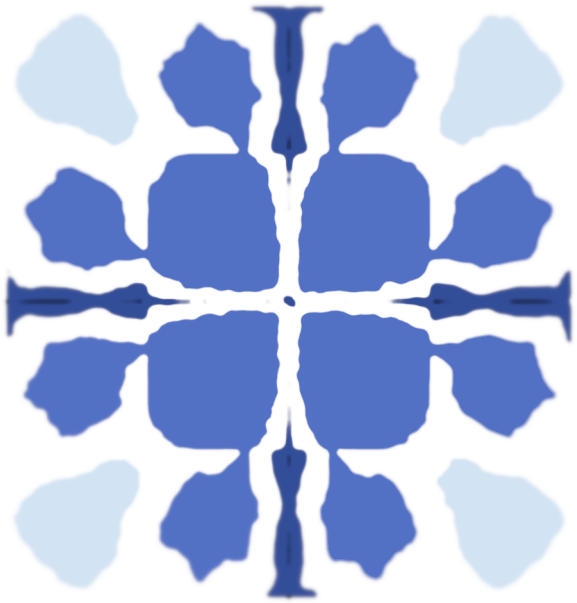
Assessment of a GPU accelerated Cartesian Fast Multipole Method

Accuracy and performance analysis of Cartesian FMM applied on the vortex particle method

Michael Kamal Rizk

Delft University of Technology



TU Delft

Assessment of a GPU accelerated Cartesian Fast Multipole Method

Accuracy and performance analysis of Cartesian FMM applied on the vortex particle method

by

Michael Kamal Rizk

in partial fulfilment of the requirements for the degree of

Master of Science
in Aerospace Engineering

at the Delft University of Technology
to be defended publicly on January 21, 2026.

Student Number:	5008883
Responsible Supervisor:	Dr.ir. A. van Zuijlen
Supervisor:	F. Martins, M.A.Sc.
Chair:	Dr.ir. M.I. Gerritsma
External Examiner:	Prof.dr.ir. L.L.M. Veldhuis
Project Duration:	April 2025 - January 2026
Faculty:	Aerospace Engineering, Delft

Cover:	FMM relative error contour (Figure 4.6b) inspired by Delft Blue art
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Abstract

Vortex particle methods (VPM) have historically faced computational bottlenecks caused by the pairwise interactions of particles in solving the vorticity-velocity coupling. Several algorithms have been developed to address this bottleneck, including the Fast Multipole Method (FMM). This work describes the development of a FMM solver which is integrated within an in-house developed open-source VPM solver. The FMM improves the computational efficiency of particle velocities and velocity gradients from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$. The formulation uses a full octree structure and a Cartesian formulation of multipole and local expansions based on Taylor series.

Inline with the increasing adoption of GPUs in performing scientific computations, the solver developed assigns computationally heavy tasks to parallelised execution on the CPU or GPU, using the taichi compiler. The accompanying introduction of reduced floating point accuracy introduces new considerations for the accuracy of the method when performed on the GPU. Error quantification is performed on a single timestep for a range of flow scenarios, with an emphasis on the effect of main FMM parameters on accuracy. Results indicate a positive impact of increasing the Taylor series truncation order on FMM accuracy, up to a plateau caused by the accumulation of rounding errors. The octree depth was found to have low impact on the accuracy.

Validation of the solver is performed through underresolved direct numerical simulation (DNS) of Lamb-Oseen vortex, vortex ring, and colliding vortex ring test cases. The impact of FMM on particle velocities and positions throughout simulations is measured, and key flow diagnostics are provided, indicating minimal FMM impact on the physical validity of simulations. More chaotic flows exhibited an unbounded nature of FMM error growth while low Reynolds number flows counteracted the FMM error. The performance of the main operations of the solver was profiled, highlighting trade-offs between speed and accuracy, and indicating areas for future performance improvements.

Acknowledgements

At long last!! the final piece of my 6 year journey at TU Delft, and if I could have written these formative years of my adult life into a book before they unfold, I don't think I could think of such a fun, unpredictable, challenging, solacing, and often times farcical set of events. My time at TU Delft has taught me that, when faced with challenges, there is no way up, down or sideways, you only get better by getting through.

Of course, if my time here was a book, then it would contain the best set of characters I can imagine. Those who have given me nothing but love, support and made my days here so much more enjoyable. To my friends, both from TU Delft, from TSH, and everywhere else, thank you for the overly fancy homemade dinners, for joining our unwarranted coffee breaks at the most minor of inconveniences, for listening intently to my rants, for being all you are. Thanks to all my professors for your encouraging and supportive attitude, and especially my thesis supervisors, Sander and Flavio, you truly were generous in your support and advice during my thesis. Sincerely, I could not have picked better supervisors. I hope many more students continue to learn and be inspired from you like I did. To my family, Mama, Maro, Teta and Gedo, whom I owe all of this, words of thanks might not be enough. Honestly, I just hope I can repay an inch of the miles of love and support you have given me, and all your admirable qualities that continue to inspire me throughout my life.

The God of heaven will prosper us. Therefore we, His servants, will arise and build. (Nehemiah 2:20)

Contents

Abstract ii

Acknowledgements iv

Nomenclature xiii

1 Introduction 1

2 Background and Literature Review 3
2.1 Vortex Particle Method . . . 3
2.1.1 Formulation of the Vortex Particle Method . . . 3
2.1.2 Large Eddy Simulations with VPM . . . 8
2.1.3 OpenONDA . . . 8
2.2 Treecodes and the Fast Multipole Method . . . 9
2.2.1 Hierarchical Treecodes . . . 9
2.2.2 Fast Multipole Method . . . 9
2.2.3 Available FMM Repositories . . . 10
2.3 Project Objective and Research Questions . . . 11

3 Fast Multipole Method 14
3.1 Octree hierarchical structures . . . 14
3.1.1 Morton Codes . . . 15
3.1.2 Finding Neighbour and Interaction Clusters . . . 16
3.2 FMM Algorithm . . . 16
3.3 Multipole and Local Expansions . . . 18
3.3.1 Particle to Particle (P2P) . . . 19
3.3.2 Particle to Multipole (P2M) . . . 19
3.3.3 Multipole to Multipole (M2M) . . . 20
3.3.4 Multipole to Local (M2L) . . . 20
3.3.5 Local to Local (L2L) . . . 21
3.3.6 Local to Particle (L2P) . . . 21
3.4 Performance considerations of the FMM code . . . 23
3.4.1 The Taichi Language . . . 23
3.4.2 Computing Taylor Coefficients Using Recurrence Relations . . . 24

4 Error Quantification 26
4.1 Error quantification method . . . 26
4.2 Description of the Test Cases . . . 28
4.2.1 Lamb-Oseen Vortex . . . 28
4.2.2 Vortex Ring . . . 29
4.3 Effect of the number of particles . . . 31
4.3.1 Lamb-Oseen Vortex with random particle positions . . . 31
4.3.2 Lamb-Oseen Vortex with uniform particle positions . . . 34
4.3.3 Vortex ring with random particle positions . . . 35
4.3.4 Vortex ring with particles positioned in a torus . . . 36
4.4 Effect of particle overlap ratio . . . 37
4.5 Effect of octree depth . . . 38

5 Validation 40
5.1 Lamb-Oseen Vortex . . . 40
5.1.1 Accuracy of FMM estimates . . . 41
5.1.2 Conformity with Diagnostics . . . 44

5.1.3 Accuracy of velocity gradients using CDS scheme . . . 45
5.2 Vortex Ring . . . 45
5.2.1 Accuracy of FMM estimates . . . 46
5.2.2 Conformity with diagnostics . . . 47
5.2.3 Accuracy of velocity gradients using CDS scheme . . . 48
5.3 Colliding Vortex Rings . . . 49
5.3.1 Accuracy of FMM estimates . . . 50
5.3.2 Conformity with Diagnostics . . . 52
5.3.3 Accuracy of velocity gradients using CDS scheme . . . 53

6 Performance Analysis **54**
6.1 Time complexity of FMM and Direct method . . . 54
6.2 Performance breakdown of FMM operations on GPU device . . . 56
6.3 Performance breakdown of FMM operations on CPU device . . . 57
6.4 A rudimentary approach to depth selection . . . 58

7 Conclusions **60**

8 Future Recommendations **62**
8.1 Improvements to the FMM solver . . . 62
8.2 Further Research . . . 62

References **64**

A Supplementary statistics from FMM error quantification **68**
A.1 Lamb-Oseen Vortex: Random Particle Distribution . . . 68
A.2 Lamb-Oseen Vortex: Uniform Particle Distribution . . . 70
A.3 Vortex Ring: Random Particle Distribution . . . 72
A.4 Vortex Ring: Toroidal Particle Distribution . . . 74

List of Figures

3.1 Illustration of the neighbouring clusters and interaction list on a quadtree with a depth of 3. . . . 15
3.2 An illustration of Morton codes for clusters in level 1, and for the clusters within cluster 111 at level 2 . . . 16
3.3 Flowchart of implemented FMM algorithm . . . 18
3.4 Illustration of P2M operation . . . 20
3.5 Illustration of P2M and M2M operations in 2D. Particles are presented by the violet dots, while cluster centers are presented by the black dots. . . . 20
3.6 Illustration of L2L and L2P operations in 2D. Particles are presented by the violet dots, while cluster centres are presented by the black dots. . . . 21
3.7 Illustration of L2P operation . . . 22
3.8 2D representation of CDS stencil . . . 23
3.9 Order of calculation of Taylor coefficients $T^{\nu}_{\boldsymbol{k}}$ . . . 25

4.1 Example of supplementary statistics for FMM quality analysis . . . 27
4.2 Illustration of a vortex ring and its defining parameters. An arbitrary particle within the ring is shown in violet. A vortex ring with this orientation of vorticity translates in the positive x direction. . . . 30
4.3 RMSE* of FMM estimates with varying p and N (Lamb-Oseen vortex, $d = 2$, $\sigma/h = 1.25$, random particle distribution, constant particle density) . . . 31
4.4 RMSE* of FMM estimates with varying p and N (Lamb-Oseen vortex, $d = 2$, $\sigma/h = 1.25$, random particle distribution, constant particle density, double precision) . . . 32
4.5 Relative error per velocity component for single and double precision FMM (Lamb-Oseen vortex, $N = 1 \times 10^5$, $d = 2$, $p = 10$, random particle distribution) . . . 33
4.6 Velocity and relative error $X - Y$ contours for a Lamb-Oseen vortex with a centered core ($p = 2$) . . . 33
4.7 Velocity and relative error $X - Y$ contours for a Lamb-Oseen vortex with a displaced core ($p = 2$) . . . 34
4.8 RMSE of FMM estimates with varying p and N (Lamb-Oseen vortex, $d = 2$, uniform particle distribution, constant particle density) . . . 35
4.9 RMSE of FMM estimates with varying p and N (vortex ring, $d = 2$, random particle distribution, constant particle density) . . . 35
4.10 RMSE of FMM estimates with varying p and N (vortex ring, $d = 2$, toroidal particle distribution, constant domain volume) . . . 36
4.11 RMSE of FMM estimates for different overlap ratios σ/h (Lamb-Oseen vortex, $d = 2$, $p = 10$, constant particle density) . . . 37
4.12 RMSE of FMM estimates for different octree depths d (Lamb-Oseen vortex, random particle distribution, constant particle density) . . . 39

5.1 Top view of Lamb-Oseen vortex at $t/t_\nu = 6.67$ for differing Reynolds numbers . . . 41
5.2 Velocities RMSE* for $Re = 100$ Lamb-Oseen VPM simulation . . . 42
5.3 Positions RMSE* for $Re = 100$ Lamb-Oseen VPM simulation . . . 42
5.4 Velocity gradients RMSE for $Re = 100$ Lamb-Oseen VPM simulation . . . 42
5.5 Velocities RMSE* for $Re = 1000$ Lamb-Oseen VPM simulation . . . 43
5.6 Positions RMSE* for $Re = 1000$ Lamb-Oseen VPM simulation . . . 43
5.7 Velocity gradients RMSE for $Re = 1000$ Lamb-Oseen VPM simulation . . . 43
5.8 VPM diagnostics for $Re = 1000$ Lamb-Oseen test case . . . 44
5.9 $\Delta\text{RMSE}^{\text{CDS}}_{\nabla\mathbf{u}}$ for $Re = 1000$ Lamb-Oseen VPM simulation . . . 45
5.10 Velocities RMSE* for $Re = 1000$ vortex ring VPM simulation . . . 46

5.11 Positions RMSE* for $Re = 1000$ vortex ring VPM simulation 46
5.12 Velocity gradients RMSE for $Re = 1000$ vortex ring VPM simulation 47
5.13 VPM diagnostics for $Re = 1000$ vortex ring test case 48
5.14 $\Delta\text{RMSE}^{\text{CDS}}_{\nabla\mathbf{u}}$ for $Re = 1000$ vortex ring VPM simulation 48
5.15 Evolution of colliding vortex rings at different simulation phases 50
5.16 Velocities RMSE* for $Re = 1000$ colliding vortex rings VPM simulation 50
5.17 Positions RMSE* for $Re = 1000$ colliding vortex rings VPM simulation 50
5.18 Velocity gradients RMSE for $Re = 1000$ colliding vortex rings VPM simulation 51
5.19 VPM diagnostics for $Re = 1000$ collding vortex rings test case 52
5.20 $\Delta\text{RMSE}^{\text{CDS}}_{\nabla\mathbf{u}}$ for $Re = 1000$ colliding vortex rings VPM simulation 53

6.1 Single timestep calculation time plot for a saturated domain on CUDA backend (GPU) 54
6.2 Single timestep calculation time plot for a saturated domain on CPU backend 55
6.3 Computation time breakdown of FMM operations for a saturated domain at $d = 2$ (GPU) 56
6.4 Computation time breakdown of FMM operations for a saturated domain at $d = 3$ (GPU) 56
6.5 Computation time breakdown of FMM operations for a toroidal particle concentration at $d = 2$ (GPU) 57
6.6 Computation time breakdown of FMM operations for a toroidal particle concentration at $d = 3$ (GPU) 57
6.7 Computation time breakdown of FMM operations for a saturated domain at $d = 2$ (CPU) 58
6.8 Computation time breakdown of FMM operations for a saturated domain at $d = 3$ (CPU) 58

A.1 Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^4$, $d = 2$, $p = 2$, random particle distribution 68
A.2 Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^4$, $d = 2$, $p = 10$, random particle distribution 69
A.3 Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^6$, $d = 2$, $p = 2$, random particle distribution 69
A.4 Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^6$, $d = 2$, $p = 10$, random particle distribution 70
A.5 Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^4$, $d = 2$, $p = 2$, uniform particle distribution 70
A.6 Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^4$, $d = 2$, $p = 10$, uniform particle distribution 71
A.7 Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^6$, $d = 2$, $p = 2$, uniform particle distribution 71
A.8 Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^6$, $d = 2$, $p = 10$, uniform particle distribution 72
A.9 Statistical figures for vortex ring, $N = 1 \times 10^4$, $d = 2$, $p = 2$, random particle distribution 72
A.10 Statistical figures for vortex ring, $N = 1 \times 10^4$, $d = 2$, $p = 10$, random particle distribution 73
A.11 Statistical figures for vortex ring, $N = 1 \times 10^6$, $d = 2$, $p = 2$, random particle distribution 73
A.12 Statistical figures for vortex ring, $N = 1 \times 10^6$, $d = 2$, $p = 10$, random particle distribution 74
A.13 Statistical figures for vortex ring, $N = 1 \times 10^4$, $d = 2$, $p = 2$, toroidal particle distribution 74
A.14 Statistical figures for vortex ring, $N = 1 \times 10^4$, $d = 2$, $p = 10$, toroidal particle distribution 75
A.15 Statistical figures for vortex ring, $N = 1 \times 10^6$, $d = 2$, $p = 2$, toroidal particle distribution 75
A.16 Statistical figures for vortex ring, $N = 1 \times 10^6$, $d = 2$, $p = 10$, toroidal particle distribution 76

List of Tables

2.1 Smoothing functions and their kernels [51] 5
2.2 Comparison of different FMM formulations 12

4.1 Default parameters of Lamb-Oseen single timestep runs 29
4.2 Default parameters of vortex ring single timestep runs 30

5.1 VPM settings for the Lamb-Oseen cases 40
5.2 VPM settings for the vortex ring cases 46
5.3 VPM settings for the colliding vortex rings cases 49

Nomenclature

Acronyms

BBFMM	Black-Box Fast Multipole Method
CDS	Central Difference Scheme
CFD	Computational Fluid Dynamics
CFL	Courant–Friedrichs–Lewy
CPU	Central Processing Unit
CSM	Core Spreading Method
CUDA	Compute Unified Device Architecture
DNS	Direct Numerical Simulation
FD	Finite Difference
FFT	Fast Fourier Transform
FMM	Fast Multipole Method
FVM	Finite Volume Method
GPU	Graphics Processing Unit
KIFMM	Kernel Independent Fast Multipole Method
L2L	Local to local
L2P	Local to particle
LES	Large Eddy Simulation
M2L	Multipole to local
M2M	Multipole to multipole
P2M	Particle to multipole
P2P	Particle to particle
PDE	Partial Differential Equation
PSE	Particle Strength Exchange
RMSE	Root Mean Square Error
SFS	Sub-filter scale
SVD	Singular value decomposition
VPM	Vortex Particle Method

Greek symbols

$\boldsymbol{\Gamma}$	Particle strength (circulation)
$\boldsymbol{\omega}$	Vorticity
Δ	Filter width
δ	Step size

ϵ	Absolute error
ϵ_M	Machine epsilon
η	Relative error
$\hat{\Delta}$	Test filter width
ν	Kinematic viscosity
ν_t	Turbulent viscosity
ϕ	Kernel potential function
ρ	Density
σ	Smoothing radius
ζ	Smoothing function

Latin symbols

g	Regularisation function
q	Regularisation function
$\mathbf{u}$	Velocity
$\mathbf{x}$	Position
$\boldsymbol{a_k}$	Kernel Taylor coefficient
$\boldsymbol{e_i}$	Cartesian basis vector i
$\boldsymbol{l}$	Local expansion
$\boldsymbol{m}$	Moment
$\boldsymbol{S}_{ij}$	Strain rate tensor
$\boldsymbol{T_k}$	Potential function Taylor coefficient
$\mathbf{A}$	Angular impulse
$\mathbf{I}$	Linear impulse
$\mathbf{K}$	Kernel
$\mathcal{E}$	Enstrophy
$C_{\square}$	LES model constant
d	Octree depth
dim	Dimension
E	Kinetic energy
h	Particle spacing
N	Total number of particles
n_p	Number of particles in cluster
p	Pressure
p	Taylor series truncation order
R	Vortex radius
r	Radius
r_t	Vortex ring thickness
Re	Reynolds number

t	Time

Superscripts

$\square^*$	Normalised or non-dimensionalised
$\square^{\mathrm{c}_i}$	Of child cluster i
$\square^{\mathrm{p}}$	Of parent cluster
$\square^h$	Discretised
$\square^p$	Of arbitrary particle
$\square^s$	Of source particle
$\square^t$	Of target particle

Subscripts

$\square_\sigma$	Regularised

1

Introduction

Vortex methods are a set of Lagrangian grid-free methods of solving incompressible Navier-Stokes equations through the velocity-vorticity interaction. The early 1930s marked the advent of vortex methods with Rosenhead's work on Kelvin-Helmholtz instabilities, where he presented numerical calculations on the evolution of a vortex sheet, having discretised it into a system of point vortices [39].

In 1941, roughly a decade later, Hrennikoff published a paper to the ASME Journal of Applied Mechanics which exhibited an early form of mesh discretization [24]. Courant later used a mesh to approximate solutions to partial differential equations (PDEs) using basis functions [11]. Such work was further developed by academics and engineers in later years into what became known as the finite element method (FEM). This was followed by the development of other Eulerian methods, such as the finite volume method (FVM). Over the years, such Eulerian grid-based methods have become ubiquitous in computational fluid dynamics (CFD). Vortex methods, on the other hand, shifted to the background of scientific research and engineering application for much of this time, despite their earlier discovery.

Indeed, vortex methods suffered from many drawbacks to grid-based Eulerian methods that contributed to this shift in interest. Low numerical dissipation present in vortex methods, as a result of the absence of a grid, resulted in accumulation of local vorticity near the turbulent regime. Vortex methods were thus less stable than grid-based methods. Furthermore, vortex methods posed few satisfactory answers to the modelling of wall interactions [34] and were mostly limited to simulations in the incompressible flow regime.

The characteristic of vortex methods that posed possibly the largest hindrance to their applicability in modelling complex flows, was the $\mathcal{O}(N^2)$ complexity of acquiring the velocity from the vorticity field, caused by accounting for interactions between all possible pairs of vortex elements. Thus, as the simulation resolution increased to account for more complex flows, the use of vortex methods became computationally prohibitive.

Despite these drawbacks, vortex methods offer many advantages that were for a long time overshadowed by their implementation challenges. Their low numerical dissipation allows them to better conserve vortical structures. They reduce computational effort in regions of low vorticity, as their element concentration follows vortical structures naturally. Furthermore, their Lagrangian nature frees them from grid-based timestep stability constraints such as the Courant–Friedrichs–Lewy (CFL) condition.

Heirarchical algorithms developed in the late 20th century, such as the Barnes-Hut treecode [4] and the Fast Multipole method (FMM) [20] have contributed to bridging the computational gap between grid-based and vortex methods, by reducing computational complexity of the velocity field from $\mathcal{O}(N^2)$ to $\mathcal{O}(NlogN)$ or $\mathcal{O}(N)$. In addition, recent advances in scientific computing, particularly the use of GPUs for parallel computations, have allowed for significantly faster computations of these pairwise interactions. Other drawbacks have also been somewhat adressed. For example, recent large eddy simulation (LES) formulations [1][50] have been developed to reduce the instability of vortex methods.

The vortex particle method (VPM) is a category of vortex methods that discretizes the flow field into

vorticity carrying particles, developed in the 1970s beginning with the seminal work of Chorin [9]. As a result of these advancements, interest in VPM has been rekindled. FMM, proposed by Greengard and Rokhlin in 1987, improved the complexity of interactions to $\mathcal{O}(N)$ [20]. FMM is based on subdividing the computational domain into a hierarchical structure, usually octrees, and approximating distant interactions through multipole expansions. One significant variation of FMM formulates multipole expansions as Cartesian Taylor series, proposed by Zhao around the same time [57], referred to as Cartesian FMM.

