stages: an initial phase with a reduced number of MPI ranks, followed by a second one using more computational resources. For instance, the early stage of the simulation, when the number of blocks is low, could be executed using as few as 64 ranks. However, this approach would eventually lead to an *out-of-memory* event as the block count increases. Nevertheless, in real-world applications, such a multi-stage strategy is likely to become the norm, as it is often challenging to accurately predict resource requirements ahead of time.

7. Conclusions and perspectives

In this article, we have presented a 3*D* Vortex Particle-Mesh (VPM) method that benefits from adaptive mesh refinement (AMR) to focus the computational resources on the physics requirements, thereby decreasing the computational cost of the method. One of the main challenges in taking advantage of AMR in hybrid particle-mesh approaches is dealing with the interpolations between particles and meshes of different

resolution. To tackle it, we relied on `murphy`, a $3D$ wavelet-based block-structured multiresolution framework to handle the mesh adaptivity. As for the particles, we introduced a new connectivity that enables their refinement and coarsening using the existing grid-based tools. From there, we developed an interpolation scheme to transfer the information from the particles to the mesh and vice versa. We then thoroughly validated our new approach on hyperbolic problems and on unbounded incompressible Navier-Stokes simulations. In all the cases, the adaptation is performed based on the wavelet scale decomposition of the fields discretized on the mesh. For the fluid simulation, the adaptation is driven by the vorticity decomposition. We finally studied the parallel efficiency and scalability of our framework, and showed that we still achieve a 80 % weak efficiency for a problem that is 128 times bigger.

At a methodological level, the particle connectivity maps their reference configuration, aligned with the underlying multiresolution mesh, to their spatial (deformed) configuration. The reference configuration does not change in time and remains aligned with a Cartesian grid, which allows us to use the `murphy` wavelet machinery to coarsen or refine the particle set. This is performed by considering both carried quantities and Eulerian positions as fields described in the reference (Lagrangian) frame. The transfer of data between the particles and the mesh is done by adapting the resolution of the source field to the resolution of the target. Once both fields are at the same resolution, we rely on the usual interpolation kernels of particle-mesh method. With such an approach, the convergence order of the interpolation between the mesh and the particles is bounded by the minimum between the wavelet order and the interpolation kernel order.

Our hybrid particle-mesh framework has been tested first on linear and non-linear hyperbolic problems using solenoidal velocity fields. It showed that the refinement threshold is a high-quality estimation of the error. Reducing this user-controlled threshold leads to a convergence of the error that presents a slope proportional to the ratio of the interpolation kernel order to the polynomial order of the wavelets. When the interpolation kernel order is smaller than the wavelet order, the accuracy of the multiresolution method is close to the uniform one. When the kernel order is higher than the wavelet order, the method overall convergence is reduced because the wavelet error dominates. Yet, the error continues to decrease with the grid refinement. We also observed that the interpolation kernels interfere with the wavelet multiresolution analysis, which results in comparable grid topologies for both lifted and non-lifted wavelet schemes. We then validate the framework by comparing diagnostics of multiresolution and uniform-resolution simulations of a four-vortex system. The results demonstrate that our AMR VPM method matches the local and global diagnostics of the uniform method while reducing the number of unknowns by a factor of 32 in the initial times and by a factor of 8 at the end of the simulations. In terms of performance, the three most expensive operations are the interpolations between the mesh and the particles, the resolution of the Poisson equation, and the grid adaptation. The interpolations have an almost perfect weak efficiency, while the elliptic solver and the adaptation, requiring global communications, see their efficiency reduced by the increase of computational resources. Overall, the solver efficiency remains above 80% with 128 times more resources. Compared to the uniform resolution, the AMR framework divided the simulation cost by 1.2 when looking at the CPUh and the memory footprint by approximately 4, yet this result highly depends on the simulated case.

At an implementation level, future works will focus on maximizing resource utilization and further reducing the computational cost of the method. This involves relaxing two key constraints. The first arises from the elliptic solver, which currently requires all blocks located at domain boundaries to remain at the coarsest refinement level. When fine-scale structures approach the boundary and trigger the refinement of a boundary block, the entire coarsest level of the grid must be refined. This constraint often leads to a resolution finer than necessary in the outer regions of the domain, thus resulting in an inefficient use of computational resources. The second constraint affects load balancing: all leaf blocks originating from the same parent must reside on the same MPI rank. Removing this limitation would allow for a more flexible distribution of the workload, thereby improving the overall resource usage.