The goal of this thesis is the implementation of one such algorithm for an existing VPM solver. The aim is to expand the current solver's capabilities into performing large scale computations where bypassing the $\mathcal{O}(N^2)$ computation of particle interactions becomes necessary. The implementation of such an algorithm would entail a characterisation of its effect on VPM accuracy. Furthermore, a characterisation of the computational performance of such an algorithm also follows. In summary, the following project objective is formulated:

Project Objective

The implementation and validation of a solver to reduce the complexity of $\mathcal{O}(N^2)$ particle interactions in VPM and the characterisation of its accuracy and computational performance.

In the pursuit of meeting this objective, a Cartesian FMM solver is developed for solving particle interactions in VPM, written in python, and designed to run on GPUs for improved parallelisation through the taichi language [25]. The work begins with an overview of VPM and a literature review on FMM in chapter 2. After the project research questions are formulated, details of the implementation are given in chapter 3. In chapter 4, an analysis of the errors introduced into VPM provides users of the solver with benchmarks of FMM errors for different flow scenarios. The solver is validated in chapter 5 using three test cases, providing an insight into the evolution of FMM error during transient simulations. The computational performance of the solver is also assessed in chapter 6, allowing users to understand where performance can be traded off with accuracy and highlighting avenues for potential future optimisations.

2

Background and Literature Review

This chapter presents the relevant theoretical background and a summary of the reviewed literature for the thesis. It begins with an overview of VPM as well as its different LES formulations in section 2.1. Variations of treecodes and FMM are introduced in section 2.2. Next, a survey of available FMM libraries is presented. Based on this investigation, research questions are synthesized in section 2.3, guiding the research activities undertaken in this thesis.

2.1. Vortex Particle Method

VPM is a class of vortex methods designed for simulation of incompressible, viscous flows. It is a fully Lagrangian method where the vorticity field is discretised into vorticity-carrying particles. The particles convect with the fluid and their velocities are calculated through the velocity-vorticity equation. This Lagrangian nature of VPM allows it to be completely grid-free. In VPM, each particle carries a vortex strength (i.e. amount of circulation). This is defined through multiplication of particles vorticity by their volume, which is not a physical volume but a discretisation parameter. The interaction between particles updates their positions, strengths and/or volumes with each time step.

In other vortex methods, such as the vortex filament method, elements often track continuous vortex lines and thus are constrained by Helmholtz's first theorem, limiting how their vorticity can evolve. VPM avoids this by treating particles as independent in this sense, allowing their vorticity to change. This feature of VPM allows for modelling of viscous diffusion, allowing the strength of particles to be modified by viscous effects, an essential feature in CFD simulations.

2.1.1. Formulation of the Vortex Particle Method

The Vorticity Transport Equation

Under the assumption of constant viscosity, the Navier-Stokes momentum equation can be expressed as:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} \ , \tag{2.1}$$

Where $\mathbf{u}$ is the velocity field, ρ is the fluid density, p is the scalar pressure field, and ν is the fluid kinematic viscosity. Furthermore, with the assumption of incompressibility, the continuity equation is:

$$\nabla \cdot \mathbf{u} = 0 \ . \tag{2.2}$$

Taking the curl of Equation 2.1 results in:

$$\frac{D\boldsymbol{\omega}}{Dt} = \underbrace{(\boldsymbol{\omega}\cdot\nabla)\,\mathbf{u}}_{\text{Vortex Stretching}} + \underbrace{\nu\nabla^2\boldsymbol{\omega}}_{\text{Viscous Diffusion}} \quad , \tag{2.3}$$

Where $\nabla \times \mathbf{u} = \boldsymbol{\omega}$ is the vorticity and $D\mathbf{f}/Dt = \partial\mathbf{f}/\partial t + (\mathbf{u}\cdot\nabla)\mathbf{f}$ is the material derivative of an arbitrary vector function $\mathbf{f}$. As the pressure term disappears in this equation, it is Lagrangian invariant by construction, making it suitable for VPM. Equation 2.3 is referred to as the vorticity transport equation.

The VPM Formulation

In VPM, the vorticity field is discretized unto either singular vortex particles or regularized (vortex blob) particles [51]. Singular vortex particles consist of a point source with an associated vortex strength. The real vorticity field $\boldsymbol{\omega}$ is approximated by the discretised field $\boldsymbol{\omega}^h$ using:

$$\boldsymbol{\omega}(\mathbf{x},t) \approx \boldsymbol{\omega}^h(\mathbf{x},t) = \sum_p \boldsymbol{\Gamma}^p(t)\,\delta(\mathbf{x}-\mathbf{x}^p(t)) \ , \tag{2.4}$$

Where p stands for a specific particle in the flow field and δ is the 3D Dirac-delta function. The particle velocities are obtained from their vorticities using the Biot-Savart law:

$$\mathbf{u}(\mathbf{x},t) = \sum_p \mathbf{K}(\mathbf{x}-\mathbf{x}^p(t)) \times \boldsymbol{\Gamma}^p(t) \ , \tag{2.5}$$

Where Γ^p is the particle strength and $\mathbf{K}(\mathbf{x})\times$ is the Biot-Savart kernel, expressed as:

$$K\left(\mathbf{x}\right)\times = -\frac{1}{4\pi}\frac{\mathbf{x}}{|\mathbf{x}|^3}\times \quad . \tag{2.6}$$

Where $\times$ denotes the cross product operator. Singular vortex particles exhibit a singularity at their centre, where the vorticity becomes unbounded as $r = \|\mathbf{x}-\mathbf{x}^p(t)\| \to 0$. Regularised VPM avoids this by distributing the vorticity over a finite region of space, resulting in a smooth vorticity distribution about the particle centre. This is defined using the smoothing function ζ_σ and a characteristic *cut-off radius* σ. The regularised vorticity field is consequently expressed as:

$$\boldsymbol{\omega}(\mathbf{x},t) \approx \boldsymbol{\omega}^h_\sigma(\mathbf{x},t) = \sum_p \boldsymbol{\Gamma}^p(t)\,\zeta_\sigma(\mathbf{x}-\mathbf{x}^p(t)) \ . \tag{2.7}$$

The particle velocities are still obtained using the Biot-Savart law. However, the kernel $\mathbf{K}_\sigma(\mathbf{x})\times$ used in each case is dependent on the smoothing function implemented. The induced velocity in a position $\mathbf{x}$ due to the regularised vorticity field is expressed as:

$$\mathbf{u}_\sigma(\mathbf{x},t) = \sum_p \mathbf{K}_\sigma(\mathbf{x}-\mathbf{x}^p(t)) \times \boldsymbol{\Gamma}^p(t) \ . \tag{2.8}$$

From the previous equations, the evolution equations for VPM can be derived. Equation 2.9 updates the positions of the particles based on their velocities, and Equation 2.10 updates their strengths based on the vortex stretching term in Equation 2.3. The *classical scheme* is formulated as:

$$\frac{d}{dt}\mathbf{x}^p(t) = \mathbf{u}^p_\sigma(t) \ , \tag{2.9}$$

$$\frac{d}{dt}\boldsymbol{\Gamma}^p(t) = (\boldsymbol{\Gamma}^p(t)\cdot\nabla)\,\mathbf{u}^p_\sigma(t) \ . \tag{2.10}$$

Winckelmans and Leonard replaced the Laplacian operator in Equation 2.10 with its transpose, to obtain the *transpose scheme*, which results in an approximation of vorticity that is closer to the real vorticity than the classical scheme. Contrary to the classical scheme, the transpose scheme was found to conserve total vorticity in the flow [51]. The strength evolution equation for the transpose scheme is expressed as:

$$\frac{d}{dt}\boldsymbol{\Gamma}^p(t) = \left(\boldsymbol{\Gamma}^p(t)\cdot\nabla^T\right)\mathbf{u}_\sigma^p(t)\ . \tag{2.11}$$

The classical and transpose scheme can also be combined into the *mixed scheme*:

$$\frac{d}{dt}\boldsymbol{\Gamma}^p(t) = \frac{1}{2}\left(\boldsymbol{\Gamma}^p(t)\cdot(\nabla+\nabla^T)\right)\mathbf{u}_\sigma^p(t)\ . \tag{2.12}$$

The mixed scheme allows one to use the particle strain-rate tensor S_{ij}^p to update particle strengths, given that:

$$\boldsymbol{S}_{ij}^p = \frac{1}{2}(\nabla+\nabla^T)\mathbf{u}_\sigma^p(t)\ . \tag{2.13}$$

The strength update equation then becomes:

$$\frac{d}{dt}\boldsymbol{\Gamma}^p(t) = \boldsymbol{S}_{ij}^p\cdot\boldsymbol{\Gamma}^p\ . \tag{2.14}$$

For simplicity, the discretised, regularised vorticity field, $\boldsymbol{\omega}_\sigma^h$ is expressed as $\boldsymbol{\omega}$ and the resultant velocities $\mathbf{u}_\sigma$ is expressed as $\mathbf{u}$ for the remainder of this thesis.

Viscous diffusion models

The viscous diffusion term in Equation 2.3 has proven to be non straightforward to resolve in VPM, which has resulted in a variety of methods used in literature that attempt to resolve it. They can be classified into stochastic and deterministic methods. A widely used stochastic method is the random walk method introduced by Chorin [9] which updates the positions of the particles by emulating random Brownian motion. Deterministic methods include Particle Strength Exchange (PSE) [15][16] in which strength is redistributed between nearby particles to emulate the effect of diffusion and the core spreading method (CSM) proposed by Leonard [29] which changes the core size of the vortex blobs to adjust for viscous diffusion.

Smoothing functions

Several smoothing functions can be used for the regularized VPM. The most commonly encountered in literature are the *Gaussian*, *low-order algebraic* proposed by Rosenhead and Moore [40][35] and the *high-order algebraic* function proposed by Winckelmans and Leonard [51], or the WL kernel for short. The formulae for each of these smoothing functions and their corresponding kernel expressions are presented in Table 2.1, with $\mathbf{x}^t$ as a target particle position, $\mathbf{x}^s$ a source particle position, $\mathbf{r} = \mathbf{x}^t - \mathbf{x}^s$, $r = \|\mathbf{x}^t - \mathbf{x}^s\|$ and $\rho = r/\sigma$.

Table 2.1: Smoothing functions and their kernels [51]

Smoothing Function	**Formula**	**Kernel**
Gaussian	$\zeta_\sigma(\rho) = \frac{1}{(2\pi)^{3/2}}e^{-\rho^2/2}$	$\mathbf{K}_\sigma(r) = \frac{1}{4\pi r^3}\left[\mathrm{erf}\left(\frac{\rho}{\sqrt{2}}\right) - \sqrt{\frac{2}{\pi}}\,\rho e^{-\frac{\rho^2}{2}}\right]\mathbf{r}$
Low-Order Algebraic	$\zeta_\sigma(\rho) = \frac{3}{4\pi(1+\rho^2)^{5/2}}$	$\mathbf{K}_\sigma(r) = -\frac{1}{4\pi}\frac{\mathbf{r}}{(r^2+\sigma^2)^{3/2}}$
High-Order Algebraic	$\zeta_\sigma(\rho) = \frac{15}{8\pi(1+\rho^2)^{7/2}}$	$\mathbf{K}_\sigma(r) = -\frac{1}{4\pi}\frac{(r^2+\frac{5}{2}\sigma^2)}{(r^2+\sigma^2)^{5/2}}\mathbf{r}$

The Gaussian smoothing function was proven to be convergent to the true vorticity equation when the number of particles increases and the cut-off radius decreases. It is a common choice in regularized VPM for this reason. The low-order algebraic smoothing is more numerically convenient as it does not involve transcendental functions, however Winckelmans and Leonard showed that it might not be convergent for all cases as the number of particles is increased. The high-order algebraic smoothing they proposed was shown to have equivalent convergence properties to Gaussian smoothing, while offering the advantage over Gaussian smoothing of providing closed form solutions for the quadratic diagnostics (kinetic energy, helicity and enstrophy) [51].

Regularization functions for the high-order algebraic kernel

In addition to the smoothing function $\zeta_\sigma(\rho)$, other functions specific to each kernel are defined by Winckelmans and Leonard which can be used to derive other flow quantities. The $q(\rho)$ and $g(\rho)$ functions for the high-order algebraic kernel are reported here as:

$$q(\rho) = \frac{\rho^3(\rho^2 + \frac{5}{2})}{4\pi(\rho^2+1)^{5/2}} \ , \tag{2.15}$$

and:

$$g(\rho) = \frac{\rho^2 + \frac{3}{2}}{4\pi(\rho^2+1)^{3/2}} \ . \tag{2.16}$$

VPM Diagnostics

Quantities of the flow that are measured to ensure the physical validity of the VPM simulations performed are referred to as flow diagnostics. These could be quantities that must be conserved by the flow or that must remain zero throughout the simulation. In the case of singular particles, these quantities are conserved. The regularization of particles introduces an approximation which results in diagnostics no longer being conserved. It must be ensured that they do not significantly diverge from their original values. VPM flow diagnostics are obtained from Winckelmans and Leonard [51] and are summarised in this section. For the definitions in this section an average of the cutoff radii for an arbitrary pair of particles i and j is given as σ_{ij} such that:

$$\sigma_{ij} = \frac{1}{2}(\sigma_i + \sigma_j) \ ,$$

consequently:

$$\rho = \frac{r}{\sigma_{ij}} \ .$$

Enstrophy

Enstrophy provides a measure of the flow dissipation. For incompressible flows, the enstrophy can be expressed as:

$$\mathcal{E} = \int \omega \cdot \omega \, \mathbf{dx} \ . \tag{2.17}$$

The enstrophy is a positive quantity, since the dissipation of energy from larger to smaller scales is always positive. While there is a small component energy transfer from smaller scales into larger scales, termed backscatter, this is greatly overshadowed by the transfer from large scales into small scales. The formulation of enstrophy in a regularised context is given as:

$$\mathcal{E}_\sigma = \sum_i^N \sum_j^N \frac{\zeta(\rho)}{\sigma_{ij}^3} (\mathbf{\Gamma}^i \cdot \mathbf{\Gamma}^j) \ . \tag{2.18}$$

Kinetic Energy

The kinetic energy is calculated through:

$$E = \frac{1}{2}\int \mathbf{u}\cdot\mathbf{u}\,\mathbf{dx}\ . \tag{2.19}$$

With a regularised method, this becomes:

$$E_\sigma = \frac{1}{2}\sum_i^N\sum_j^N \frac{g(\rho)}{\sigma_{ij}}\left(\mathbf{\Gamma}^i\cdot\mathbf{\Gamma}^j\right)\ . \tag{2.20}$$

For a simulation where all scales of turbulence are resolved or modelled, the rate of decay of kinetic energy is related to the enstrophy through the viscosity:

$$\frac{dE}{dt} = -\nu\mathcal{E}\ . \tag{2.21}$$

For inviscid simulations, the value of E_σ should remain conserved as long as the regularised vorticity field remains a close representation of the real vorticity field. In viscous simulations, E_σ is constantly decreasing through the effect of energy dissipation in the Kolmogorov length scale. For an under-resolved simulation, there is no mechanism for the dissipation of such small turbulence scales, and thus the kinetic energy decay rate seen in such simulations is lower than in the relation in Equation 2.21. In this way, Equation 2.21 is an indicator of whether turbulence is being sufficiently resolved or modelled in the simulation.

In this work, the kinetic energy decay is estimated using a third order backward difference scheme, which is given by:

$$\frac{dE}{dt} \approx \frac{11E_t - 18E_{t-\Delta t} + 9E_{t-2\Delta t} - 2E_{t-3\Delta t}}{6\,\Delta t}\ , \tag{2.22}$$

However, other numerical schemes can also be used.

Total vorticity, Linear Impulse and Angular Impulse

In viscous unbounded flows, provided that the flow vorticity at infinity is zero, the total vorticity (Ω) remains zero, while the linear ($\mathbf{I}$) and angular ($\mathbf{A}$) impulse are conserved [51]. The total vorticity is obtained from summing the all particle strengths:

$$\mathbf{\Gamma}_{\text{tot}} = \sum_i^N \mathbf{\Gamma}^i\ , \tag{2.23}$$

$$\Omega = \|\mathbf{\Gamma}_{\text{tot}}\|\ . \tag{2.24}$$

The linear and angular impulse are calculated by:

$$\mathbf{I} = \frac{1}{2}\sum_i^N \mathbf{x}^i\times\mathbf{\Gamma}^i\ , \tag{2.25}$$

and:

$$\mathbf{A} = \frac{1}{3}\sum_i^N \mathbf{x}^i\times\left(\mathbf{x}^i\times\mathbf{\Gamma}^i\right)\ . \tag{2.26}$$

respectively.

The need for fast N-body solvers

Equation 2.8 and Equation 2.9 present a major challenge in implementing VPM using current computational capabilities. In order to update the positions of particles, the induced velocity on every particle in the domain must be computed. The calculation of the induced velocities on the particles on each other requires one computation for each unique pair of particles, resulting in a problem with $\mathcal{O}(N^2)$ complexity. This method of particle interaction computations is henceforth referred to as the *direct method*. Due to the high complexity of the direct method, the need for more efficient methods to accelerate these computations arises when N is large. This paves the way for the use of fast N-body solvers, methods which reduce the complexity of these pairwise interactions. Among those are treecodes and FMM, which will be discussed in more detail in section 2.2.

2.1.2. Large Eddy Simulations with VPM

Resolving the full range of turbulence scales down to the Kolmogorov scale is a challenge in Eulerian grid-based methods, with the computational complexity reaching $\mathcal{O}(Re^3)$ for DNS computations due to the need for increasingly fine meshes to capture smaller scales [38]. Therefore, LES simulations are usually performed, where the largest scales of turbulence are resolved, while scales below a cutoff filter determined by the grid spacing, are modelled.

Alvarez cites the vortex stretching term in Equation 2.3 as a cause for the instability of VPM near the turbulent regime [1]. The instability is caused by the rapid increase in local vorticity in that region, which is not damped due to the low numerical dissipation of VPM. The presence of turbulent diffusion can help alleviate this instability, which can be achieved through the formulation of an LES scheme for VPM which models the regions of higher turbulence rather than resolving them directly.

VPM methods, in contrast to Eulerian methods, are meshless. The filter width cannot be determined by a grid spacing. As a consequence, several proposed formulations of LES models tailored to VPM exist in literature. Winckelmans investigated possible LES models for VPM. The models included a variant of the Smagorinsky model, replacing the Eulerian grid filter h that normally defines the LES filter size, with the particle cut-off radius σ [50]:

$$\nu_\mathrm{t} = (C_s\,\sigma)^2\,\|\boldsymbol{S}_{ij}\|, \tag{2.27}$$

Where $\|\boldsymbol{S}_{ij}\| = \sqrt{2\boldsymbol{S}_{ij}\boldsymbol{S}_{ij}}$ is the Frobenius norm of the strain rate tensor scaled to account for symmetry.

Possible models also included a similar model based on local enstrophy proposed by Mansour, Ferziger, and Reynolds [31] :

$$\nu_\mathrm{t} = (C_v\,h)^2\,\sqrt{(\boldsymbol{\omega}_i\boldsymbol{\omega}_i)}\ , \tag{2.28}$$

and a model based on the relative rate of change of local enstrophy due to vortex stretching:

$$\nu_\mathrm{t} = (C_w\,h)^2\ 2\frac{\boldsymbol{\omega}_i\boldsymbol{S}_{ij}\boldsymbol{\omega}_j}{\boldsymbol{\omega}_i\boldsymbol{\omega}_i}\ . \tag{2.29}$$

They also highlighted the potential for a dynamic Smagorinsky model to be used with VPM.

2.1.3. OpenONDA

The VPM solver considered in this thesis is OpenONDA [32], an open source FVM and VPM solver, developed by researchers in the Delft University of Technology and released under the GNU General Public License. As of the time of this writing, the VPM solver can be run with an LES or DNS formulation of VPM, with treatment of the viscous diffusion using a choice of either the PSE or CSM, random walk or a hybrid of the CSM and random walk methods. The code uses the Gaussian, super Gaussian and high-order algebraic kernel due to its convergence properties and to facilitate the computation of quadratic diagnostics. The focus of this thesis is solely on the high-order algebraic kernel. The VPM scheme in OpenONDA is written in python, with computationally heavy operations written

in the compiled taichi language [25] which is used to compile computations to be run in parallel on the GPU or the CPU. As of the time of this writing, the LES options in OpenONDA are the classical Smagorinsky model which is modified for compatibility with VPM, and sub-filter scale (SFS) stretching model inspired by that of Alvarez [1]. The working methods of these schemes are summarized here to indicate whether additional considerations would need to be integrated to the implemented method.

The Smagorinsky model uses a fixed value of the Smagorinsky constant C_s. In the OpenONDA Smagorinsky model, the filter width in the classical Eulerian LES is substituted with a filter width computed from the particle volumes $\Delta^p = (vol^p)^{1/3}$ resulting in:

$$\nu_t = (C_s \, \Delta^p)^2 \, \|\boldsymbol{S}_{ij}\| \ , \tag{2.30}$$

The other LES option in OpenONDA, which modifies the SFS vortex stretching term rather than the effective particle viscosity, uses the strain rate tensor $\boldsymbol{S}_{ij}$ to modify particle strengths $\boldsymbol{\Gamma}^i$ through:

$$\tau_{\text{stretch}} = -C_{\text{stretch}} \ \Delta^2 \ \|\boldsymbol{S}_{ij}\| \ (\boldsymbol{S}_{ij} \cdot \boldsymbol{\omega}) \ vol \ , \tag{2.31}$$

which is used to update particle strengths with each timestep as:

$$\boldsymbol{\Gamma}_{i+1}(t) = \boldsymbol{\Gamma}_i(t) + \tau_{\text{stretch}} \cdot dt \ . \tag{2.32}$$

2.2. Treecodes and the Fast Multipole Method

2.2.1. Hierarchical Treecodes

Treecode algorithms accelerate the computation of pairwise interactions in N-body problems, such as the Poisson equation in vortex particle methods. This is done by recursively dividing areas of the computational domain where particles are concentrated and treating the particles in these areas as clusters. The division results in hierarchical structures labelled as quadtrees (2D) or octrees (3D).

The idea behind treecodes is to avoid calculating direct particle-particle inductions for the entire domain by expanding the effect of the particles on their cluster, and using that to find the induction on other clusters (cluster-cluster) or on particles where the induction is to be calculated (cluster-particle). Direct particle-particle interactions are only calculated when the distance between the particles is considered small.

An early implementation of particle clustering was presented by Spalart and Leonard [43]. Although they did not implement a treecode, Spalart and Leonard proposed clustering groups of particles and computing long-distance interactions from cluster to cluster, rather than each point individually, which reduced the computational effort to $\mathcal{O}(N^{3/2})$.

Appel [2] proposed the first treecode algorithm to speed up N-body gravitational simulations in astrophysics, claiming to reach a complexity of $\mathcal{O}(NlogN)$. The algorithm approximated the gravitational force from a far cluster of bodies by the equivalent gravitational force induced by their collective centre of mass, and divided the computational space using a binary tree.

Barnes and Hut [4] improved on this, replacing the binary tree with an octree which houses no more than one particle in each cluster, rebuilding the tree at each time step. This approach avoided tangling experienced by [2], which introduced errors. They also proved the $\mathcal{O}(NlogN)$ complexity of their method [4][37].

2.2.2. Fast Multipole Method

The fast multipole method was first proposed by Greengard and Rokhlin as a method for speeding up numerical simulation of N-body problems, from $\mathcal{O}(N^2)$ up to $\mathcal{O}(N)$, significantly faster than the previously mentioned treecodes [20]. FMM was proposed as a solution for problems with gravitational

and Coulomb interactions, and the authors recognised its suitability in solving the induced velocities through the Poisson equation in the vortex method.

FMM also allows for an arbitrary accuracy of computation, up to machine precision [57], making it more suitable for the levels of accuracy needed for fluid dynamics simulations. This accuracy is achieved by performing multipole expansions of the particles at the lowest cell level, where the multipole expansion truncation error is selected to the level of accuracy required [41]. The additional contribution of local expansions allows the algorithm to take advantage of cluster-cluster interactions (i.e. calculate the effect of a cluster on another cluster, rather than the effect of the cluster on a single particle), which further reduces the complexity from $\mathcal{O}(NlogN)$ to $\mathcal{O}(N)$.

The method of multipole expansion using spherical harmonics, later proposed by Greengard and Rokhlin allowed for performing 3D FMM, detailed in [21] and [17]. Due to its use of spherical harmonics, this FMM formulation is only suitable for harmonic kernels such as the Gaussian kernel, but cannot be used for non-harmonic kernels, such as the high order algebraic kernel proposed by Winckelmans and Leonard [41]. An alternative method of multipole expansion was developed around the same time by Zhao, where they used Cartesian rather than spherical coordinates, and expressed the expansions in terms of Taylor series, expanding the algorithm's applicability to non-harmonic kernels [41].

The formulation in [21] also addressed another bottleneck in the original FMM formulation in [20], which had complexity with respect to the degree of multipole expansion p of $\mathcal{O}(p^4)$ in 3D, reducing the complexity to $\mathcal{O}(p^2logp)$. In general, the use of a larger degree of multipole expansions p is desired because it increases the accuracy of these expansions, and leads to a lower required number of pairwise interactions, which are costly [10]. Because of this, the improvement to p complexity in [21] was instrumental to FMM efficiency for 3D N-body problems.

A kernel-independent formulation of the FMM algorithm (KIFMM) was later introduced by Ying, Biros, and Zorin [52] replacing analytical multipole expansions of the kernel with equivalent density computations which are obtained by solving local exterior and interior problems on surfaces. The only requirement for the kernel is to be associated with a second-order non-oscillatory elliptic PDE.

Fong and Darve [18] further extended the FMM capabilities to any numerically defined kernel with the Black-Box FMM (BBFMM) which uses Chebyshev polynomials as an interpolation basis for computing low-rank approximations of the kernel function, incorporating this with a FMM tree structure. The authors cite BBFMM as a useful method for more complicated analytically defined kernels.

Recent developments in treecodes and FMM include parallelization of the algorithms over CPU or GPU architectures. An early method to parallelize treecodes was introduced by Warren and Salmon [48] making use of Morton (Z-order) for efficiently locating octree cells, a widely used method in most FMM libraries.

FMMs structured hierarchical ordering and tree traversal make it a suitable algorithm for parallelization on CPU or GPU architectures. Upward (combining child cluster effects on their parent cluster) and downward passes (parent to child) in the FMM algorithm are particularly suitable components for parallelization.

Hierarchical algorithms were initially more efficient to run on CPUs than GPUs. Hamada et al. presented a 20.2 TFlops vortex particle simulation of homogenous isotropic turbulence (HIT) using reformulated FMM suited to run on the GPU [22], highlighting the large scale potential of parallelised FMM, and outperforming CPU efficiency. Yokota et al. showed 74% parallel efficiency of FMM compared to 14% for Fast Fourier Transform (FFT) based methods [54].

2.2.3. Available FMM Repositories

In this section, a survey is performed of the various open-source FMM libraries that are made available online. The focus is on their main capabilities, their complexity of implementation and interfaces for integration with existing solvers. This survey will help inform the most relevant libraries to support the development of an in-house FMM solver.

pyFMM
Cruz and Barba [13] developed a 2D FMM code with the application of VPM with a Gaussian smoothing function in mind. The code is developed in python. Although the Laurent series formulation used in 2D is not suitable for 3D FMM, the code could serve as an example of how FMM can be tailored for VPM applications as well as giving a simpler view of FMM programming than other more complicated 3D codes.

ScalFMM
Developed by Blanchard et al. from Inria [7], ScalFMM is a highly customizable FMM code with a wide array of potential applications. The code allows for multiple different formulations of multipole and local expansions, including Chebyshev, barycentric Lagrange [46] and uniformly spaced interpolation. The code however might be considered too complicated for an already predefined application, thus it might not be as user-friendly as other codes.

PVFMM
PVFMM [30] developed by Malhotra and Biros is a high-order, adaptive and scalable parallel FMM library written in C/C++. It is tailored towards solving PDEs but can be equivalently used for N-body problems. It is based on the kernel-independent FMM formulation. It uses piecewise Chebyshev polynomials which allows to compute the resulting potential and its derivatives. The library also includes a python binding, which allows it to be integrated with a code like OpenONDA.

BBFMM3D
Using the Black-Box FMM method, the authors created the BBFMM3D library [18]. It is written in C++ and is the only library in the ones reviewed that uses the black-box method. It however, only contains bindings for MATLAB, so is not easily implementable with OpenONDA. Also, given that the VPM formulation in OpenONDA uses a symmetric analytically defined kernel, there is no need for implementing the black-box method.

ExaFMM-t
ExaFMM [47] is a parallel exa-scale FMM code developed by the Barba Group. It is written in C++ and uses the kernel-independent formulation of FMM [52]. The library is GPU enabled using CUDA. The library is written with conciseness, reusability and maintainability in mind. As a result, the authors claim the code to be orders of magnitude shorter than most other FMM libraries, with only around 6000 lines of code. The library also includes python bindings using the `pybind11` library, which can be used for coupling with OpenONDA.

PyExaFMM
PyExaFMM [26] was also developed by the Barba Group as a python implementation of the pyExaFMM library. The library makes extensive use of `numba` [28], a python library which allows for compiling python code for high-performance applications. Since this function is written in python, it can be very useful for the development of a similar code or for direct integration with OpenONDA.

DUST
While not a standalone FMM library. The aerodynamic solver DUST [45] contains a Cartesian formulation of FMM written in FORTRAN. The mathematical formulations applied in this code can be useful in developing a Cartesian FMM solver.

2.3. Project Objective and Research Questions

The literature review performed on the possible methods of speeding up the $\mathcal{O}(N^2)$ particle interactions in VPM highlighted many different FMM formulations which can be implemented. In order to select a suitable formulation to be implemented for this thesis, a comparison of the main benefits and drawbacks of each formulation is summarised in Table 2.2.

Table 2.2: Comparison of different FMM formulations

Formulation	Benefits	Drawbacks
Cartesian	• Simple to implement • Confirmed suitability with high-order algebraic kernel [41]	• Large amount of summations ($\mathcal{O}(p^6)$ [42])
Spherical Harmonic	• Fewer summations ($\mathcal{O}(p^4)$ or $\mathcal{O}(p^2 logp)$ [42])	• Not immediately suitable for high-order algebraic kernel [41] • Computationally expensive trigonometric calculation of spherical harmonic functions [56]
BBFMM	• Widest scope of possible kernels (non-oscillatory numerical or analytical kernels) [18] • Minimal number of coefficients due to use of singular-value decomposition (SVD) [18]	• Complex to implement
KIFMM	• Wide scope of possible kernels (second-order non-oscillatory elliptic [52])	• Complex to implement • Slower than Cartesian FMM [23]

Based on this comparison, a Cartesian FMM formulation was selected to be implemented due to its relative simplicity of implementation and suitability of use with the high-order algebraic kernel. The literature review also highlighted the widespread use of GPUs in numerical computations and algorithms such as FMM. Thus, in order for the FMM implementation to remain performant compared to the direct method, which would be parallelised on the GPU, the FMM solver should also attempt to exploit taichi's abilities in performing fast parallel computations on the GPU. To fill the knowledge gaps necessary for achieving this objective, a few research questions follow. Each question is followed by a justification for its investigation.

Research Question 1

How does the use of the Cartesian FMM solver affect the accuracy and speed of obtaining induced velocities compared to direct computation?

Authors including Alvarez [1], Berdowski [6] mention the use of FMM for their LES formulation of the VPM solver. However, they use a Gaussian kernel, exploiting the spherical harmonic formulation of the FMM expansions, which is unsuitable for the high-order algebraic kernel used in OpenONDA [41]. Stock, Gharakhani, and Stone also mentions using FMM to solve their VPM-LES formulation, but there are no details of the implementation [44]. Cruz and Barba [12] provide an in depth quantification of FMM error for a 2D Laurent series FMM. On the contrary, there has been no literature found that thoroughly quantify the errors of using a 3D Cartesian FMM for VPM. Furthermore, the use of single float precision in computations, as is customary on GPUs, might pose additional considerations for the accuracy. Answering these question will help to determine the level of resolution required for the FMM to introduce an acceptable level of error into the VPM solution while maintaining fast summation.

Research Question 1A

How does a change in FMM parameters impact the accuracy of Cartesian FMM when applied to VPM?

Implementing an accurate FMM simulation relies on a selection of FMM parameters, mainly the truncation order p and the octree depth d. It is useful to implement a parametric analysis on the accuracy of FMM for different flows to investigate its sensitivity to these changes. The goal of this research question is to collect sufficient data points on this to inform user's selection of FMM settings depending on the case they are investigating.

Research Question 1B

How do the different operations of FMM implementation scale with a change in FMM parameters and number of particles?

The other aspect that users will need to decide on the FMM settings to use, are the performance gains from FMM. For this, a profiling of the computational time and complexity for the FMM operations is needed. Users can use this to gain insight on whether FMM should be used, and what settings to use to improve the VPM computational performance sufficiently without significantly affecting accuracy of results.

Research Question 2

Can a central-differencing scheme be reliably incorporated with Cartesian FMM to compute the velocity gradient tensor?

In a 3D VPM it is also required to compute the velocity stretching term in Equation 2.3. This remains an $\mathcal{O}(N^2)$ problem if it is computed independently from the induced velocities computed with FMM. To maintain the advantage of $\mathcal{O}(N)$ complexity, the velocity gradient computations required for the stretching term must also be integrated into the FMM solver. The accuracy of the velocity gradient tensor is also required for the formulation of the strain-rate tensor, which plays a key role in LES modelling, for example through the Smagorinsky model, as seen in Equation 2.27.

Berdowski et al. [5] implemented a CDS scheme for a spherical harmonic $\mathcal{O}(NlogN)$ FMM, where both the velocities and their gradients are obtained using CDS, and analysed the errors resulting thereof. This was performed for double-precision computations, where floating point errors are insignificant. It is unclear whether such method can be reliably used for single-precision computations, where floating point errors are more significant, and whether that would introduce significant error to the evolution of VPM simulations.

3

Fast Multipole Method

This chapter lays out the relevant theory and methodology of the FMM code written for improving the computational complexity of VPM simulations. The code is written in python, with the use of the numpy library to leverage its speed in performing array computations. The Taichi compiled language [25] is used for performing the computationally expensive parts of the algorithm. The underlying mathematics of the multipole and local expansions are the same as those used by Sabatino [41] and are also implemented in the FMM code included in the DUST aerodynamic solver [45]. The main data structure involved in the FMM method, octrees, are described in section 3.1. An overview of the FMM algorithm is presented in section 3.2. The mathematical operations encountered in FMM traversing the octree are then presented in section 3.3. These operations are the building blocks implemented to form the FMM algorithm. Finally, details of performance considerations for the code are outlined in section 3.4.

3.1. Octree hierarchical structures

Central to the functioning of FMM is the hierarchical structure used to contain the particles in the domain. FMM, when performed in 3D, uses the octree hierarchical structure. An octree is a tree data structure, where a 3D space is recursively subdivided into eight smaller cubes, referred to as clusters. In 2D, the hierarchical structure is referred to as a quadtree, where the domain is recursively subdivided into four smaller squares.

The definitions of the operations performed in FMM are intertwined with the octree itself. It is thus essential to sufficiently define the octree to understand the functioning of FMM. Below are some key definitions commonly used for octrees which will be used to describe their working methodology in the subsequent theory.

Domain The physical region encompassing all particles describing the flow field at a specific timestep.

Cluster Subdivisions of the computational domain that encompass the particles within them.

Parent A cluster which is subdivided into several smaller child clusters

Child A subdivision of a parent cluster.

Octree The full hierarchy of parent and child clusters that exists within the computational domain.

Level The number of subdivisions required to reach the cluster (the full domain is level 0, the largest parent clusters are at level 1, etc.).

Depth The maximum level present in the octree.

Leaf clusters The set of clusters in the octree which do not have children clusters (i.e. are not further subdivided).

Full octree An octree in which every parent cluster is divided into exactly eight children clusters and all clusters contain children down to the same leaf level.

Neighbour cluster A cluster which is of the same level and shares a face, edge or vertex with the current cluster (See Figure 3.1).

Interaction list A list of clusters which are children of neighbour clusters of the current cluster's parent, but which are not themselves neighbours of the current cluster (See Figure 3.1).

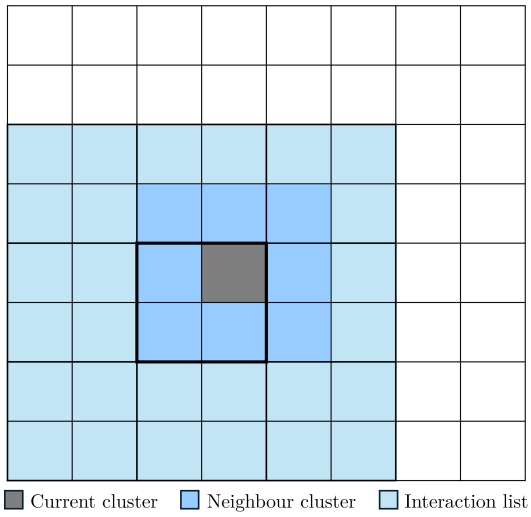


Figure 3.1: Illustration of the neighbouring clusters and interaction list on a quadtree (2D) with a depth of 3. This can also be generalised to three dimensions for octrees.

In octrees, the entire domain is recursively subdivided into children clusters depending on a predetermined criterion. Subdivision could be performed until a predetermined depth, denoted d, is reached, or for a maximum number of particles in each cluster, denoted n_p. It is nonetheless important to set a limit on the maximum depth that can be reached by the octree, as octrees with larger depth can rapidly become more memory intensive, with memory complexity of $\mathcal{O}(8^d)$.

In this implementation, a full octree structure is always generated. A benefit of this approach is that one is assured of the presence of all leaf clusters at the finest level, and thus the operations can be performed without traversing the tree to find the leaves. The other benefit is that all leaf clusters are of the same level, so the neighbour clusters and interaction lists are unambiguously defined. This comes at the cost of an increase in unpopulated leaves when building the octree, which add to the memory requirements of the method, limiting the effective depth that can be reached by FMM. Furthermore, this method results in clusters with a significantly lower number of particles than the threshold n_p. In an alternative method where clusters are selectively subdivided only if they contain more particles than n_p, there are less clusters in the resulting octree because clusters contain a number of particles closer to n_p. This can greatly reduce the number of FMM operations performed but increases the intricacy of the FMM implementation.

3.1.1. Morton Codes

Morton codes [36], also known as Z-order codes, are a method of encoding cluster coordinates in hierarchical quadtree and octree structures which preserves spatial locality. Morton codes which lie close in value to each other correspond to clusters which are often spatially close to each other.

In an octree, the Morton code of a cluster describes its relative location in the domain. Every three bits represent the cluster's location at a level of the octree. In an octree, for example, a cluster with a 12 bit Morton code is located at the 4th level of the octree. Each of the bits in the three-bit segment represents the relative position in a certain Cartesian direction, with the first value being 0 if in the more negative e_i coordinate, and 1 for the subdivision in the more positive e_i coordinate. This can be seen more clearly in Figure 3.2.

To obtain the Morton code of a cluster's parent cluster, the rightmost three bits of the child cluster are

simply omitted.

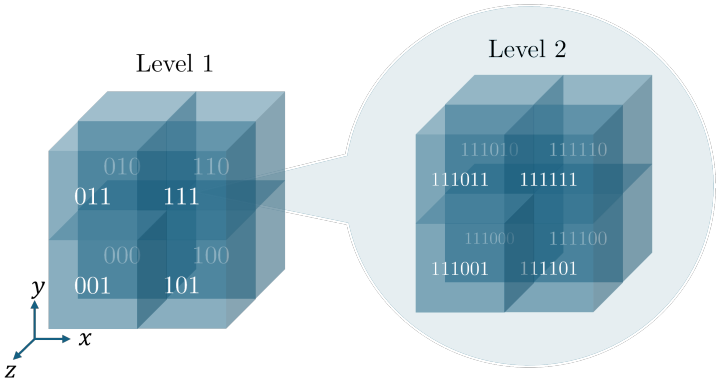


Figure 3.2: An illustration of Morton codes for clusters in level 1, and for the clusters within cluster 111 at level 2

To gain the advantage of spatial locality, octree Morton codes are often interleaved. Interleaving is a process where the x, y and z coordinates are expressed in bits and are intertwined to form the Morton code. In this implementation, the Morton codes are not interleaved and are simply implemented as defined above.

3.1.2. Finding Neighbour and Interaction Clusters

Locating the neighbour clusters of a selected cluster begins with determining the coordinates of the cluster center based on its Morton code. Once the coordinates are found, a vector in each of the 26 directions towards the centers of the neighbour clusters is appended, resulting in a set of 26 potential neighbour cluster coordinates. The code then attempts to find the Morton codes whose centers correspond to these coordinates in the list of available Morton codes at that level. If they are found, the Morton code is appended to the neighbouring list. If they are not found, the program has attempted to find a Morton code outside the domain, or corresponding to a cluster that does not contain any particles, and so that set of coordinates is ignored.

To determine the interaction list, the neighbours of the selected cluster's parent are found using the previously mentioned method, and their children are added to the interaction list. The neighbours of the selected cluster itself are also found. If any cluster appears as both a neighbour of the selected cluster and a child of the selected cluster parent's neighbours, it is removed from the interaction list.

3.2. FMM Algorithm

The FMM algorithm requires the positions and strength vectors of the particles. The first step of the algorithm is the formation of the octree. The extremes of the particle positions are used to construct the dimensions of the domain. The largest dimension is used to force-symmetrize the domain into a cube, as this formulation of FMM does not support the use of clusters with non-symmetric dimensions. Following this, the octree is generated by subdividing the domain into child clusters recursively until the subdivision criterion set by the user is reached.

The FMM algorithm separates the particle interactions into two classes, the *near field*, and the *far field*. The strategy implemented in FMM is to handle the interactions with particles in the near field directly, as these are usually the nearest to the target particle and thus usually are the most impactful. Interactions from particles in the far field are not handled directly, instead they are approximated. First, the near field and far field for any given particle are defined here:

Near field The region in the domain where the effect of source particles is computed directly on the target particle. In FMM, this consists of the leaf cluster containing the target particle, as well as the cluster's neighbours.

Far field The region in the domain where the effect of source particles on the target particle is approx-

imated through multipole and local expansions. This is any region of the domain which is not in the near field.

For clarity, particles that fall in the near field of a target particle are assigned into a set denoted N_{near}, while those that fall in the far field are assigned to a set N_{far}. It is important to note that the sets N_{near} and N_{far} are not fixed throughout the implementation, and are based on the different near and far fields in relation to different leaf clusters.

The next step is the upwards pass, which consists of the particle to multipole (P2M) and multipole to multipole (M2M) operations. The P2M operation is performed for all the leaf clusters. The aim of the P2M operation is to translate moments of particle strengths in a leaf cluster to the centre of the cluster. Once the P2M operation is completed for all leaf clusters in the octree, M2M operations translating the moments from children to parents are performed until level 2 is reached.

The multipole to local (M2L) operation, which makes up part of the downwards pass is performed for the interaction list of each cluster at each level. This is the reason the M2M operation is only performed until level 2. Due to the definition of interaction lists given in section 3.1, at level 1, there exists no interaction list, as all other clusters at that level are neighbours of the selected cluster. There is thus no need to perform M2M to that level as it cannot be used by M2L. Level 2 is thus the first level where the multipole expansions can be converted to local expansions via the M2L operation.

After the upwards pass, Morton codes for all clusters at all levels of the octree are added to a cluster list. Before the downwards pass is performed, any clusters which do not contain any particles are removed from the cluster list. This greatly reduces the computational effort required as the computationally heavy M2L operations are bypassed for empty clusters. The downwards pass begins at level 2 and the M2L operations are performed first for all clusters in the domain. This is where the moments saved in the cluster centres during the upwards pass are used, in combination with the VPM kernel implemented, to obtain local expansions. These local expansions carry information on the induced velocities caused by these moments as they have undergone the kernel implementation. The local expansions are then passed down to their children through the local to local (L2L) interaction. It is necessary that the M2L operation is performed before the L2L operation at a given level, such that all the far field interactions are taken into account before they are passed down.

Finally, once the local expansions are passed down to the leaf clusters, they are converted to particle far field induced velocities $\mathbf{u}_{\text{far}}$ and velocity gradient tensor $\nabla\mathbf{u}_{\text{far}}$ through the local to particle (L2P) operation. The near field velocities and gradients ($\mathbf{u}_{\text{near}}$ and $\nabla\mathbf{u}_{\text{near}}$) are computed through the particle to particle (P2P) operation between all particles in the near field.

At the end of the downwards pass, the particles are grouped by the clusters they are in, and their induced velocities are saved as an array per cluster. In order to return the velocities array, the induced velocities are reordered to match the order of particles in the input positions array. The same is done for the velocity gradient tensor if it is required by the user.