Finally, the present wavelet-based refinement and coarsening criteria are agnostic to the physics of the underlying PDE. Further work is required to develop an automatic criterion based on the Navier-Stokes physics to adequately capture emerging fine-scale features or discard negligible information. More broadly,

a deeper understanding of the behavior of the wavelet-scale decomposition within the NS equations could also help assess whether the wavelet-scale decomposition could be efficiently combined with large eddy simulation (LES). Indeed, LES involves modeling the influence of unresolved small-scale structures on the resolved ones, hence a significant challenge since the definition of the resolved scales varies in space in a multiresolution context.

Acknowledgments

We would like to acknowledge the insightful discussions with Jonathan Lambrecht (UCLouvain), Gilles Poncelet (UCLouvain), Thomas Gillis(MIT, UCLouvain), and Wim van Rees (MIT). Computational resources have been provided by the Consortium des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11 and by the Walloon Region. The present research also benefited from computational resources made available on Lucia, the Tier-1 supercomputer of the Walloon Region, infrastructure funded by the Walloon Region under the grant agreement n°1910247. Additional computational resources have been provided by EuroHPC for the access to LUMI-C (EHPC-DEV-2025D01-070).

Declaration of generative AI and AI-assisted technologies in the writing process

During the writing process, the author(s) used Microsoft Copilot to improve readability and language. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

References

[1] T. Gillis, W. M. van Rees, Murphy-a scalable multiresolution framework for scientific computing on 3d block-structured collocated grids, SIAM Journal on Scientific Computing 44 (5) (2022) C367–C398. arXiv:https://doi.org/10.1137/21M141676X, doi:10.1137/21M141676X.
URL https://doi.org/10.1137/21M141676X

[2] G.-H. Cottet, P. Koumoutsakos, Vortex Methods, Theory and Practice, Cambridge University Press, 2000.

[3] G. S. Winckelmans, Encyclopedia of Computational Mechanics, John Wiley & Sons, Ltd, 2004, Ch. Vortex Methods, pp. 129–153. doi:10.1002/0470091355.ecm055.

[4] M. Gazzola, P. Chatelain, W. M. van Rees, P. Koumoutsakos, Simulations of single and multiple swimmers with non-divergence free deforming geometries, Journal of Computational Physics 230 (19) (2011) 7093–7114.

[5] W. M. van Rees, M. Gazzola, P. Koumoutsakos, Optimal morphokinematics for undulatory swimmers at intermediate Reynolds numbers, Journal of Fluid Mechanics 775 (2015) 178–188.

[6] P. Chatelain, M. Duponcheel, D.-G. Caprace, Y. Marichal, G. Winckelmans, Vortex particle-mesh simulations of vertical axis wind turbine flows: from the airfoil performance to the very far wake, Wind Energy Science 2 (1) (2017) 317. doi:10.5194/wes-2-317-2017.

[7] P. Balty, D.-G. Caprace, J. Waucquez, M. Coquelet, P. Chatelain, Multiphysics simulations of the dynamic and wakes of a floating vertical axis wind turbine, in: TORQUE, Vol. 1618, IOP Publishing, Delft, 2020, p. 062053. doi:10.1088/1742-6596/1618/6/062053.

[8] P. Chatelain, A. Curioni, M. Bergdorf, D. Rossinelli, W. Andreoni, P. Koumoutsakos, Billion vortex particle Direct Numerical Simulations of aircraft wakes, Computer Methods in Applied Mechanics and Engineering 197 (13) (2008) 1296–1304. doi:10.1016/j.cma.2007.11.016.

[9] D.-G. Caprace, S. Buffin, M. Duponcheel, P. Chatelain, G. Winckelmans, Large Eddy Simulation of Advancing Rotor for Near to Far Wake Assessment, in: 43rd European Rotorcraft Forum, ERF, Milan, Italy, 2017, p. 570.

[10] G.-H. Cottet, D. Jiroveanu, B. Michaux, Vorticity dynamics and turbulence models for large-eddy simulations, Modélisation Mathematique et Analyse Numérique (2002).

[11] W. M. van Rees, A. Leonard, D. I. Pullin, P. Koumoutsakos, A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high Reynolds numbers, Journal of Computational Physics 230 (8) (2011) 2794–2805. doi:10.1016/j.jcp.2010.11.031.