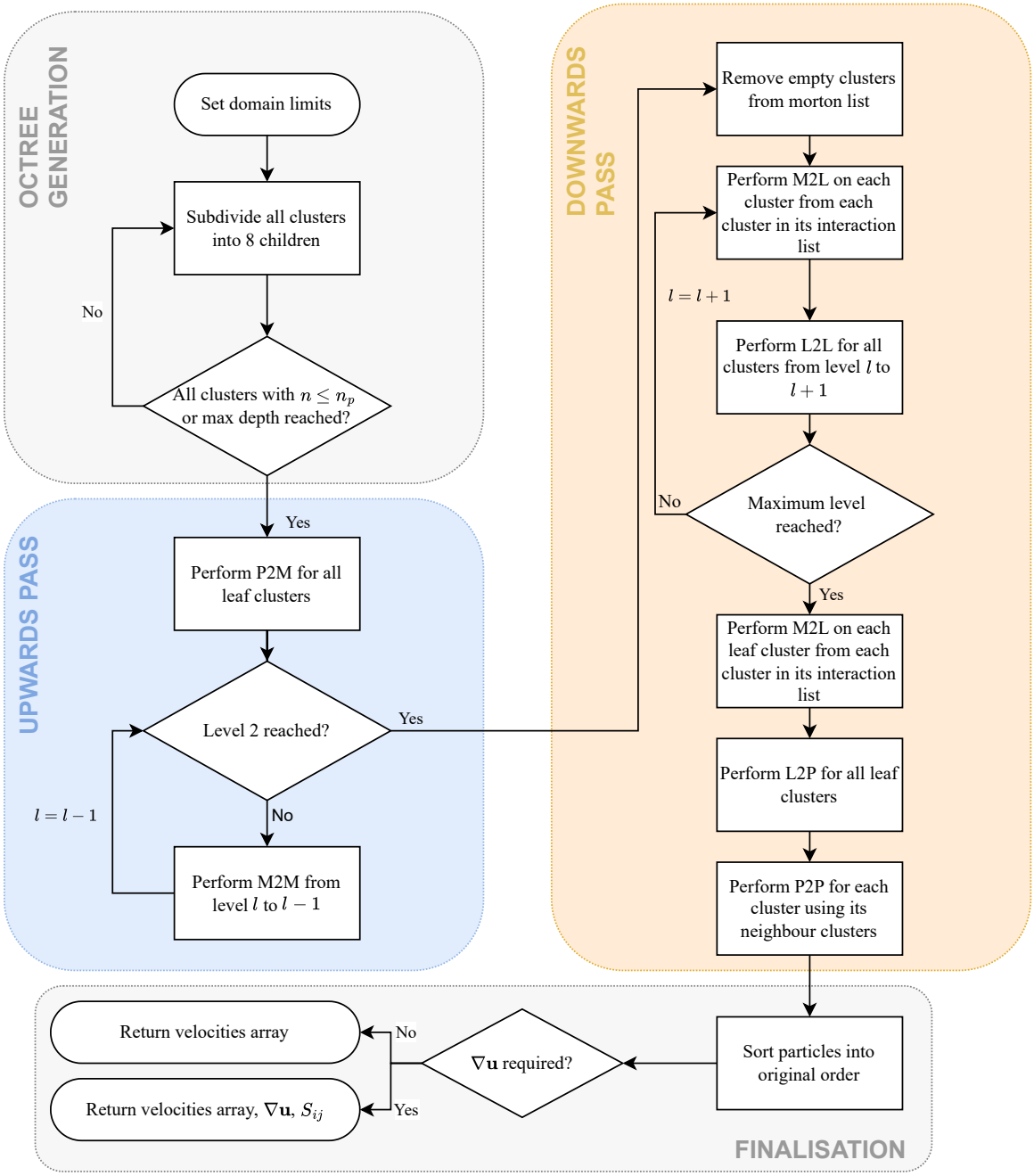


Figure 3.3: Flowchart of implemented FMM algorithm

3.3. Multipole and Local Expansions

In this implementation of FMM, multipole expansions are formed using Cartesian Taylor series. Since they are performed in 3D, the derivatives of the Taylor series are expressed in integer multi-index form $\boldsymbol{k} = [k_1\ k_2\ k_3]$ which are the order of derivatives in the x_1, x_2 and x_3 directions respectively. Operations using $\boldsymbol{k}$ follow the rules of integer multi-index notation. Namely, and with $\boldsymbol{n}$ being another arbitrary multi-index vector:

$$|\boldsymbol{k}| = k_1 + k_2 + k_3$$

$$\boldsymbol{k!} = k_1!k_2!k_3!$$
$$\boldsymbol{x^k} = x_1^{k_1}x_2^{k_2}x_3^{k_3}$$
$$\boldsymbol{D_y^k} = D_{y_1}^{k_1}D_{y_2}^{k_2}D_{y_3}^{k_3}$$
$$\binom{\boldsymbol{k}}{\boldsymbol{n}} = \binom{k_1}{n_1}\binom{k_2}{n_2}\binom{k_3}{n_3}$$
$$\boldsymbol{k} \leq p : 0 \leq k_1 \leq p \wedge 0 \leq k_2 \leq p \wedge 0 \leq k_3 \leq p$$
$$\boldsymbol{n} \leq \boldsymbol{k} : 0 \leq n_1 \leq k_1 \wedge 0 \leq n_2 \leq k_2 \wedge 0 \leq n_3 \leq k_3$$

Summations are performed such that:

$$\sum_{\boldsymbol{n}\leq\boldsymbol{k}} = \sum_{\boldsymbol{n}\in\{\boldsymbol{n}\in\mathbb{Z}^3:\boldsymbol{n}\leq\boldsymbol{k}\}} .$$

i.e. all combinations of the vector $\boldsymbol{n}$ that satisfy $n_1, n_2, n_3 \geq 0$ and $\boldsymbol{n} \leq \boldsymbol{k}$ as previously defined are substituted into a single summation. It is not a multiplicative sum:

$$\sum_{\boldsymbol{n}\leq\boldsymbol{k}} \neq \sum_{n_1\leq k_1}\sum_{n_2\leq k_2}\sum_{n_3\leq k_3} .$$

3.3.1. Particle to Particle (P2P)

The P2P operation is performed to calculate the induced velocity through the WL kernel, using Equation 2.8. In FMM, this interaction is performed in the near-field, the region that consists of the current particle's leaf cluster and its neighbours. The effect of particles in the far field, on the other hand, is approximated through the multipole and local expansions performed in the FMM algorithm.

The induced velocity of source particles at position $\mathbf{x}^s$ in the near field of the target particle at position $\mathbf{x}^t$, ensuring that self induction is omitted ($s \neq t$), is given by:

$$\mathbf{u}^t_{\text{near}}(t) = \sum_{s\in N_{\text{near}}|s\neq t} \mathbf{K}_\sigma(\mathbf{x}^t - \mathbf{x}^s(t)) \times \mathbf{\Gamma}^s(t) , \tag{3.1}$$

Where $\mathbf{u}^t_{\text{near}}$ is the portion of induced velocity on a target particle t by the particles in its near field.

Velocity gradient calculation

The P2P operation can also be used to calculate the influence of the velocity gradients arising from the near field interactions $\nabla\mathbf{u}_{\text{near}}$ using Equation 3.2 where $\zeta_\sigma = \zeta(\rho)/\sigma_{ts}$:

$$\nabla\mathbf{u}^t_{\text{near}} = \sum_{s\in N_{\text{near}}} \frac{-q(\rho)}{\|\mathbf{r}\|^3}\mathbf{skew}(\mathbf{\Gamma}^s) + \left(\frac{3q(\rho)}{\|\mathbf{r}\|^5} - \frac{\zeta_\sigma}{\|\mathbf{r}\|^2}\right)(\mathbf{r}\otimes(\mathbf{r}\times\mathbf{\Gamma}^s)) \quad \text{where} \quad \mathbf{r} = \mathbf{x}^t - \mathbf{x}^s . \tag{3.2}$$

3.3.2. Particle to Multipole (P2M)

This operation forms the first part of the upwards pass. In P2M, the strengths of the individual particles are aggregated and translated into the centre of the leaf clusters in which they are located. This is done for each $\boldsymbol{k}$ and stored in memory for each leaf cluster. The calculation of the moments for each $\boldsymbol{k}$ is shown in Equation 3.3, where $\mathbf{x}^s$ denotes each particle in the cluster and $\mathbf{x}^c$ are the coordinates of the cluster centre. We define a set of the particles present in an arbitrary leaf cluster as N_{cluster}. The moment vector for a multi-index vector k is:

$$\boldsymbol{m_k}(c) = \sum_{s \in N_{\text{cluster}}} (\mathbf{x}^s - \mathbf{x}^c)^{\boldsymbol{k}} \, \boldsymbol{\Gamma}^s \ . \tag{3.3}$$

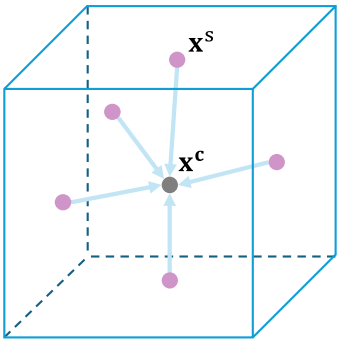


Figure 3.4: Illustration of P2M operation

3.3.3. Multipole to Multipole (M2M)

This operation is performed between different levels. The purpose is to translate the moments calculated in the previous step, using P2M operations, from child clusters onto their parent clusters. The moment calculated for each parent $\boldsymbol{m}(\mathrm{p})$ is therefore the sum of the translation of the moment of its children clusters $c = \{c_1, c_2, ..., c_8\}$:

$$\boldsymbol{m_k}(\mathrm{p}) = \sum_{c_i \in c} \sum_{\boldsymbol{n} \leq \boldsymbol{k}} \binom{\boldsymbol{k}}{\boldsymbol{n}} (\mathbf{x}^{\mathrm{c}_i} - \mathbf{x}^{\mathrm{p}})^{\boldsymbol{k}-\boldsymbol{n}} \boldsymbol{m_n}(\mathrm{c}_i) \ . \tag{3.4}$$

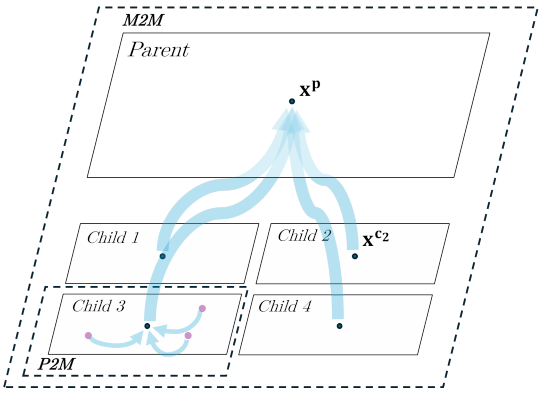


Figure 3.5: Illustration of P2M and M2M operations in 2D. Particles are presented by the violet dots, while cluster centers are presented by the black dots.

3.3.4. Multipole to Local (M2L)

This operation is performed between clusters on the same level. For each cluster on a given level l, the interaction list is first determined. For each cluster in the interaction list, the M2L translation is performed. The moments of the source cluster are used to build a local Taylor series expansion in the clusters of the interaction list. Through this, clusters can receive moment information of the moments present in other clusters. This is central to the working method of FMM, as the far field interactions are approximated by the Taylor series resulting from the M2L operation. The transformation of the moments from one cluster into local expansions in another is given as:

$$\boldsymbol{l_k}(\mathbf{x}^c) = (-1)^{|\boldsymbol{k}|} \sum_{\boldsymbol{n} \leq p} \frac{(\boldsymbol{k}+\boldsymbol{n})!}{\boldsymbol{k}!\boldsymbol{n}!} \boldsymbol{a_{k+n}}(\mathbf{x}^c, \mathbf{y}^c) \times \boldsymbol{m_n}(\mathbf{y}^c) \ , \tag{3.5}$$

Where $\mathbf{x}^c$ and $\mathbf{y}^c$ are the center positions of an arbitrary target and source cluster respectively, and $\boldsymbol{a_k}$ are the Taylor coefficients of the Winckelmans-Leonard kernel, and are defined as:

$$\boldsymbol{a_k} = \frac{1}{\boldsymbol{k}!} D_{\boldsymbol{y}}^{\boldsymbol{k}} \boldsymbol{K}_\sigma(\mathbf{x}^c, \mathbf{y}^c) \ . \tag{3.6}$$

There are several methods to calculate the kernel Taylor coefficients $\boldsymbol{a_k}$. Using a symbolic differentiation library, such as SymPy [33] was attempted, which offers the advantage of performing the derivative calculations for each $\boldsymbol{k}$ only once and subsequently substituting different values of $\mathbf{x}^c - \mathbf{y}^c$ when needed. While this method is accurate, substituting the numerical values into large symbolic equations proved too slow for the large amount of M2L evaluations that FMM performs. Central difference schemes are also a proposed method to obtain $\boldsymbol{a_k}$. The advantage of this method is that it is easily parallelisable, and so can be quickly computed in parallel GPU architectures. However, it is excepted to be prone to truncation and rounding errors with the use of single float precision.

A method proposed by Wee et al. using recurrence relations [49] was found to be the most suitable due to its high speed and immunity to truncation errors that could be present in finite difference schemes. This method, used for calculating the Taylor coefficients $\boldsymbol{a_k}$ is discussed in detail in subsection 3.4.2.

3.3.5. Local to Local (L2L)

Analogous to the M2M operation, this operation is performed in the downwards pass to translate the local expansions from parent to child clusters. Once the M2L operations have all been performed for level l, the local expansions are passed on to the children in level $l+1$. The local expansion in a child cluster c_i is given by:

$$\boldsymbol{l_k}(\mathrm{c}_i) = \sum_{\boldsymbol{k} \leq \boldsymbol{n} \leq p} \binom{\boldsymbol{n}}{\boldsymbol{k}} (\mathbf{x}^{\mathrm{c}_i} - \mathbf{x}^{\mathrm{p}})^{\boldsymbol{n}-\boldsymbol{k}} \boldsymbol{l_n}(\mathrm{p}) \ . \tag{3.7}$$

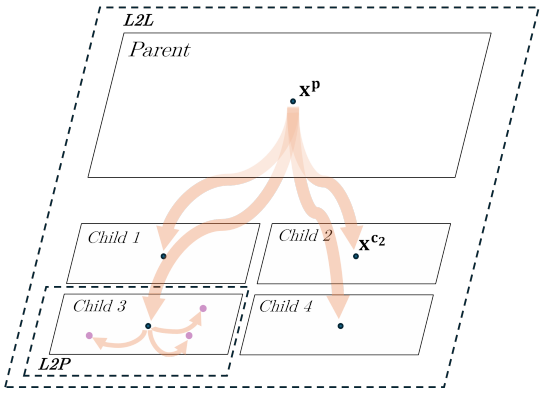


Figure 3.6: Illustration of L2L and L2P operations in 2D. Particles are presented by the violet dots, while cluster centres are presented by the black dots.

3.3.6. Local to Particle (L2P)

Once the deepest level of the tree is reached, the local expansions calculated at the leaf clusters is transmitted to the particles themselves as an induced velocity. These local expansions represent the FMM approximation of the induced velocities from particles in the far field. The induced velocity on a target particle t from particles in its far field is:

$$\mathbf{u}^t_{\text{far}}(t) = \sum_{\boldsymbol{k} \leq p} (\mathbf{x}^t - \mathbf{x}^c)^{\boldsymbol{k}} \, \boldsymbol{l}_{\boldsymbol{k}}(c) \tag{3.8}$$

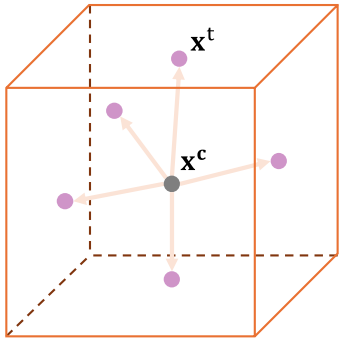


Figure 3.7: Illustration of L2P operation

Velocity gradient calculation using the central difference scheme

During the L2P step, the velocity gradient tensor resulting from the particles in the far field, $\nabla \mathbf{u}_{\text{far}}$ can also be calculated. This is done through the use of a central difference scheme (CDS). This method is also used by Berdowski [6]. In the case of Berdowski, FMM computes the velocity potential at the particles locations. Therefore, they use a 7-point stencil for the velocities, and a 25-point stencil for the velocity gradients. Here, the velocities are the output of FMM, so only a 7-point stencil is required for a second order accurate CDS calculation of the velocity gradients. The user is also given the options of a 4th order and 6th order accurate scheme, which require the use of 13-point and 19-point stencils respectively. The formulas for the second, fourth and sixth order accuate schemes are given in Equation 3.9, Equation 3.10 and Equation 3.11 respectively, while Figure 3.8 illustrates the locations required to build the stencil in a 2D case.

$$\left.\frac{\partial u_i}{\partial x_i}\right|_{\mathbf{x}^t} \approx \frac{u_i^{\mathbf{x}^t+\delta\mathbf{e}_i} - u_i^{\mathbf{x}^t-\delta\mathbf{e}_i}}{2\,\delta} \,, \tag{3.9}$$

$$\left.\frac{\partial u_i}{\partial x_i}\right|_{\mathbf{x}^t} \approx \frac{-u_i^{\mathbf{x}^t+2\delta\mathbf{e}_i} + 8u_i^{\mathbf{x}^t+\delta\mathbf{e}_i} - 8u_i^{\mathbf{x}^t-\delta\mathbf{e}_i} + u_i^{\mathbf{x}^t-2\delta\mathbf{e}_i}}{12\,\delta} \,, \tag{3.10}$$

$$\left.\frac{\partial u_i}{\partial x_i}\right|_{\mathbf{x}^t} \approx \frac{u_i^{\mathbf{x}^t-3\delta\mathbf{e}_i} - 9\,u_i^{\mathbf{x}^t-2\delta\mathbf{e}_i} + 45\,u_i^{\mathbf{x}^t-\delta\mathbf{e}_i} - 45\,u_i^{\mathbf{x}^t+\delta\mathbf{e}_i} + 9\,u_i^{\mathbf{x}^t+2\delta\mathbf{e}_i} - u_i^{\mathbf{x}^t+3\delta\mathbf{e}_i}}{60\,\delta} \,. \tag{3.11}$$

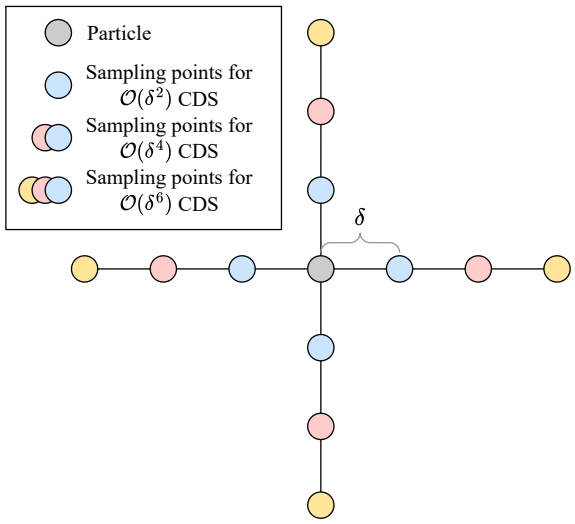


Figure 3.8: 2D representation of CDS stencil

For the CDS, the step size δ is dependent on the machine rounding error, and the smoothness of the velocity field. The stencil length was selected to be smaller than the particle spacing so that the velocity field is locally sufficiently smooth $\delta << h$. With numerical differentiation, there is a tradeoff between rounding errors and truncation errors. A suitable value for δ requires further analysis, as CDS schemes computed numerically are ill-conditioned [19] and an optimal value for δ can only be found if higher derivatives of the function being differentiated are known [8].

3.4. Performance considerations of the FMM code

Along with the accuracy and stability of the FMM code, improving its speed of computations is also of paramount importance. Since FMM is used for its speed, faster FMM computations provide greater flexibility to increase the accuracy of the code, and to use a larger number of particles where this is necessary to better describe the flow field. This section describes the methods used to obtain a performant FMM code.

3.4.1. The Taichi Language

Taichi [25] is a domain-specific language tailored towards computations on spatially sparse data structures, which is integrated as a library within python. Taichi is a statically-typed compiled language which potentially offers significant speed-up in computational performance compared to python. Furthermore, the taichi compiler automatically parallelises the outermost for loop on both CPUs and GPUs, further increasing the performance capabilities for parallelisable operations. Taichi is also force-inlined and thus does not support recursion at the time of this writing [27]. This meant that certain operations, such as calculating the Taylor coefficients, as demonstrated in subsection 3.4.2 had to be written using a tabulation approach (bottom-up computation of all derivatives) rather than a memoization (top-down computation of derivatives as required) approach, which would have required the use of recursive functions.

FMM operations, such as the M2M and M2L operations, rely on a large number of summations. They therefore often require the use of nested for loops which significantly slow down code execution in interpreted languages such as python. This is resulting from the overhead that is required by the interpreter for each iteration within the loop. Such operations are thus passed on from the python interpreter, referred to as the *host* to the taichi kernel to be run on the GPU, referred to as the *device*.

Taichi supports the use of multiple backends for CPU and GPU. Most GPU backends only support single float precision. Backends such as CUDA which also support double float precision, are significantly slower in handling their computation. The FMM implementation of this thesis therefore uses single float

precision for computations by default. The only exception for this is for the calculation of factorials as for values of $p > 16$, the maximum factorial computed is $(2p)! = \mathcal{O}(10^{38})$ which approaches the maximum number representable by single float precision.

3.4.2. Computing Taylor Coefficients Using Recurrence Relations

The Taylor coefficients of the kernel $\boldsymbol{a_k}$ are computed using recurrence relations. The recurrence relations method for the Winckelmans-Leonard (WL) kernel is based on the methodology from Wee et al. [49] and is outlined below.

First, the potential function for the WL kernel is given:

$$\phi_\sigma^{WL}(\mathbf{x},\mathbf{y}) = \frac{1}{4\pi}\frac{|\mathbf{x}-\mathbf{y}|^2 + \frac{3}{2}\sigma^2}{(|\mathbf{x}-\mathbf{y}|^2+\sigma^2)^{3/2}} \,. \tag{3.12}$$

Where $\mathbf{x}$ is an arbitrary target position vector, and $\mathbf{y}$ is an arbitrary source position vector. This is split into two components:

$$\phi_\sigma^{WL}(\mathbf{x},\mathbf{y}) = \frac{\phi_\sigma^1(\mathbf{x},\mathbf{y})}{4\pi} + \frac{\sigma^2\phi_\sigma^3(\mathbf{x},\mathbf{y})}{8\pi} \,, \tag{3.13}$$

With $\phi_\sigma^\nu(\mathbf{x},\mathbf{y}) = (|\mathbf{x}-\mathbf{y}|^2+\sigma^2)^{-\nu/2}$. Taylor coefficients can be calculated for ϕ_σ^ν in the same way as the WL kernel:

$$T_{\boldsymbol{k}}^\nu = \frac{1}{\boldsymbol{k}!} D_{\mathbf{y}}^{\boldsymbol{k}} \phi_\sigma^\nu(\mathbf{x},\mathbf{y}) \,. \tag{3.14}$$

The following recurrence relation is used to determine the values for the Taylor coefficients:

$$T_{\boldsymbol{k}}^\nu = \frac{(2|\boldsymbol{k}|+\nu-2)\sum_{i=1}^3 (x_i-y_i) T_{\boldsymbol{k}-\boldsymbol{e_i}}^\nu - (|\boldsymbol{k}|+\nu-2)\sum_{i=1}^3 T_{\boldsymbol{k}-\boldsymbol{2e_i}}^\nu}{|\boldsymbol{k}|(|\mathbf{x}-\mathbf{y}|^2+\sigma^2)} \,, \tag{3.15}$$

With $\boldsymbol{e_i}$ being the Cartesian basis vector. The initial value of the recurrence relation $T_{\boldsymbol{0}}^\nu$ is the potential function itself:

$$T_{\boldsymbol{0}}^\nu = (|\mathbf{x}-\mathbf{y}|^2+\sigma^2)^{-\nu/2} \,. \tag{3.16}$$

Additionally, if any of the components of $\boldsymbol{k}$ is less than 0, then $T_{\boldsymbol{k}}^\nu = 0$.

As a result of Equation 3.13:

$$T_{\boldsymbol{k}}^{WL} = \frac{T_{\boldsymbol{k}}^1}{4\pi} + \frac{\sigma^2 T_{\boldsymbol{k}}^3}{8\pi} \,. \tag{3.17}$$

The WL kernel Taylor coefficient $\boldsymbol{a_k}$ is reconstructed from the WL potential Taylor coefficients $T_{\boldsymbol{k}}^{WL}$:

$$\boldsymbol{a_k} = -\sum_{i=1}^3 \boldsymbol{e}_i (k_i+1) T_{\mathbf{k}+\hat{\mathbf{e}}_i}^{WL} \,. \tag{3.18}$$

This implementation of the recurrence relations to calculate the Taylor coefficients is done sequentially, as the values of higher derivatives of the potential function $T_{\boldsymbol{k}}^{WL}$ rely on the lower derivatives. The order of calculation is by incrementing k_3 derivatives first, then k_2 and finally k_1. The order of $T_{\boldsymbol{k}}^\nu$ derivative coefficient calculation is more clearly illustrated in Figure 3.9. This order of calculations ensures that all required lower derivatives have already been calculated and are avaiable to calculate higher derivatives through the recurrence relation.

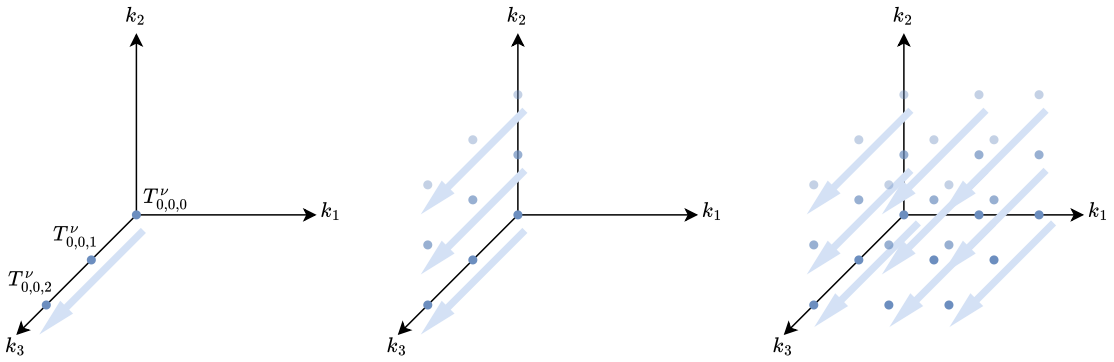


Figure 3.9: Order of calculation of Taylor coefficients $T^{\nu}_{\boldsymbol{k}}$

An array that contains the kernel Taylor coefficients $\boldsymbol{a_k}$ is generated for each unique value of $\mathbf{x} - \mathbf{y}$ upto the greatest value of $\boldsymbol{k} + \boldsymbol{n}$ from Equation 3.5, which is equivalent to $\mathbf{2}\boldsymbol{k}$ when $\boldsymbol{n} = \boldsymbol{k}$. The summation in Equation 3.5 uses all the possible values of $\boldsymbol{k}$ in the range of $(\mathbf{0}, \mathbf{2}\boldsymbol{k})$ and therefore this tabulation approach allows for fast computation of partials, without performing any wasted calculations which will not be used in Equation 3.5.

4

Error Quantification

To ensure the FMM code developed provides an accurate estimate of the induced velocities in the computational domain, the errors of the FMM calculations are quantified for different test cases, using the computations performed using the direct method as a benchmark. Lamb-Oseen vortex and vortex ring test cases are investigated, with various simulation parameters changed, including the FMM octree depth and Taylor series truncation order, providing an overview of the sensitivity of FMM accuracy to inform the user's parameter selection. The chapter begins with an overview of the method used for error quantification, presented in section 4.1. The test cases are then introduced in section 4.2. The effect of the number of particles for different flow scenarios is given in section 4.3. The effect of overlap ratio and octree depth are provided in section 4.4 and section 4.5 respectively.

4.1. Error quantification method

To determine whether implementing FMM introduces errors which significantly affect the evolution of the particle positions, some key statistical metrics were selected. A suitable parameter to assess the absolute error across multiple particles is the root mean square error (RMSE). The difference in each component of the induced velocity (u_x, u_y, u_z) by the direct method and FMM can be assessed using this parameter. The implemented equation to calculate the RMSE is:

$$\mathrm{RMSE} = \sqrt{\frac{\sum(u_x^{\mathrm{FMM}} - u_x^{\mathrm{Direct}})^2 + \sum(u_y^{\mathrm{FMM}} - u_y^{\mathrm{Direct}})^2 + \sum(u_z^{\mathrm{FMM}} - u_z^{\mathrm{Direct}})^2}{3N}} \,, \tag{4.1}$$

Where $3N$ represents the total number of velocity components, three for each particle. This definition of RMSE is scale-dependent and might not be suitable for comparison between different test cases. RMSE provides a measure of the absolute error which is not indicative of its overall significance on the flow velocities. RMSE is thus normalised by the maximum particle velocity magnitude in the computational domain to obtain a normalised value RMSE^* :

$$\mathrm{RMSE}^* = \frac{\mathrm{RMSE}}{\max(\|\boldsymbol{u}\|)} \,. \tag{4.2}$$

RMSE^* provides an indication of the overall error, but does not indicate where these errors are present. Large relative errors on small velocity components will not significantly affect the flow field while those on large components will. Furthermore, large relative errors are expected for near-zero velocity components.

To characterize the FMM error, the RMSE^* is supplemented with a histogram of the relative error as in Figure 4.1a. A histogram of relative error visualises the spread of the error in all the calculated velocity components. The magnitude of the relative error is not taken in order to maintain information

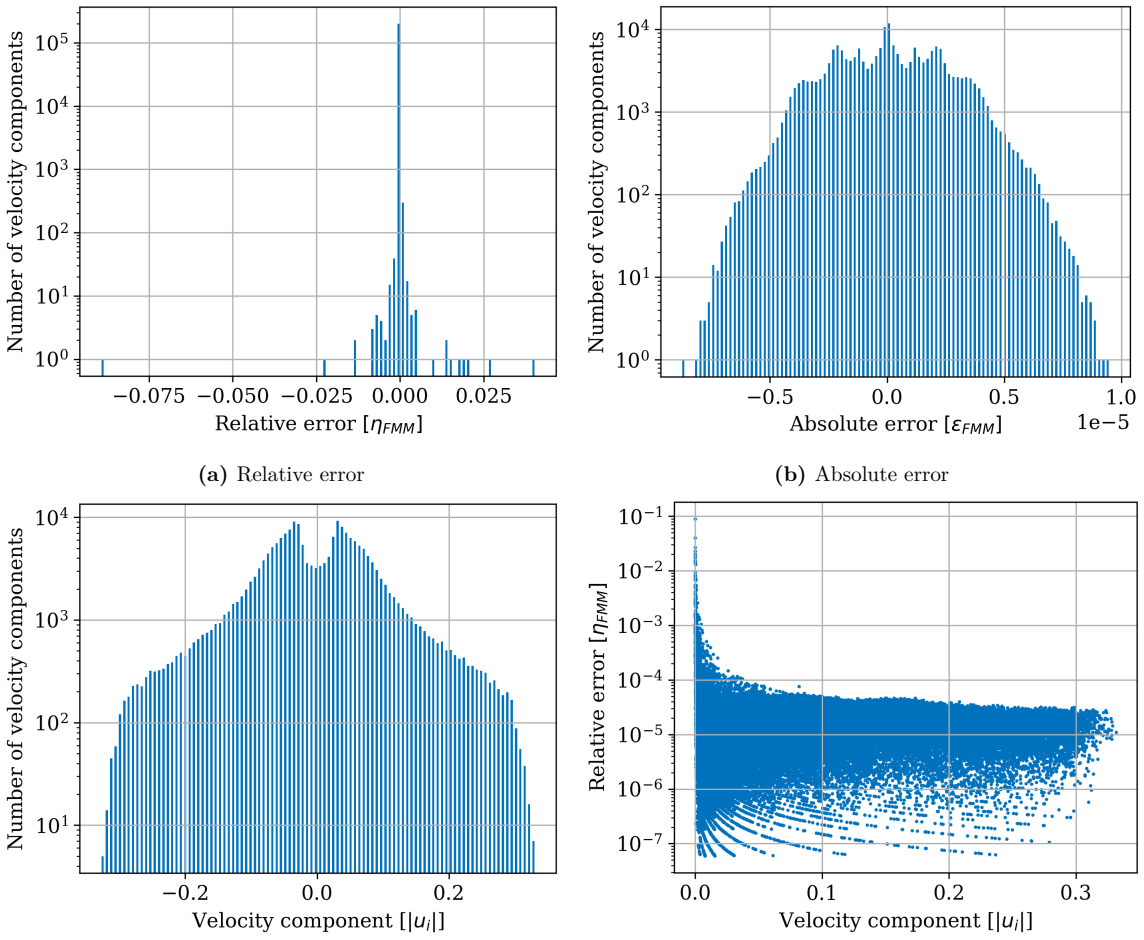
on whether FMM is over or underestimating the induced velocities. The relative error of FMM (η_{FMM}) is given by:

$$\eta_{\text{FMM}} = \frac{u_i^{\text{FMM}} - u_i^{\text{Direct}}}{u_i^{\text{Direct}}} \ . \tag{4.3}$$

The absolute error is also shown as in Figure 4.1b, and velocity component magnitudes as in Figure 4.1c. Comparing the absolute errors and velocity magnitude histograms gives an indication of whether the errors being made are likely to affect the induced velocities significantly. Additionally, the velocity component magnitude histogram serves as a check that two different runs have similar velocity magnitudes, and thus can be compared using RMSE* . The absolute error of FMM (ϵ_{FMM}) is given by:

$$\epsilon_{\text{FMM}} = u_i^{\text{FMM}} - u_i^{\text{Direct}} \ . \tag{4.4}$$

Additionally, Figure 4.1d shows a plot of the relative error compared to the velocity component magnitudes. This plot indicates whether the errors are occurring for the larger velocity components or for the smaller ones. If the error is large for small velocity components, then it most likely attributable to near-zero relative error blow-up and does not constitute a significant effect on the flow field. A large error at the region of large velocity components is more indicative of a poor FMM estimate of the velocity field.



(a) Relative error

(b) Absolute error

(c) Velocity components

(d) Relative error per velocity component

Figure 4.1: Example of supplementary statistics for FMM quality analysis

For the remaining test cases in this chapter, a sample of results resembling those in Figure 4.1 are provided in Appendix A to portray the conformity of the FMM results to the accuracy requirements detailed in this section. Some of these results are also included in this chapter when they are required for analysis.

RMSE for positions and velocity gradient tensor

Similarly for the velocities, RMSE^* for the particle positions and RMSE for the velocity gradient tensor are defined, which will be used to analyse the results in chapter 5. For some cases, such as the Lamb-Oseen vortex, the flow is rotational and the particles return to their original positions after some time, therefore the maximum distance magnitude travelled by a particle in the flow ($\max(d)$) is used to normalise the positions RMSE.

$$\text{RMSE} = \sqrt{\frac{\sum(x^{\text{FMM}} - x^{\text{Direct}})^2 + \sum(y^{\text{FMM}} - y^{\text{Direct}})^2 + \sum(z^{\text{FMM}} - z^{\text{Direct}})^2}{3N}} \ , \tag{4.5}$$

And,

$$\text{RMSE}^* = \frac{\text{RMSE}}{\max(d)} \ . \tag{4.6}$$

For the velocity gradient tensor, which is of (3×3) size for each particle:

$$\text{RMSE} = \sqrt{\frac{\sum\sum_i^3\sum_j^3 \left(\frac{\partial u_i}{\partial x_j}^{\text{FMM}} - \frac{\partial u_i}{\partial x_j}^{\text{Direct}}\right)^2}{9N}} \ . \tag{4.7}$$

4.2. Description of the Test Cases

4.2.1. Lamb-Oseen Vortex

The Lamb-Oseen vortex simulates the effect of a line vortex present in the center of a physical domain, with the vorticity initially being zero in all other regions. At $t = 0$, the vortex begins as a point vorticity at the center and spreads for $t > 0$ into a Gaussian distribution of vorticity due to the effect of viscosity. The distribution of vorticity in time and space due to the line vortex is given by Equation 4.8, where r is the perpendicular distance from the line vortex. Because the Lamb-Oseen vortex would contain a singularity at $t = 0$, the particle initialization is performed at any non-zero time. With a Lamb-Oseen vortex oriented in the positive z direction the vorticity is given by:

$$\omega_z(r,t) = \frac{\Gamma}{4\pi\nu t}\exp\left(-\frac{r^2}{4\nu t}\right) \quad \text{with} \quad r = \sqrt{dx^2 + dy^2} \ . \tag{4.8}$$

While Equation 4.9 describes the resulting analytical circumferential velocity of particles in a Lamb-Oseen vortex:

$$u_\theta = \frac{\Gamma}{2\pi r}\left[1 - \exp\left(-\frac{r^2}{4\nu t}\right)\right] \ . \tag{4.9}$$

When setting up the initialization for the Lamb-Oseen vortex for the test cases, the center vortex is converted through Equation 4.8 into discretized vorticity for each particle in the domain. The resultant velocity of each particle is then calculated through the P2P interactions of the particles in place of the analytical approach given in Equation 4.9. This is so that errors arising from the discretization of the field are removed from the computation of the FMM error, as the goal is to evaluate how accurate FMM is compared to direct simulation.

Due to the 2-dimensional shape of the induced velocity for the Lamb-Oseen vortex, the velocities in the z direction are zero for both the FMM and direct methods. In order to avoid division by zero when calculating the relative error of FMM for the Lamb-Oseen test case, only the induced velocities in the x and y direction are taken into account.

Table 4.1 shows the default simulation parameter values used in the single-timestep simulations performed for the Lamb-Oseen vortex test cases. The overlap ratio was selected due to the particle overlap constraint of VPM [51]. Which states that if an overlap ratio $\sigma/h < 1$ is selected, then the VPM simulation is at risk of becoming divergent. On the other hand, higher σ/h values become less representative of the physical reality of the simulation, as the induced velocities due to the regularised kernel become more diffused. Barba [3] gives an optimal range of around $1.0 < \sigma/h < 1.4$ for reducing the error, so a value of $\sigma/h = 1.25$ is taken for these simulations. A sensitivity analysis in section 4.4 is later performed to ensure that this value does not fall within the high error region for FMM.

The variables in Table 4.1 are varied in some tests to inform a study on the effect of these parameters on FMM accuracy. The reader is consulted to refer to this table for the values of the parameters when they are not provided explicitly.

Table 4.1: Default parameters of Lamb-Oseen single timestep runs

Parameter	Value
Vortex strength Γ	π
Vortex radius R	1.0
Kinematic viscosity ν	5×10^{-4}
Overlap ratio σ/h	1.25
Domain dimensions	$10 \times 10 \times 10$

The vortex radius R is defined as $R = \sqrt{4\nu t_0}$ where t_0 is the initialization time of the simulation.

4.2.2. Vortex Ring

A vortex ring constitutes a toroidal structure that convects through space along its central axis. The structure is the resultant of a ring of vorticity present within the torus, around which particles rotate, termed the *vortex ring core*. This can be thought of as a Lamb-Oseen vortex, which, rather than extending infinitely through space as a line, is folded into the shape of a ring. Helmholtz's second theorem of vorticity states that a vortex line must either extend infinitely in space or form a closed loop. In that way, the Lamb-Oseen vortex and vortex ring cases present the two possible configurations which fulfil Helmholtz's second theorem of vorticity. Figure 4.2 shows a diagram of a vortex ring and its defining parameters.

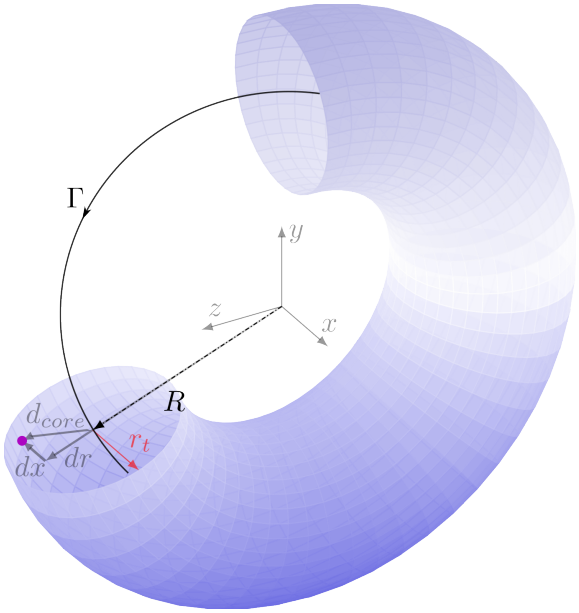


Figure 4.2: Illustration of a vortex ring and its defining parameters. An arbitrary particle within the ring is shown in violet. A vortex ring with this orientation of vorticity translates in the positive x direction.

For an unperturbed vortex ring which convects in the positive x direction the vorticity magnitude can be calculated using:

$$\|\boldsymbol{\omega}\| = \frac{\Gamma}{4\pi\nu t} \exp\left(\frac{-d_{core}^2}{4\nu t}\right) \ , \tag{4.10}$$

With d_{core} being the perpendicular distance to the vortex ring core as shown in Figure 4.2.

$$d_{core} = \sqrt{(dr - R)^2 + dx^2} \quad \text{and} \quad dr = \sqrt{dy^2 + dz^2} \ .$$

This results in the following expressions for the vorticity components:

$$\omega_x = 0 \quad , \quad \omega_y = -\|\boldsymbol{\omega}\| \sin\theta \quad , \quad \omega_z = \|\boldsymbol{\omega}\| \cos\theta \quad \text{with} \quad \theta = \arctan\frac{z}{y} \ . \tag{4.11}$$

The vortex ring was selected as a test case because, unlike the Lamb-Oseen vortex, velocities are induced in all three dimensions. Furthermore, the vortex ring contains an assymetry in dimensions, being much thinner in the x direction than the y and z directions. The particles take a toroidal shape, which tests the algorithm's capability of handling empty and sparsely populated clusters.

Table 4.2: Default parameters of vortex ring single timestep runs

Parameter	Value
Vortex strength Γ	π
Ring radius R	2.5
Ring thickness r_t	1.0
Kinematic viscosity ν	5×10^{-4}
Overlap ratio σ/h	1.25
Domain dimensions	$10 \times 10 \times 10$

4.3. Effect of the number of particles

The first effect to be investigated is the number of particles used in the simulation. Ideally, the number of particles would have no effect or a converging effect on the FMM error. This would ensure an upper bound for the error when increasing the number of particles. For this analysis, the number of particles is varied from $N = 1 \times 10^4$ to $N = 3 \times 10^6$. The simulations are performed for the Lamb-Oseen vortex and the vortex ring case to investigate whether convergence is dependent on the vorticity distribution in the domain.

4.3.1. Lamb-Oseen Vortex with random particle positions

In this test case, the domain was scaled with each N in order to keep the average particle distance constant. The original domain size was $(10 \times 10 \times 10)$ for the case with $N = 1 \times 10^4$, consisting of 10 particles per dimension. Domains with N particles were scaled up to $(dim \times dim \times dim)$ to maintain this same distance with the following rule:

$$dim = \left(\frac{N}{10}\right)^{\frac{1}{3}} . \tag{4.12}$$

In order to maintain the same induced velocity field independent of the number of particles, Equation 4.9 was scaled to maintain u_θ for the increase in the radius of the vortex using the following scaling rules:

$$\Gamma^{'} = \frac{dim}{10}\Gamma \quad \text{and} \quad r^{'} = \frac{dim}{10}r \quad \text{and} \quad (\nu t)^{'} = \frac{dim}{10}(\nu t) .$$

For this set of runs, a random particle distribution is selected, as this most closely resembles the particle distribution at most post initialization iterations of a VPM run, especially for those with high Reynolds numbers and thereby high amounts of flow mixing. A random particle distribution additionally contains some particles with a high proximity. Investigating the FMM robustness in accurately assessing target particles induced velocities in these cases is valuable. It is possible that large gradients of vorticity field due to the proximity of the particles might affect the accuracy of the Cartesian approximation of FMM. When creating the random particle distribution, the same random seed is used for all runs to ensure that random initialization does not affect the results.