[12] J.-M. Etancelin, G.-H. Cottet, F. Pérignon, C. Picard, Multi-cpu and multi-gpu hybrid computations of multi-scale scalar transport, in: Proceedings of the 26th International Conference on Parallel Computational Fluid Dynamics, Trondheim, Norway, 2014, pp. 83–84.

[13] D. Rossinelli, B. Hejazialhosseini, W. van Rees, M. Gazzola, M. Bergdorf, P. Koumoutsakos, MRag-I2d: Multi-resolution adapted grids for remeshed vortex methods on multicore architectures, Journal of Computational Physics 288 (2015) 1–18. doi:http://dx.doi.org/10.1016/j.jcp.2015.01.035.
[14] J.-B. Keck, Numerical modelling and High Performance Computing for sediment flows, Theses, Université Grenoble Alpes (Dec. 2019).
URL https://theses.hal.science/tel-02433509
[15] T. Y. Hou, Convergence of a variable blob vortex method for the Euler and Navier-Stokes equations, SIAM Journal on Numerical Analysis 27 (1990) 1387–1404.
[16] P. Ploumhans, G. S. Winckelmans, Vortex Methods for High-Resolution Simulations of Viscous Flow Past Bluff Bodies of General Geometry, Journal of Computational Physics 165 (2) (2000) 354–406. doi:10.1006/jcph.2000.6614.
[17] G.-H. Cottet, P. Koumoutsakos, M. L. O. Salihi, Vortex methods with spatially varying cores, Journal of Computational Physics 162 (1) (2000) 164–185. doi:https://doi.org/10.1006/jcph.2000.6531.
URL https://www.sciencedirect.com/science/article/pii/S0021999100965318
[18] M. Bergdorf, G.-H. Cottet, P. Koumoutsakos, Multilevel adaptive particle methods for convection-diffusion equations, Multiscale Model. Simul. 4 (1) (2005) 328–357.
[19] C. Roy, Strategies for Driving Mesh Adaptation in CFD (Invited), no. 0 in Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, 2009. doi:doi:10.2514/6.2009-1302.
URL https://doi.org/10.2514/6.2009-1302
[20] M. Adams, P. Colella, D. T. Graves, J. Johnson, N. Keen, T. J. Ligocki, D. F. Martin, P. McCorquodale, D. Modiano, P. Schwartz, T. Sternberg, B. V. Straalen, Chombo software package for amr applications design document, Tech. rep., Lawrence Berkeley National Laboratory (2019).
[21] R. D. Hornung, S. R. Kohn, Managing application complexity in the samrai object-oriented framework, Concurrency and Computation: Practice and Experience 14 (5) (2002) 347–368. doi:https://doi.org/10.1002/cpe.652.
URL https://doi.org/10.1002/cpe.652
[22] W. Zhang, A. Almgren, V. Beckner, J. Bell, J. Blaschke, C. Chan, M. Day, B. Friesen, K. Gott, D. Graves, M. P. Katz, A. Myers, T. Nguyen, A. Nonaka, M. Rosso, S. Williams, M. Zingale, Amrex: a framework for block-structured adaptive mesh refinement, The Journal of Open Source Software (2019).
[23] M. J. Berger, J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, Journal of Computational Physics 53 (3) (1984) 484–512. doi:10.1016/0021-9991(84)90073-1.
URL http://www.sciencedirect.com/science/article/pii/0021999184900731
[24] K. Yu, B. Dorschner, T. Colonius, Multi-resolution lattice green's function method for incompressible flows (10 2020).