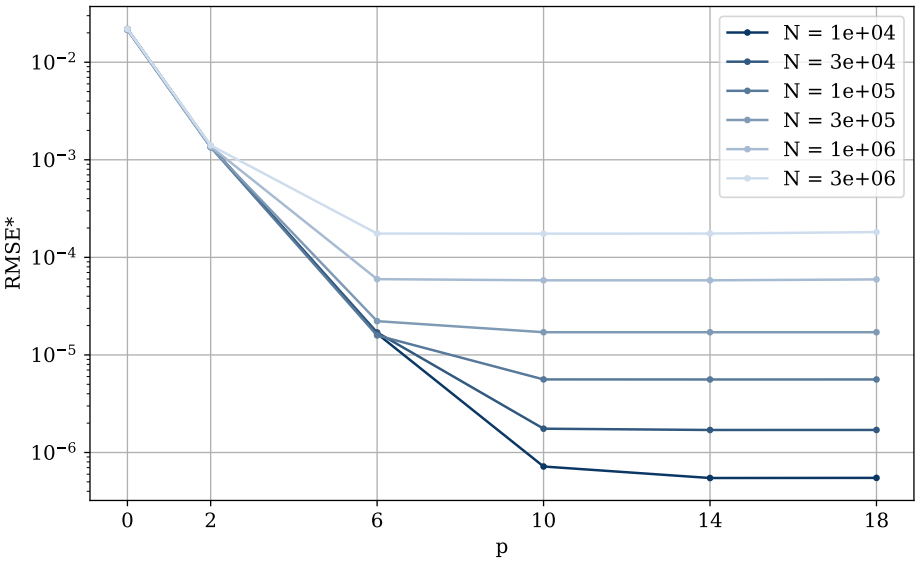


Figure 4.3: RMSE* of FMM estimates with varying p and N (Lamb-Oseen vortex, $d = 2$, $\sigma/h = 1.25$, random particle distribution, constant particle density)

Figure 4.3 shows the RMSE* for calculated for one-timestep simulations for different numbers of particles. The main feature is that an increase in the truncation order p generally results in a better FMM estimate of the true velocity field. Thus, the Cartesian FMM approximation is convergent towards the true solution. Furthermore, the errors of $p = 0$ and $p = 2$ are overlapping, indicating that they are independent of the number of particles. At higher values of p, the error begins to flatten out, indicating a diminishing effect of including additional Taylor terms, particularly at $p > 10$. The error in the high p region starts to become dependent on the number of particles. Evidently, simulations with a larger number of particles exhibit higher RMSE* values. The ratio of RMSE* between simulations is also roughly equivalent to the ratio of number of particles between them, indicating that a form of systematic error is contributing to this phenomenon.

The two main sources of error expected for FMM are the truncation error, which is associated with truncating the Taylor series approximation of the multipole expansions, and floating-point rounding errors, resulting from the single (32-bit) representation of floating point numbers in this code. In order to confirm the source of error, an equivalent set of runs with double precision is performed, and the results included in Figure 4.4.

Effect of arithmetic float precision

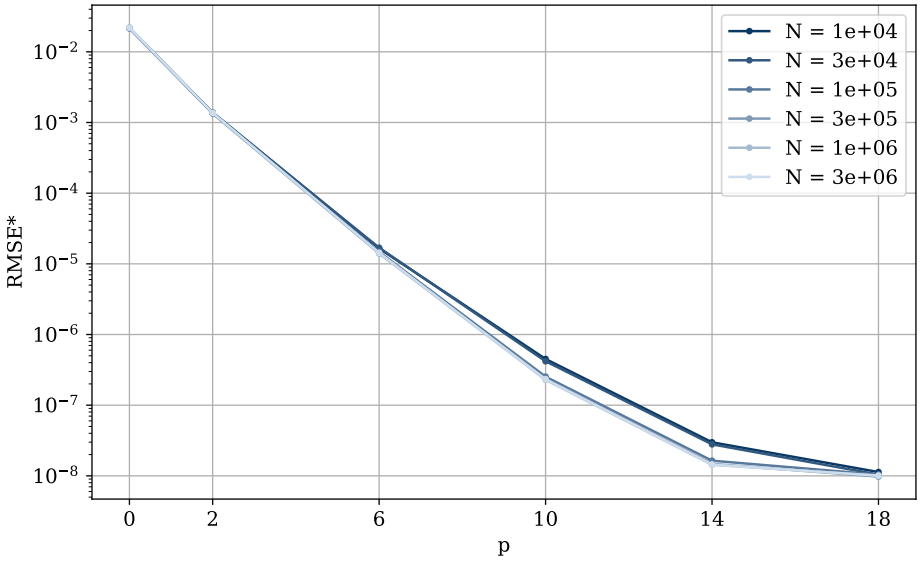
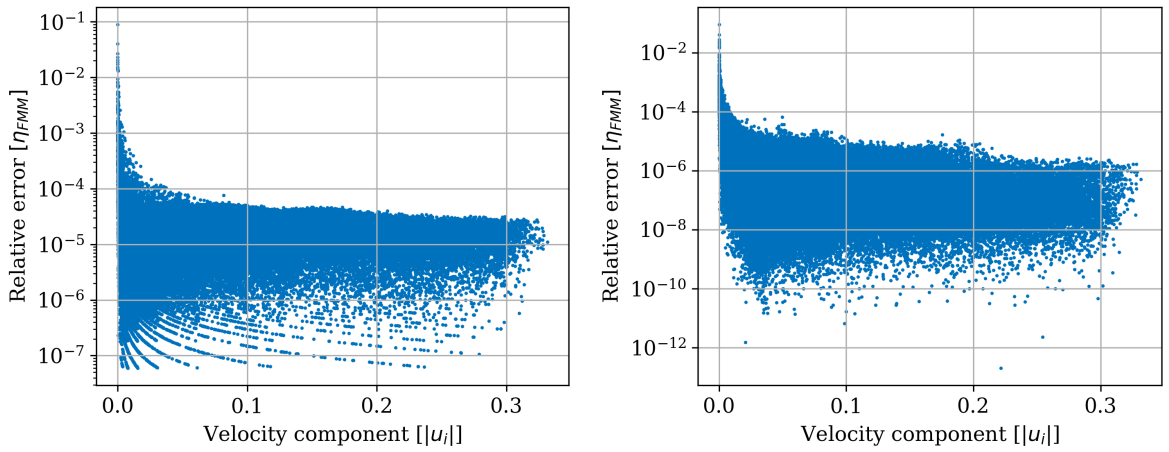


Figure 4.4: RMSE* of FMM estimates with varying p and N (Lamb-Oseen vortex, $d = 2$, $\sigma/h = 1.25$, random particle distribution, constant particle density, double precision)

The results from Figure 4.4 confirm the significant role of the rounding errors on the error plateau seen in Figure 4.3. In the test cases with higher p, the double precision runs continue to improve in accuracy, contrary to the single precision case. The RMSE* recorded for both precisions is also similar for low p, where the truncation error dominates. Since the rounding error in these cases is a fraction of the truncation error, it does not pose a significant effect on the prediction. It is only where the truncation error becomes very low where the effect of the rounding error becomes visible.

Furthermore, there is no increase in error for cases with a larger number of particles, indicating an accumulation of rounding error with the number of particles in single precision. This points to the P2M, L2P or P2P operations as potential sources of rounding error accumulation as they are dependent on the number of particles, unlike the M2M, M2L and L2L operations, which are independent thereof.

This analysis leads to the main deduction that there is a balance to be kept between the truncation errors, which dominate on the left hand side of Figure 4.3, and the rounding errors, which dominate on the right hand side of the plot. Careful consideration of the buildup of rounding errors is required when performing FMM in single precision, especially for test cases with a high number of particles.



(a) Single precision

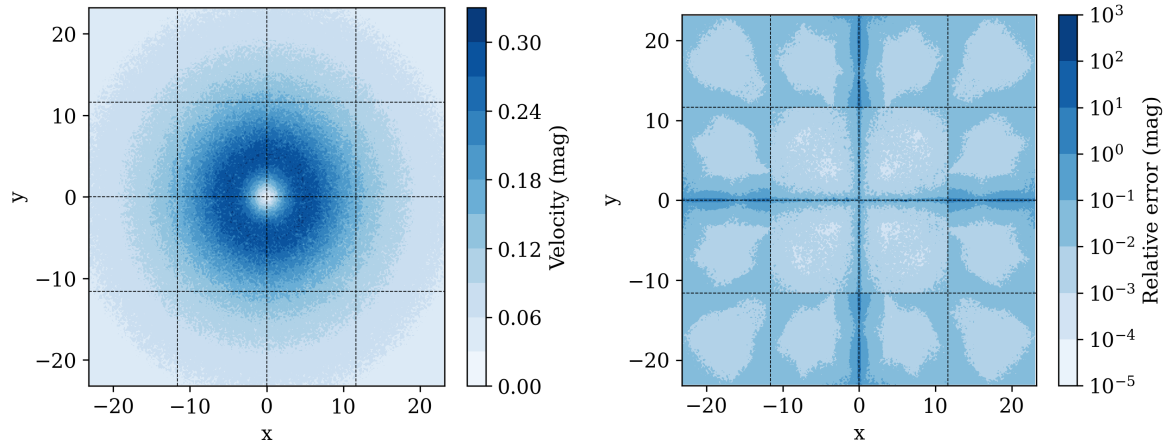
(b) Double precision

Figure 4.5: Relative error per velocity component for single and double precision FMM (Lamb-Oseen vortex, $N = 1 \times 10^5$, $d = 2$, $p = 10$, random particle distribution)

The distribution of the relative error of velocity components for a case with single and double precision in Figure 4.5a and Figure 4.5b respectively further highlights the causes of plateau in single precision accuracy. The order of magnitude of error for both cases begins slightly below the threshold of 1×10^{-4}. However, the case for double precision is able to reach accuracies of an order of magnitude 1×10^{-12}, while the single precision case stops at 1×10^{-7}. There is a clear limit of accuracy at the machine epsilon, which presents at $\epsilon_M \approx 1 \times 10^{-7}$ for single precision floating point numbers. Also noticable are relative error striations for single precision in the low error region. These were found to be lines of constant absolute error. The absence of these striations in the double precision case indicates that they are a result of the rounding errors in single precision.

Spatial Analysis of error sources

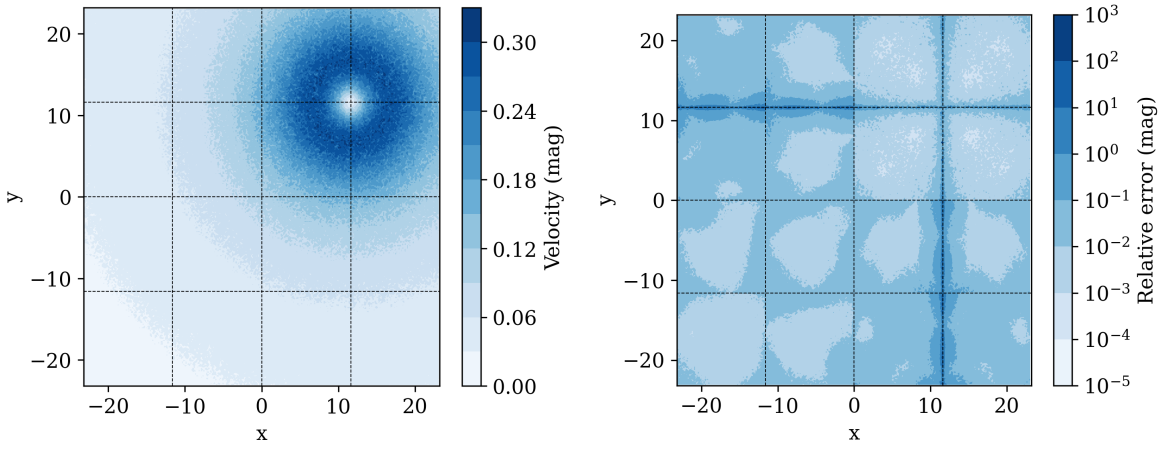
Another aspect to be clarified is whether the error patterns of FMM are mainly resulting from the flow field characteristics or are related to the shape of the hierarchical octree structure. Figure 4.6 and Figure 4.7 show the $X - Y$ plane view of the two Lamb-Oseen test cases. In Figure 4.6 the vortex core is centered in the domain, while in Figure 4.7, it is displaced from the domain center. Figure 4.6b and Figure 4.7b show the relative error distribution of each case. The edges of the clusters in the FMM octree are indicated with dotted lines.



(a) Velocity contour

(b) Relative error contour

Figure 4.6: Velocity and relative error $X - Y$ contours for a Lamb-Oseen vortex with a centered core ($p = 2$)



(a) Velocity contour

(b) Relative error contour

Figure 4.7: Velocity and relative error $X - Y$ contours for a Lamb-Oseen vortex with a displaced core ($p = 2$)

From Figure 4.6 and Figure 4.7 it was found that the errors of FMM depend both on the cluster structure of the FMM octree and on the velocity distribution in the domain. This is shown by Figure 4.7 where the error pattern follows the velocity distribution as it changes its position in the domain. However, the pattern is seen to be largely determined by the edges of the clusters in the domain. Thus it can be established that the velocity distribution has a clear role to play on the errors of FMM. One can expect higher absolute errors, and thus a higher RMSE, when there are larger velocities in the domain.

4.3.2. Lamb-Oseen Vortex with uniform particle positions

The same test case is repeated with a change in the particle distribution. In this case, uniform particle positions are used, where the particles are distributed into a 3D lattice configuration. N is selected such that the number of particles in each dimension is even. This is to ensure that there is no particle coinciding with the center of the domain, which artificially increases the error due to the singularity present there.

Assessing FMM accuracy for the uniform particle distribution was chosen as it is a common choice for initializing VPM simulations. As previously mentioned, a random particle distribution would be a more accurate representation of the simulation as it develops in time. In this way, assessing the accuracy for both distributions provides a more complete overview of the entire simulation runtime.

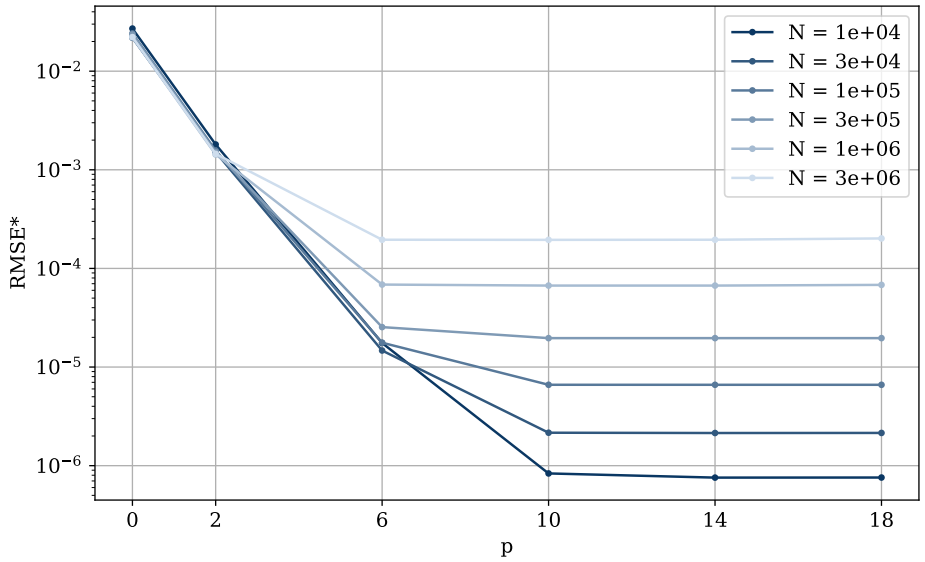


Figure 4.8: RMSE of FMM estimates with varying p and N (Lamb-Oseen vortex, $d = 2$, uniform particle distribution, constant particle density)

Figure 4.8 shows the RMSE* diagram for the uniform particle distribution. The trend of RMSE* closely resembles that of the random distribution for all numbers of particles. This leads to the conclusion that a local randomness of particle proximity does not have a significant effect on FMM accuracy.

4.3.3. Vortex ring with random particle positions

To ensure the convergence of the FMM independent of the strength distributions of the particles in the domain, another test case was investigated with a vortex ring strength distribution. Particles in a vortex ring simulation are usually not in a random distribution but rather form a toroidal structure, which is investigated in subsection 4.3.4. In this case the random particle distribution is used for comparison with the results from subsection 4.3.1

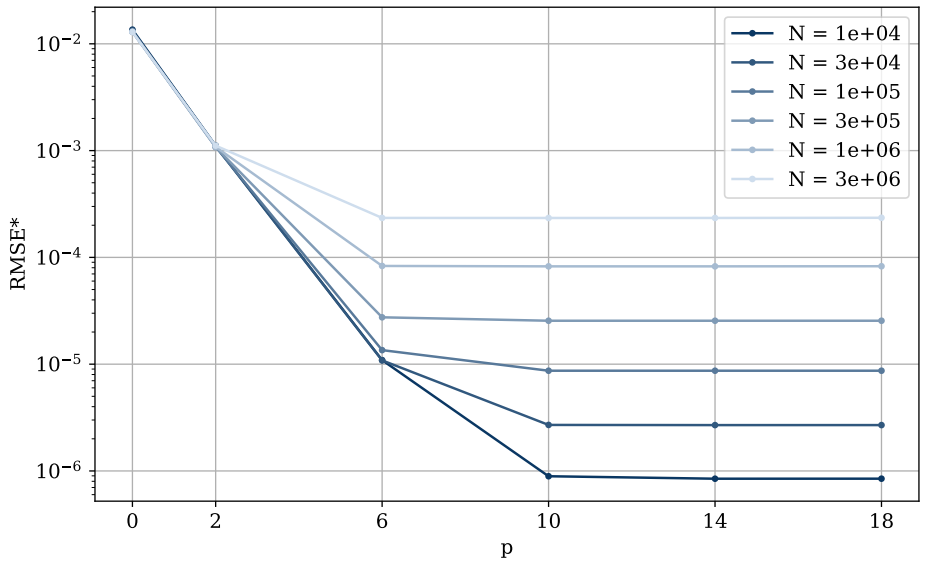


Figure 4.9: RMSE of FMM estimates with varying p and N (vortex ring, $d = 2$, random particle distribution, constant particle density)

Figure 4.9 indicates a minimal effect of the strength distribution in the domain on the RMSE*. All results show a small increase in RMSE* for the vortex ring case, but the trend of error remains the same. A potential cause is an increase of particles with a high strength in the vortex ring case compared to the Lamb-Oseen case. Since this effect is insignificant it can be assumed that strength distribution has an insignificant effect on FMM accuracy.

4.3.4. Vortex ring with particles positioned in a torus

The previous test cases contained particle distribution that populated the entirety of the cubical domain, for the rest of the discussion in this report, this is referred to as a *saturated domain*. A saturated domain results in all clusters in FMM containing a similar number of particles, and that no clusters in the domain are empty. In contrast, when simulating a flow such as a vortex ring, particles are concentrated in a toroidal structure. To investigate the accuracy of FMM in handling such sparsely populated regions, a test case of a vortex ring in a toroidal structure is also investigated. This test is different from the previous tests in that the domain and toroid size are kept the same between all numbers of particles. Instead, given that particle strengths are product of their vorticity and volume, particle volumes are adjusted to maintain a constant vorticity field.

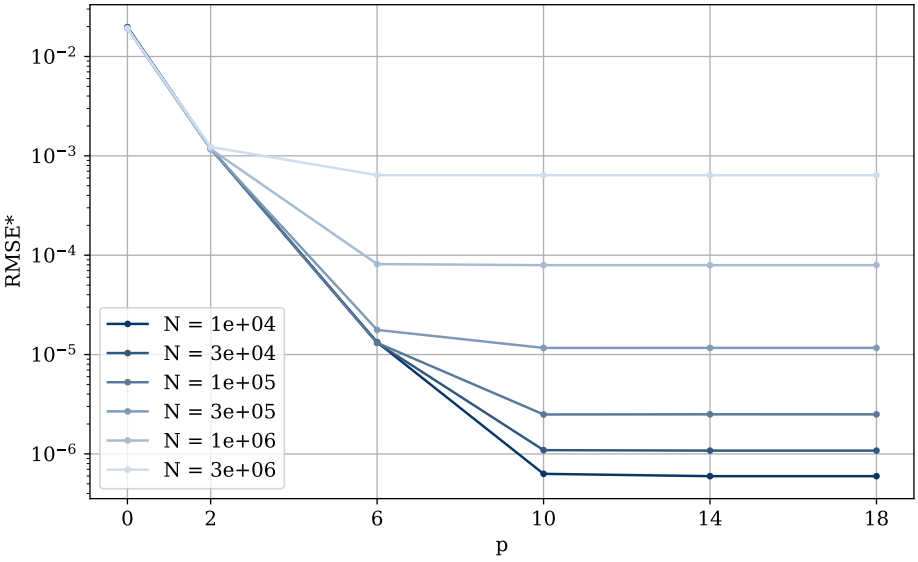
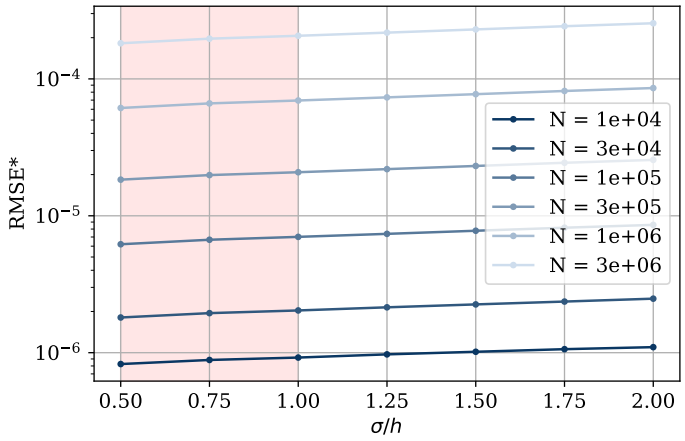


Figure 4.10: RMSE of FMM estimates with varying p and N (vortex ring, $d = 2$, toroidal particle distribution, constant domain volume)

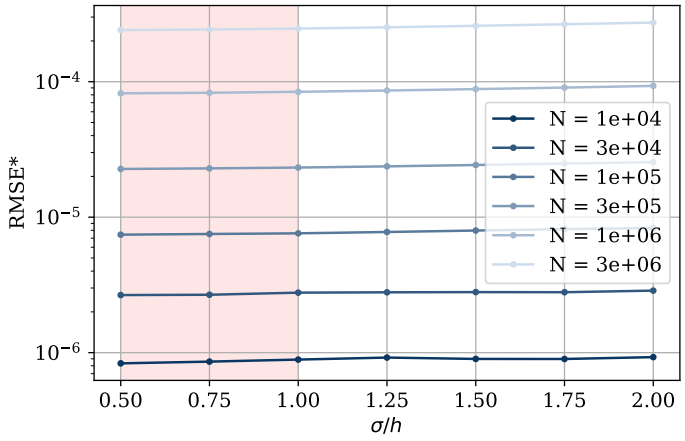
Particle torus concentration proved to significantly affect FMM accuracy in comparison to a saturated domain such as the random or uniform distribution. For low p, the toroidal particle concentrations shows similar RMSE* to the case in Figure 4.9. However, as p increases and the rounding errors begin to dominate, RMSE* becomes more variable between cases with different N. Interestingly, the results are more accurate for a small number of particles and less accurate for a large number of particles. In a torus, compared to a saturated particle field, one can expect an improved FMM accuracy because for each cluster, far field interactions are caused by the majority of particles. These particles are much further, and therefore have a smaller contribution to the induced velocity, meaning that the FMM approximation approaches that of the direct method due to the dominance of near-field interactions. The increase in RMSE* for large N might be linked to the fact that, while there are the same number of particles in the domain, in the torus situation, these particles are all concentrated in the high velocity region of the flow resulting in a higher RMSE, whereas there are many particles with large distance from the vortex core in the random distribution case.

4.4. Effect of particle overlap ratio

The overlap ratio is a parameter that can be varied by the user for different simulations. Usually, an overlap ratio in the range of 1-1.4 is selected as previously mentioned. Values of $\sigma/h < 1$ are not necessarily convergent and those much higher than 1 can be overly diffusive. For this test case a range of $0.5 \leq \sigma/h \leq 2$ is selected to investigate the accuracy of FMM for non-overlapping and heavily overlapping kernels. It is important to note that values of $\sigma/h < 1$ especially are not expected to be physically meaningful, but are included in this study purely to assess the numerical accuracy of FMM in non-overlapping situations, which could occur if some particles become spatially isolated during the simulation. For this reason, this range is highlighted in red to indicate their physical unreliability.



(a) Random particle distribution



(b) Uniform particle distribution

Figure 4.11: RMSE of FMM estimates for different overlap ratios σ/h (Lamb-Oseen vortex, $d = 2$, $p = 10$, constant particle density)

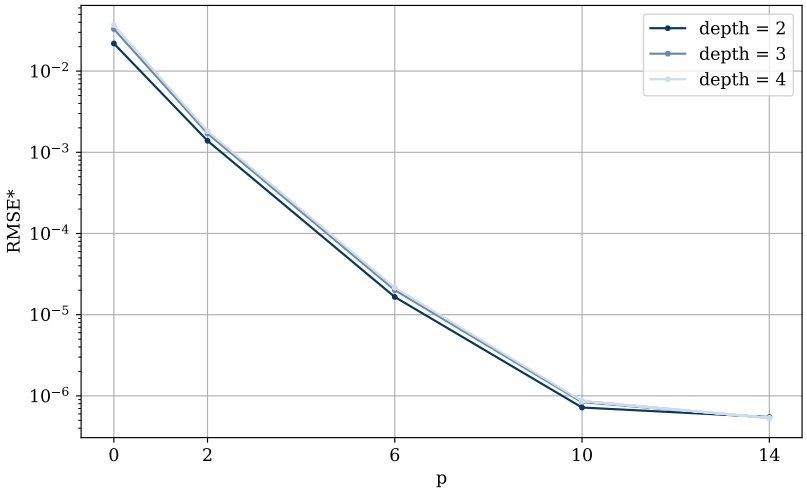
Figure 4.11a and Figure 4.11b indicate the accuracy of FMM for the range of overlap ratios investiagated. As can be seen from the figures, there is a slight decrease in accuracy as the overlap ratio is increased. Overall, since the FMM error is very small and has a less physical effect compared to the type of errors that can result from an unsuitable overlap ratio, there is no justification to readjusting the overlap ratio to achieve a higher FMM accuracy. The overlap ratio accuracy is also not significantly affected by the number of particles in the domain, thus the same overlap ratio can be used for all N.

4.5. Effect of octree depth

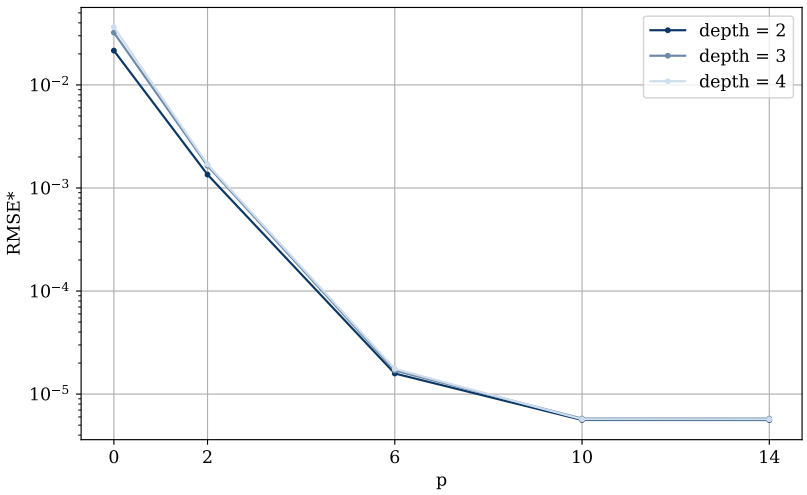
Since the octree depth d, along with the truncation order p, is one of two main FMM user settings, it is necessary to assess its impact on FMM accuracy. Additional depths of $d = 3$ and $d = 4$ are applied to the same test case in subsection 4.3.1, which uses $d = 2$ by default. The results of RMSE^* are given in Figure 4.12.

Figure 4.12 indicates a minimal effect of depth used on the accuracy of FMM. For all truncation orders p the accuracy is very similar for all depths. This also remains the same for all N. Results from Cruz and Barba [12] which implemented a similar error analysis for a 2D FMM with the Gaussian kernel, showed a decreased sensitivity of FMM accuracy to the octree depth with larger N in the domain. The results seen here are thus quite plausible, given that the results from Cruz and Barba did not show variation of error up to $d = 4$ and $p = 15$ for $N > 1 \times 10^4$.

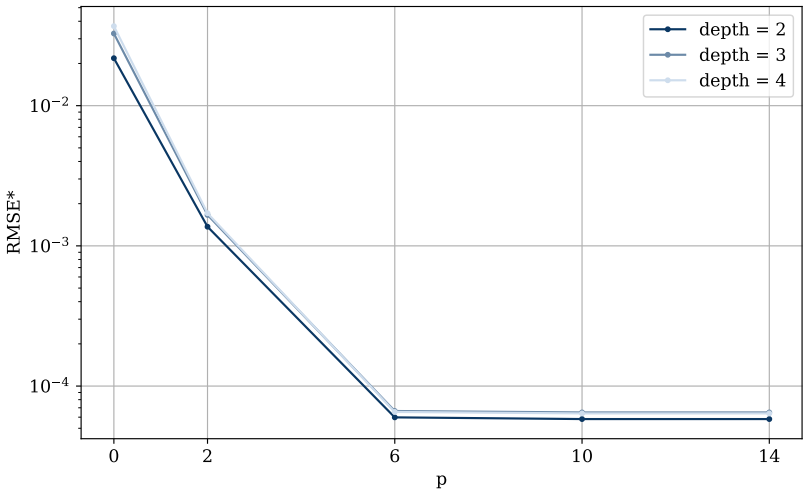
There are two potential reasons for the slight increase in error for the larger depths. Firstly, the higher the depth, the smaller the near field of each target particle in the domain. Thus, a greater fraction of the induced velocity has to be approximated using the multipole and local expansions rather than computed directly. This effect can be seen from the worse approximation of higher depths at the $p = 0$ region. This gap in accuracy is better bridged with the improvement of the FMM approximation as p increases.



(a) $N = 1 \times 10^4$



(b) $N = 1 \times 10^5$



(c) $N = 1 \times 10^6$

Figure 4.12: RMSE of FMM estimates for different octree depths d (Lamb-Oseen vortex, random particle distribution, constant particle density)

5

Validation

The validation of the FMM solver developed in this work is performed through integrating it with the OpenONDA solver. This chapter is concerned with assessing the error propagation in the simulations performed using FMM, with the goal of ensuring that the user parameters of FMM are selected such that they do not significantly affect the VPM solution. In contrast with the analysis in the previous chapter, simulations are run for multiple timesteps, involving all intermediate steps of the VPM run. This is also where an analysis of the impact of the integrated CDS in FMM for calculating the vortex stretching component becomes relevant. This analysis also relies on investigating the VPM flow diagnostics for the runs performed with FMM to ensure the physical validity of the resulting solution is maintained. The Lamb-Oseen vortex, vortex ring, and colliding vortex ring are used as test cases, giving a view of FMM's performance for a variety of flows with differing physical characteristics.

5.1. Lamb-Oseen Vortex

The first set of test cases is performed using the Lamb-Oseen vortex. The Lamb-Oseen vortex is mostly stable and can be contrasted to other flows (such as colliding vortex rings) where simulation divergence and blow-up is more likely at high Reynolds numbers. Additionally, the Lamb-Oseen vortex is a flow with smooth velocity gradients, unlike a case with colliding vortex rings. The results in this section will thus be contrasted with section 5.3 to investigate the impact of flow irregularity on FMM accuracy.

For all simulations in this chapter, the Reynolds number is defined using the vortex strength as:

$$Re_\Gamma = \frac{\Gamma}{\nu} \, . \tag{5.1}$$

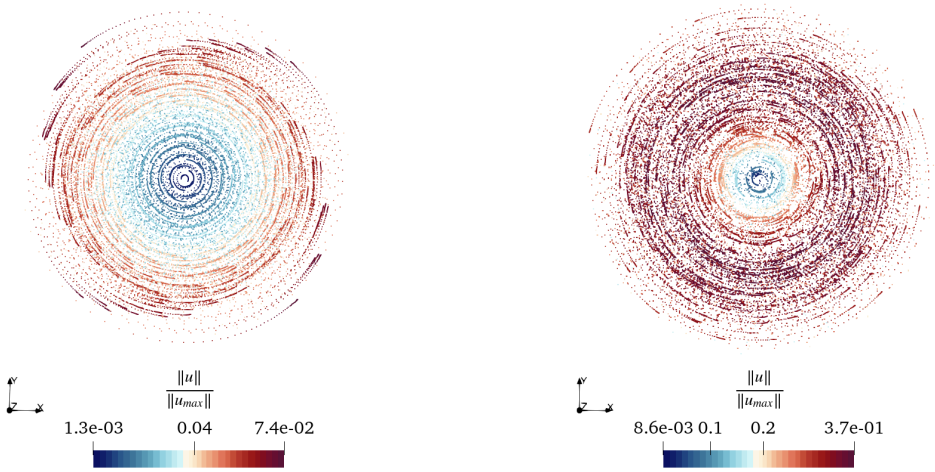
Additional settings used in the Lamb-Oseen VPM simulation are provided in Table 5.1.

Table 5.1: VPM settings for the Lamb-Oseen cases

Parameter	Value
Number of particles N	5×10^4
spacing h	0.2
Initial vortex time t_0	30 [s]
vortex strength Γ	π
Kinematic viscosity ν	Re/Γ
Timestep size Δt	0.25
Overlap ratio σ/h	$0.8\sqrt{h}$
Domain dimensions	$4 \times 4 \times 20$
Viscous diffusion method	CSM

5.1.1. Accuracy of FMM estimates

A change in the Reynolds number can have an effect on the accuracy of FMM. Flows with high Reynolds numbers undergo a less significant viscous diffusion, leading to a more turbulent flow field. With this there are stronger velocity gradients that need to be accurately captured by FMM. This test case involves two Lamb-Oseen vortices with the parameters in Table 5.1, but with $Re = 100$ and $Re = 1000$. The time is normalised by the diffusion time $t_\nu = R^2/4\nu$, where R is the characteristic Lamb-Oseen vortex radius defined as $R = \sqrt{4\nu t_0}$.



(a) $Re = 100$ **(b)** $Re = 1000$

Figure 5.1: Top view of Lamb-Oseen vortex at $t/t_\nu = 6.67$ for differing Reynolds numbers

As seen in the comparison in Figure 5.1, the case with $Re = 1000$ appears more turbulent due to the lower effect of viscosity. Furthermore, particle velocities in this case are higher due to the lower frictional forces caused by viscosity, as can also be deduced from Equation 4.9. A comparison of the performance of FMM in both of these cases provides an insight into FMM's sensitivity to flow turbulence.

In this test, some parameters of FMM are varied to investigate the solution error and the convergence to diagnostics. This involves FMM with and without the computation of velocity gradients using CDS. FMM runs are performed with $p = 2$ and $p = 10$, both with a depth of $d = 2$. The impact of variables such as the CDS step size δ, the convergence order of the CDS used $\mathcal{O}(\delta)$, are also studied. The accuracy of velocity gradient calculation for CDS is later discussed for the $Re = 1000$ case in subsection 5.1.3.

Results for $\mathbf{Re = 100}$

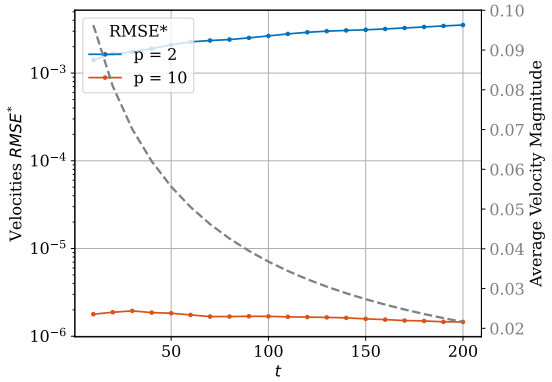


Figure 5.2: Velocities RMSE* for $Re = 100$ Lamb-Oseen VPM simulation

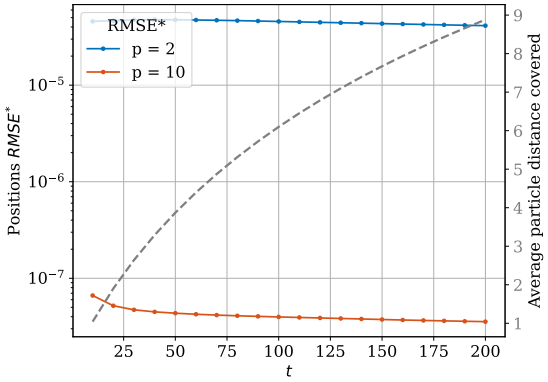


Figure 5.3: Positions RMSE* for $Re = 100$ Lamb-Oseen VPM simulation

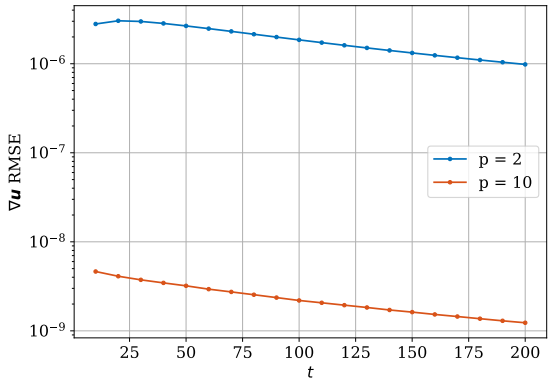


Figure 5.4: Velocity gradients RMSE for $Re = 100$ Lamb-Oseen VPM simulation

The velocity accuracy at the first timestep is consistent with the RMSE* order of magnitude seen in Figure 4.3. For the $p = 2$ case, this error grows with time. This is caused by the accumulation of FMM errors due to the errors in estimating the particle positions in successive timesteps. This in turn feeds into a different strength distribution resulting from the effect of vortex stretching in Equation 2.3, exacerbating the FMM error for the subsequent timestep. For the $p = 10$ case however, RMSE* appears to remain constant. In this case the approximation of the velocity is most likely sufficiently accurate as to not significantly affect subsequent timesteps. Simulations with FMM can be continued indefinitely for the $p = 10$ case without significant accumulation of FMM error.

Figure 5.3 and Figure 5.4 show the RMSE* of the particle positions and velocity gradients computed using the direct method. The positions RMSE* reduces as the simulation evolves for both the $p = 2$ and $p = 10$ cases. The results indicate that FMM error is overall convergent for this test case.

Results for $\mathbf{Re = 1000}$

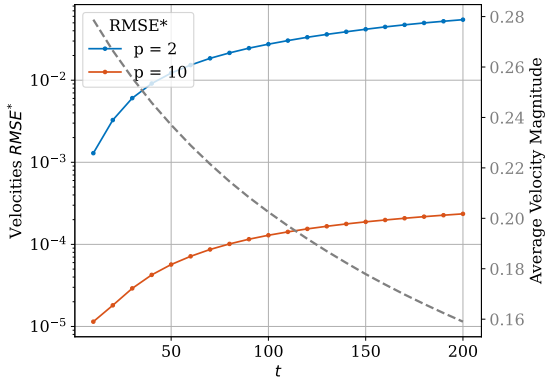


Figure 5.5: Velocities RMSE* for $Re = 1000$ Lamb-Oseen VPM simulation

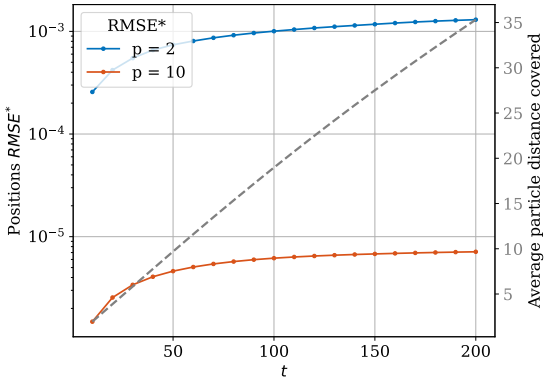


Figure 5.6: Positions RMSE* for $Re = 1000$ Lamb-Oseen VPM simulation

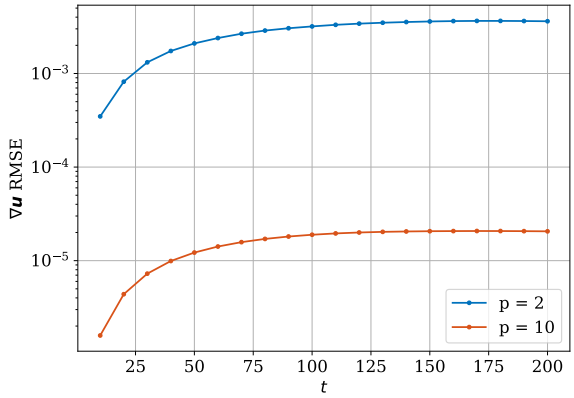


Figure 5.7: Velocity gradients RMSE for $Re = 1000$ Lamb-Oseen VPM simulation

For the $Re = 1000$, the results differ significantly. Velocity RMSE* starts at the same value as the $Re = 100$ case, but subsequently increases significantly. The increase of RMSE* for $p = 2$ is slightly less than two orders of magnitude, approaching 1×10^{-1}, which presents a high error. The rate of error growth is slightly lower for the $p = 10$ case. Furthermore, both simulations seem to plateau at the later timesteps of the simulation. There is a reduction in average velocity magnitude plot, which indicates that the FMM velocity RMSE* has an inverse relationship with the induced velocities. The diminishing increase in RMSE* indicates a stability of the FMM solution with higher timesteps, although it becomes advisable to use a larger p for longer simulations to avoid excessive growth of FMM velocity error. Similar to the $Re = 100$ case, the positions RMSE* and velocity gradient RMSE are more convergent than the velocities RMSE* , presenting a lesser concern to the final solution error.

From this analysis, we can deduce that the Reynolds number is the main reason for the discrepancy in error between both Lamb-Oseen cases. In the $Re = 100$ case, the particle velocities are quickly damped by the viscosity as it reduces by one order of magnitude. The reduction in velocity offsets the accumulation of FMM error. In contrast, in the $Re = 1000$ case the velocity halves, and the accumulation of preceding FMM error overcomes the reduction in velocity.

Since the case with the higher Reynolds number is less convergent, its analysis is of greater interest to the error stability of FMM. The analysis for the diagnostics in subsection 5.1.2 and CDS scheme in subsection 5.1.3 is therefore performed for the $Re = 1000$ case.

5.1.2. Conformity with Diagnostics

With physical simulations, an accurate prediction of the velocity field is necessary but insufficient. It is also necessary for the FMM-calculated field to adhere to the flow diagnostics set out for VPM simulations. Figure 5.8 shows the evolution of key flow diagnostics throughout the simulation. The first diagnostic is the kinetic energy decay rate dE/dt and the product $-\nu\mathcal{E}$, which demonstrates the extent to which turbulence is resolved in the simulation. The total circulation, linear impulse and angular impulse are also shown, which should be conserved.

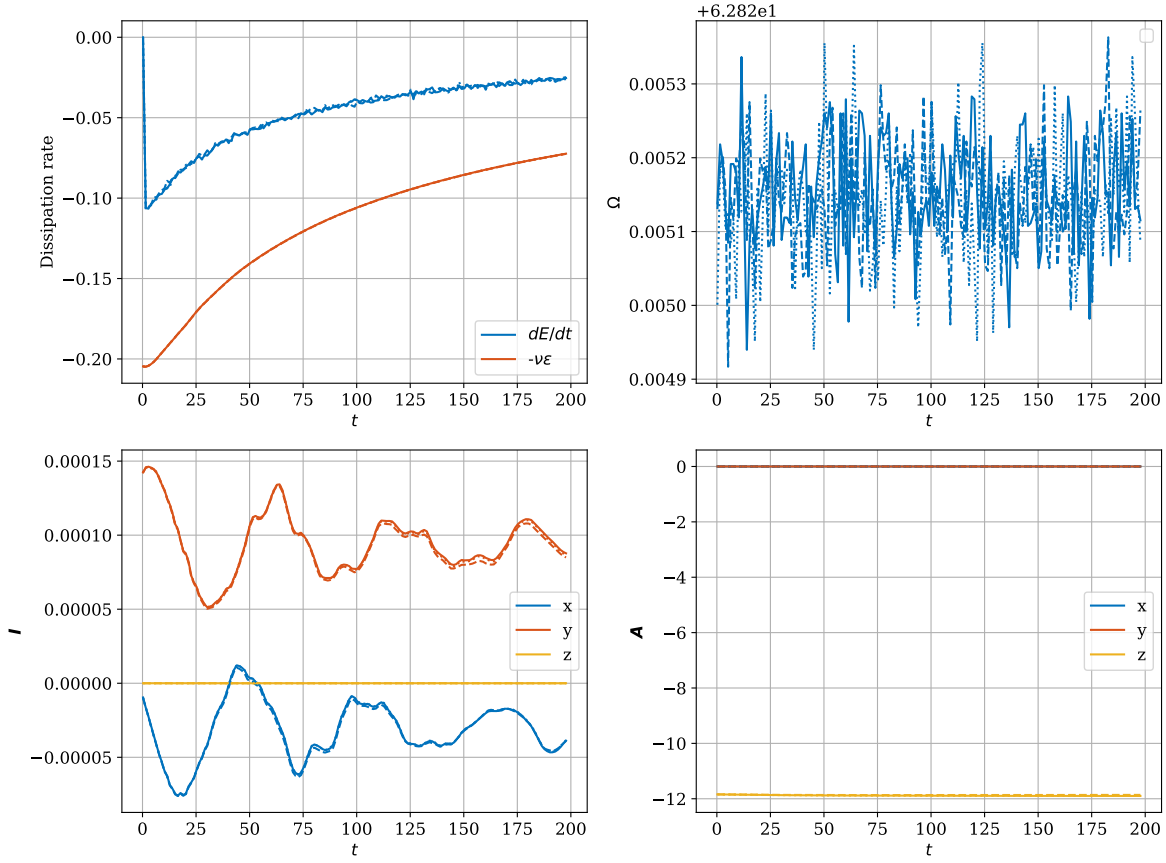


Figure 5.8: VPM diagnostics for $Re = 1000$ Lamb-Oseen test case (**Clockwise from top left:** measured kinetic energy decay rate contrasted to viscosity-enstrophy decay rate, total circulation, angular impulse, linear impulse)
Legend: ——— Direct - - - - FMM $[p = 2]$ ····· FMM $[p = 10]$

The kinetic energy decay rate indeed indicates an underresolution of the turbulence scales. One can see that the measured rate of energy decay is almost half that of the analytical estimate given by $-\nu\mathcal{E}$. Furthermore, the total circulation and angular impulse are indeed conserved. The measured circulation in this case is a constant $\Omega \approx 2\pi$, owing to the fact that the Lamb-Oseen vortex has nonzero circulation at infinity, since all particle strength vectors point in the same direction, whereas in the vortex ring case they cancel each other. The linear and angular impulse are also conserved. Oscillations are apparent for $\mathbf{I}_x$ and $\mathbf{I}_y$ around a small constant value. The oscillations are damped by viscous effects and are most likely a product of spatial inconsistencies occurring during the rotation of the Lamb-Oseen vortex. Most importantly, the FMM estimates display close overlap with the direct method for both the $p = 2$ and $p = 10$ cases for all diagnostics during the entirety of the simulation. It can therefore be deduced that FMM has a minimal effect on the physical validity of such simulations.