[25] W. Hou, T. Colonius, An adaptive lattice green's function method for external flows with two unbounded and one homogeneous directions, Journal of Computational Physics 519 (2024) 113370. doi:https://doi.org/10.1016/j.jcp.2024.113370.
URL https://www.sciencedirect.com/science/article/pii/S0021999124006181
[26] T. Tu, D. R. O'Hallaron, O. Ghattas, Scalable Parallel Octree Meshing for TeraScale Applications, Vol. 2005, 2005. doi:10.1109/SC.2005.61.
[27] H. Sundar, R. S. Sampath, G. Biros, Bottom-up construction and 2:1 balance refinement of linear octrees in parallel, SIAM Journal on Scientific Computing 30 (5) (2008) 2675–2708. doi:10.1137/070681727.
URL https://doi.org/10.1137/070681727
[28] C. Burstedde, L. Wilcox, O. Ghattas, p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees, SIAM Journal on Scientific Computing 33 (3) (2011) 1103–1133. doi:10.1137/100791634.
URL https://doi.org/10.1137/100791634
[29] S. Popinet, Gerris: a tree-based adaptive solver for the incompressible euler equations in complex geometries, Journal of Computational Physics 190 (2) (2003) 572–600. doi:https://doi.org/10.1016/S0021-9991(03)00298-5.
URL https://www.sciencedirect.com/science/article/pii/S0021999103002985
[30] S. Popinet, A quadtree-adaptive multigrid solver for the serre–green–naghdi equations, Journal of Computational Physics 302 (2015) 336–358. doi:https://doi.org/10.1016/j.jcp.2015.09.009.
URL https://www.sciencedirect.com/science/article/pii/S0021999115005902
[31] J. A. van Hooft, S. Popinet, A fourth-order accurate adaptive solver for incompressible flow problems, Journal of Computational Physics 462 (2022) 111251. doi:https://doi.org/10.1016/j.jcp.2022.111251.
URL https://www.sciencedirect.com/science/article/pii/S0021999122003138
[32] F. Schornbaum, U. Rüde, Extreme-scale block-structured adaptive mesh refinement, SIAM Journal on Scientific Computing 40 (04 2017). doi:10.1137/17M1128411.
[33] D. Jude, J. Sitaraman, A. Wissink, An octree-based, cartesian navier–stokes solver for modern cluster architectures, The Journal of Supercomputing 78 (06 2022). doi:10.1007/s11227-022-04324-7.
[34] S. G. Mallat, A theory for multiresolution signal decomposition: the wavelet representation, IEEE Transactions on Pattern Analysis and Machine Intelligence 11 (7) (1989) 674–693. doi:10.1109/34.192463.
[35] K. Schneider, O. V. Vasilyev, Wavelet methods in computational fluid dynamics, Annual Review of Fluid Mechanics 42 (1) (2009) 473–503. doi:10.1146/annurev-fluid-121108-145637.
URL https://doi.org/10.1146/annurev-fluid-121108-145637
[36] T. Engels, K. Schneider, J. Reiss, M. Farge, A 3d wavelet-based incompressible navier-stokes solver for fully adaptive computations in time-varying geometries, arXiv:1912.05371 (December 2019).
[37] J. T. Rasmussen, G.-H. Cottet, J. H. Walther, A multiresolution remeshed Vortex-In-Cell algorithm using patches, Journal of Computational Physics 230 (2011) 6742–6755. doi:10.1016/j.jcp.2011.05.006.

[38] T. Lonfils, Vortex particle-mesh method with combined immersed boundary and mesh refinement techniques. Application to bluff-body and wake vortex flows, Ph.D. thesis, Université catholique de Louvain (2011).
[39] M. E. Ossmani, P. Poncet, Efficiency of multiscale hybrid grid-particle vortex methods, Multiscale Modeling & Simulation 8 (5) (2010) 1671–1690. doi:10.1137/090765006.
URL https://doi.org/10.1137/090765006
[40] M. Bergdorf, P. Koumoutsakos, A Lagrangian Particle-Wavelet Method, Multiscale Modeling & Simulation: A SIAM Interdisciplinary Journal 5 (3) (2006) 980–995.
[41] C. Mimeau, I. Mortazavi, A review of vortex methods and their applications: From creation to recent advances, Fluids 6 (2) (2021). doi:10.3390/fluids6020068.
URL https://www.mdpi.com/2311-5521/6/2/68
[42] D.-G. Caprace, G. Winckelmans, P. Chatelain, An immersed lifting and dragging line model for the vortex particle-mesh method, Theoretical and Computational Fluid Dynamics 34 (2020) 21–48. doi:10.1007/s00162-019-00510-1.
[43] P. Balty, P. Chatelain, T. Gillis, Flups - a flexible and performant massively parallel fourier transform library, IEEE Transactions on Parallel & Distributed Systems 34 (07) (2023) 2011–2024. doi:10.1109/TPDS.2023.3254302.
URL http://doi.ieeecomputersociety.org/10.1109/TPDS.2023.3254302
[44] J. T. Beale, A. Majda, Vortex Methods I: convergence in 3 dimensions, Mathematics of Computation 39 (159) (1982) 1–27.
[45] J. T. Beale, On the accuracy of vortex methods at large times, in: B. E. et al. (Ed.), Proc. Workshop on Comput. Fluid Dyn. and React. Gas Flows, IMA, Univ. of Minnesota, 1986, Springer-Verlag, New York, 1988, p. 19.
[46] P. Koumoutsakos, Inviscid axisymmetrization of an elliptical vortex, Journal of Computational Physics 138 (2) (1997) 821–857. doi:10.1006/jcph.1997.5749.
[47] R. Cocle, G. Winckelmans, G. Daeninck, Combining the vortex-in-cell and parallel fast multipole methods for efficient domain decomposition simulations, Journal of Computational Physics 227 (21) (2008) 9091–9120. doi:10.1016/j.jcp.2007.10.010.
[48] J. Monaghan, Extrapolating B splines for interpolation, Journal of Computational Physics 60 (2) (1985) 253 – 262. doi:10.1016/0021-9991(85)90006-3.
[49] M. Bergdorf, Multiresolution Particle Methods for the Simulation of Growth and Flow, Ph.D. thesis, ETH, Zurich (2007).
[50] G. S. Winckelmans, A. Leonard, Contributions to Vortex Particle Methods for the Computation of Three-Dimensional Incompressible Unsteady Flows;, Journal of Computational Physics 109 (2) (1993) 247–273.
[51] D. Donoho, Interpolation wavelet transforms, Tech. rep., Stanford University (1992).
[52] A. Cohen, I. Daubechies, J.-C. Feauveau, Biorthogonal bases of compactly supported wavelets, Communications on Pure and Applied Mathematics 45 (5) (1992) 485–560. doi:10.1002/cpa.3160450502.
URL http://dx.doi.org/10.1002/cpa.3160450502
[53] W. Sweldens, R. Piessens, Quadrature formulae and asymptotic error expansions for wavelet approximations of smooth functions, SIAM Journal on Numerical Analysis 31 (4) (1994) 1240–1264.
URL http://www.jstor.org/stable/2158124
[54] W. Sweldens, The Lifting Scheme: A construction of Second Generation Wavelets, SIAM Journal on Mathematical Analysis 29 (2) (1998) 511–546. arXiv:http://dx.doi.org/10.1137/S0036141095289051, doi:10.1137/S0036141095289051.
[55] W. Sweldens, The lifting scheme: a custom-design construction of biorthogonal wavelets, Applied and Computational Harmonic Analysis 3 (1996) 186–200.
[56] Y. Marichal, P. Chatelain, G. Winckelmans, Immersed interface interpolation schemes for particle–mesh methods, Journal of Computational Physics 326 (2016) 947 – 972. doi:http://dx.doi.org/10.1016/j.jcp.2016.09.027.
[57] G. Poncelet, J. Lambrecht, T. Gillis, P. Chatelain, A high-order adaptive multiresolution unbounded multigrid poisson solver, To be determined (2025).
[58] D.-G. Caprace, T. Gillis, P. Chatelain, FLUPS - a Fourier-based library of unbounded Poisson solvers, arXiv:2006.09300 (2020).
URL https://arxiv.org/abs/2006.09300
[59] J. H. Williamson, Low-storage Runge-Kutta schemes, Journal of Computational Physics 35 (1) (1980) 48–56.
[60] R. J. Leveque, High-resolution conservative algorithms for advection in incompressible flow, SIAM Journal on Numerical Analysis 33 (2) (1996) 627–665. arXiv:https://doi.org/10.1137/0733033, doi:10.1137/0733033.
URL https://doi.org/10.1137/0733033
[61] D. Enright, R. Fedkiw, J. Ferziger, I. Mitchell, A hybrid particle level set method for improved interface capturing, Journal of Computational Physics 183 (1) (2002) 83–116. doi:https://doi.org/10.1006/jcph.2002.7166.
URL https://www.sciencedirect.com/science/article/pii/S0021999102971664
[62] S. C. Crow, Stability theory for a pair of trailing vortices, AIAA Journal 8 (12) (1970) 2172–2179. doi:10.2514/3.6083.
[63] A. Gholami, D. Malhotra, H. Sundar, G. Biros, FFt, Fmm, or Multigrid? A comparative Study of State-Of-the-Art Poisson Solvers for Uniform and Nonuniform Grids in the Unit Cube, SIAM Journal on Scientific Computing 38 (3) (2016) C280–C306. arXiv:https://doi.org/10.1137/15M1010798, doi:10.1137/15M1010798.