5.1.3. Accuracy of velocity gradients using CDS scheme

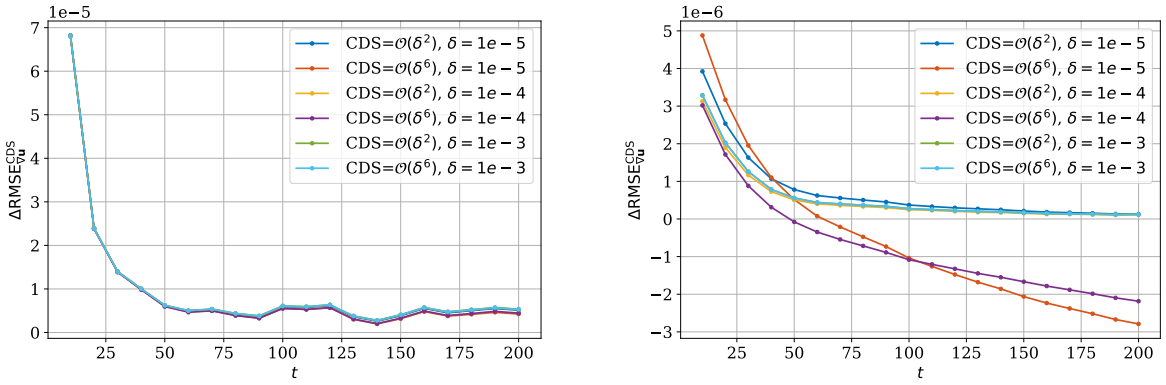
The sources of error for the implemented CDS scheme for calculating the velocity gradients in FMM are twofold. First, there is the error arising indirectly from the inaccuracy of FMM itself. This is a cumulative effect arising from the difference in particle positions, velocities, radii and strengths from FMM estimation to the direct calculation. This results in errors in the gradient calculations. The second source of error, and the one for which the CDS scheme shall be assessed, arises from the CDS scheme itself. As with all numerical central-difference schemes, there are truncation and rounding errors involved. We can express the CDS error as:

$$\eta_{\text{CDS}} = \eta_{\text{CDS}}^{\text{FMM}} + \eta_{\text{CDS}}^{\text{self}} \ . \tag{5.2}$$

Based on this we can define a change in the RMSE due to the implementation of CDS:

$$\Delta\text{RMSE}_{\nabla\mathbf{u}}^{\text{CDS}} = \text{RMSE}_{\nabla\mathbf{u}}^{\text{FMM+CDS}} - \text{RMSE}_{\nabla\mathbf{u}}^{\text{FMM}} \ . \tag{5.3}$$

For example, the RMSE of the velocity gradients where FMM is used to calculate the velocities, while their gradients are calculated using the direct method (e.g. $p = 2$, no CDS) can be subtracted from one where both the velocities and their gradients are calculated with FMM (e.g. $p = 2$, CDS with $\mathcal{O}(\delta^2)$, $\delta = 1 \times 10^{-5}$) to obtain the overall change to RMSE caused by the CDS.



(a) $p = 2$ **(b)** $p = 10$

Figure 5.9: $\Delta\text{RMSE}_{\nabla\mathbf{u}}^{\text{CDS}}$ for $Re = 1000$ Lamb-Oseen VPM simulation

Figure 5.9a and Figure 5.9b show this change for different CDS schemes for the Lamb-Oseen test case. The change in RMSE is seen to be in the order of 1×10^{-5} for $p = 2$, about two orders of magnitude less than the values in Figure 5.7. The accuracy of the CDS scheme for $p = 2$ can be considered sufficient. The accuracy of the CDS scheme is seen to be convergent but is not highly sensitive to the step size or the order of convergence for the $p = 2$ case. In contrast, the $p = 10$ case CDS schemes show high variance between them. CDS schemes with $\mathcal{O}(\delta^6)$ performed equal or better than $\mathcal{O}(\delta^2)$ for all step sizes. Increasing the step size also showed better accuracy for $p = 10$. This is indicative that the solution field is sufficiently smooth such that a larger step size can be employed to compute gradients accurately, and contrasts with the significant effect of round-off errors caused by the float precision used for CDS.

5.2. Vortex Ring

Due to the varied results obtained for the Lamb-Oseen vortex and the vortex ring in chapter 4, validation is also performed for the vortex ring particles within a torus. The torus initialization is obtained by implementing a hexagonal particle initialisation as in the Lamb-Oseen case. Particles lower than a percentage strength of the maximum particle strength in the domain are omitted, leaving behind the

strongest particles, which form a torus around the vortex core. The settings used are provided in Table 5.2.

Table 5.2: VPM settings for the vortex ring cases

Parameter	Value
Number of particles N	5×10^4
spacing h	0.1
vortex strength Γ	π
Reynolds number Re	1000
Timestep size Δt	0.25
Overlap ratio σ/h	$0.8\sqrt{h}$
Domain dimensions	$4 \times 4 \times 4$
Viscous diffusion method	CSM

5.2.1. Accuracy of FMM estimates

Figure 5.10 and Figure 5.11 show the RMSE* of the particle velocities and positions respectively for the vortex ring case at $Re = 1000$.

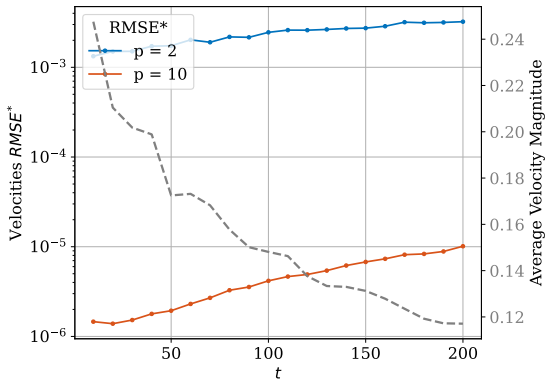


Figure 5.10: Velocities RMSE* for $Re = 1000$ vortex ring VPM simulation

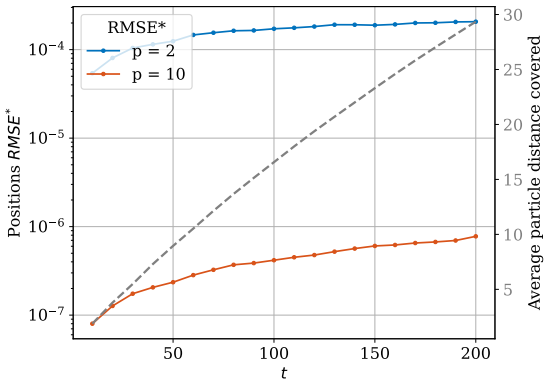


Figure 5.11: Positions RMSE* for $Re = 1000$ vortex ring VPM simulation

The evolution of RMSE* for the vortex ring case was found to be similar to that of the $Re = 1000$ Lamb-Oseen case. Again, the RMSE* estimates at the start of the simulations are consistent with those of Figure 4.10. Interestingly, the increase of RMSE* in Figure 5.10 is less pronounced than that of the Lamb-Oseen case in Figure 5.5. A possible reason for this being that the near field has a more dominant effect in the vortex ring case, as most particles in the far field are further positioned from the target particles, due to the hollow section in the center of the ring. A less dominant effect of FMM estimated far field interactions could lead to significantly lower errors in particle velocities and positions. The average velocity magnitude in Figure 5.10 can be seen to follow a rugged structure. The reason found for this was a regular exchange in particle concentration at different radii from the vortex core throughout the simulation, which affected the average induced particle velocity.

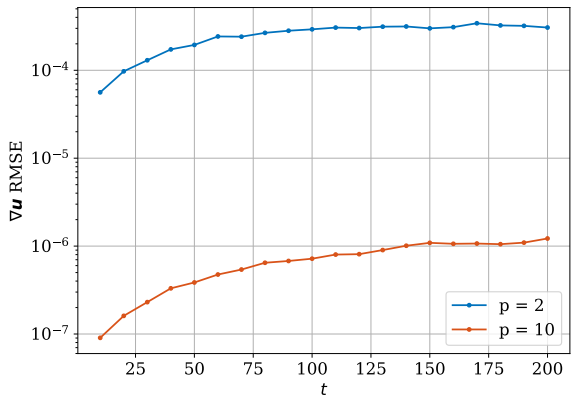


Figure 5.12: Velocity gradients RMSE for $Re = 1000$ vortex ring VPM simulation

The velocity gradients without the use of CDS shown in Figure 5.12 also indicate convergence as the simulation time increases. It is interesting to note that the RMSE errors in Figure 5.12 are orders of magnitude lower than those of the Lamb-Oseen vortex case in Figure 5.7, most likely due to the improved accuracy of the velocity and position estimates compared to the Lamb-Oseen case.

These results indicate that the FMM implementation is successful in regenerating the octree structure for each timestep, which becomes a necessity for this test case as the vortex ring advects through space. Additionally, the variation of the number of particles in each cluster was not shown to pose a negative effect on how the FMM accuracy evolves in these cases.

5.2.2. Conformity with diagnostics

The key flow diagnostics are shown for the vortex ring test case in Figure 5.13. Figure 5.13 shows an under-resolved simulation with the a similar dissipation trend between the measured kinetic energy decay and the $-\nu\mathcal{E}$ estimate. The total vorticity in the flow is conserved throughout the simulation. Unlike the Lamb-Oseen case, the measured total vorticity in this case is close to zero. As a self-inclosed vortex line forms the vortex ring core, where the vorticity of the particles decays with distance from the core, the total circulation at infinity is zero. For the linear impulse, $\mathbf{I_y} = 0$ and $\mathbf{I_z} = 0$, while $\mathbf{I_x}$ is non-zero, which results from the vortex ring advection in the positive x direction. Lastly, the angular impulse shows a non-conserved $\mathbf{A_z}$ component. This could be linked to the small value of vorticity present in the total vorticity plot and might be explained by imperfections in the initialization method. Due to the hexagonal structure of the particle initialization, the removal of the weakest particles might not result in a perfectly axisymmetrical system, which could lead to the total vorticities not perfectly cancelling out to zero. Overall, both $p = 2$ and $p = 10$ FMM simulations again show good convergence to the direct calculated values for all diagnostics.

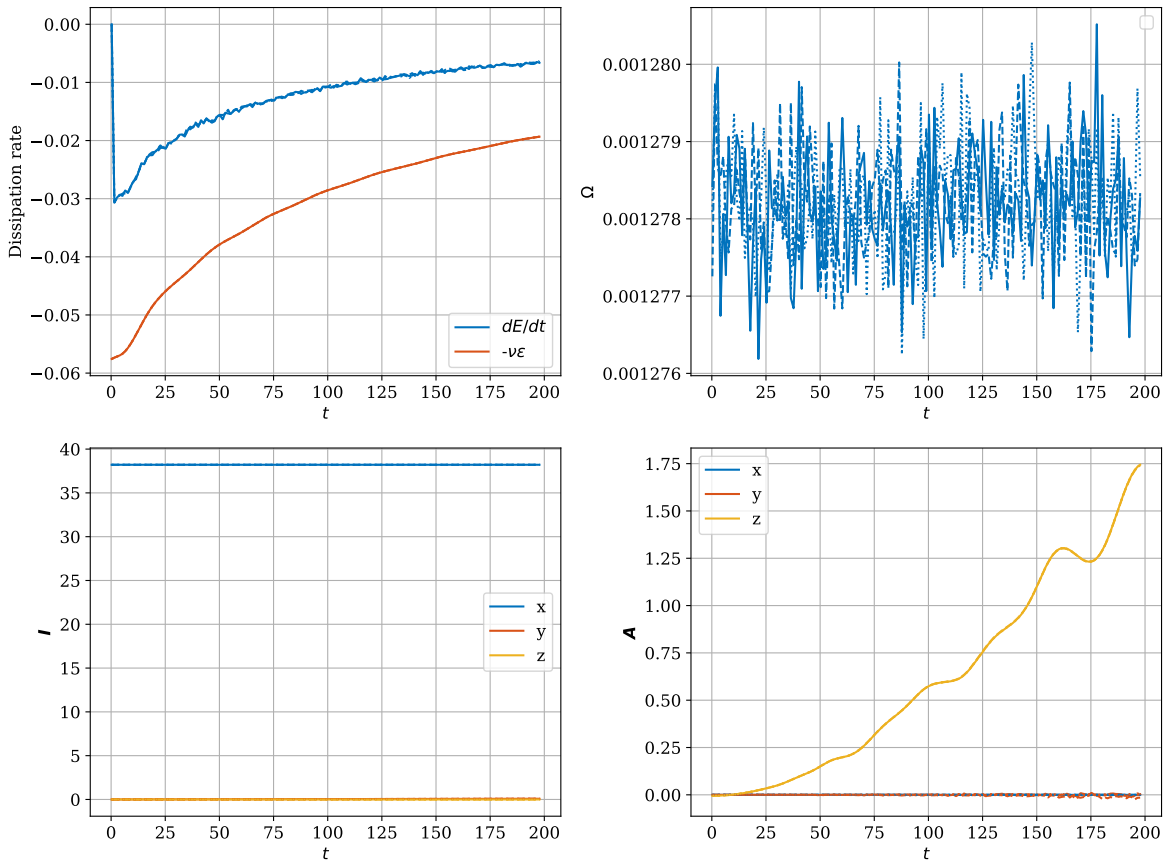
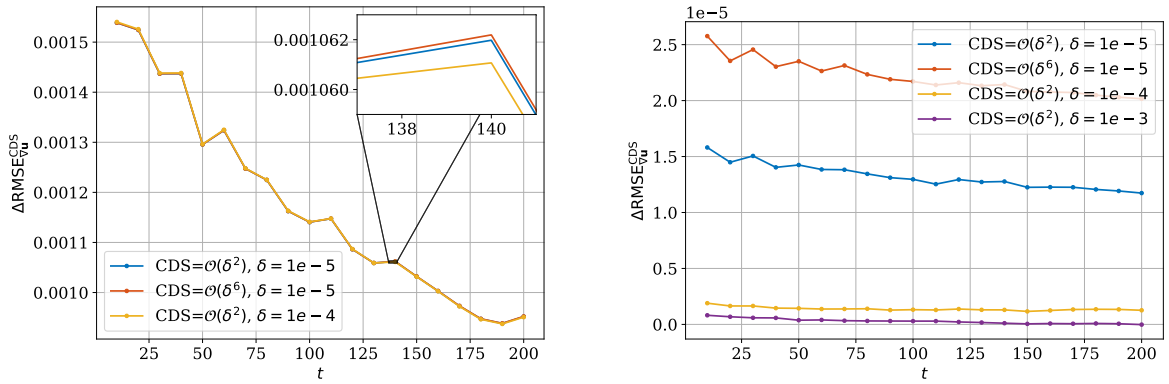


Figure 5.13: VPM diagnostics for $Re = 1000$ vortex ring test case (**Clockwise from top left:** measured kinetic energy decay rate contrasted to viscosity-enstrophy decay rate, total circulation, angular impulse, linear impulse)
Legend: —— Direct - - - - FMM $[p = 2]$ ····· FMM $[p = 10]$

5.2.3. Accuracy of velocity gradients using CDS scheme



(a) $p = 2$ **(b)** $p = 10$

Figure 5.14: $\Delta\text{RMSE}^{\text{CDS}}_{\nabla \mathbf{u}}$ for $Re = 1000$ vortex ring VPM simulation

Figure 5.14 shows the CDS errors for $p = 2$ and $p = 10$ for the vortex ring case. Unlike the FMM estimates, which improve compared to the Lamb-Oseen case, the CDS scheme calculation of velocity gradients is less accurate than the results in subsection 5.1.3. The CDS errors in for $p = 2$ in Figure 5.14a

are convergent but show low sensitivity to variation in CDS parameters, the same pattern seen for the Lamb-Oseen case in Figure 5.9a. Based on this, it can be inferred that the choice of CDS scheme has minimal effect on accuracy when p is small, as the FMM errors themselves are much more significant. For the $p = 10$ case, the CDS errors are also convergent. Like the Lamb-Oseen case, Figure 5.14b shows that a larger step size resulted in a smaller CDS error.

5.3. Colliding Vortex Rings

Underresolved DNS simulations are run for colliding vortex rings. This is a test case where two vortex rings on the same axis propagate towards each other and collide. Both rings, lying on the same axis, are given the same strength and propagate with the same velocity towards each other. Theoretically, since the problem is symmetric along the $X = 0$ plane, neither vortex ring should overcome the other, and the plane of symmetry should appear similar to a wall between the two rings as they collide. The rings used are of the same dimensions as those in the vortex ring test case and the settings used for this case are provided in Table 5.3.

Table 5.3: VPM settings for the colliding vortex rings cases

Parameter	Value
Number of particles N	5×10^4
spacing h	0.2
vortex strength Γ	π
Reynolds number Re	1000
Timestep size Δt	0.25
Overlap ratio σ/h	$0.8\sqrt{h}$
Domain dimensions	$12 \times 12 \times 12$
Viscous diffusion method	CSM

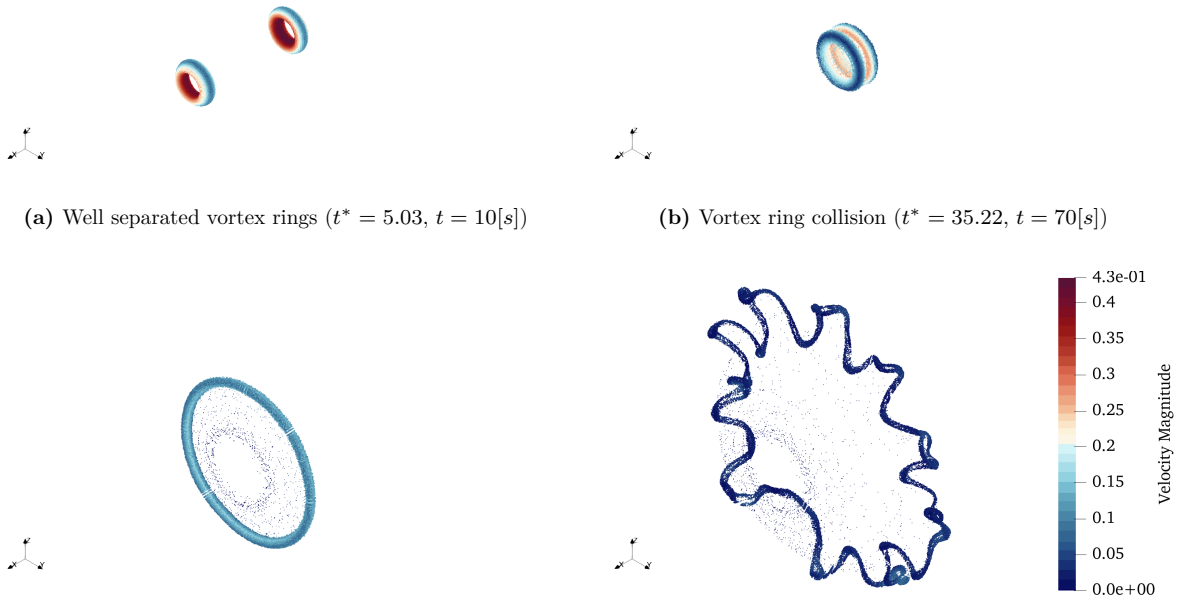
The colliding vortex ring case highlights the impact of FMM while the rings are spaced, and therefore predominantly rely on FMM far field computations for the interactions between them. The velocity field at the end of a vortex ring collision is very sensitive to disturbances, contains strong velocity gradients and often display irregular secondary flow structures. It is in question whether an FMM estimated velocity field will accurately exhibit the same secondary flow structures as those computed directly. Figure 5.15 shows an example of the evolution of the velocity field of colliding vortex rings at $Re = 1000$. For the subsequent description we define non-dimensionalised terms for the ring translational velocities and the time. A non-dimensionalised time is defined for these stages of the simulation using:

$$t^* = \frac{t\,\Gamma}{R^2}\ . \tag{5.4}$$

And the non-dimensionalised velocity is defined as:

$$u^* = \frac{u\,R}{\Gamma}\ . \tag{5.5}$$

This begins with the rings centred around coordinates of $[4R, 0, 0]$ and $[-4R, 0, 0]$ convecting towards each other at a translation speed of $u^* \approx 0.113$, which is illustrated in Figure 5.15a. As the rings collide, their translation motion towards each other approaches zero and both rings stretch uniformly in the $X - Y$ plane with an initial speed of $u^* \approx 0.119$ as shown in Figure 5.15b and Figure 5.15c. The flow after the collision is initially mostly coherent and transitions into a turbulent flow after some time. This vortex breakdown is shown in Figure 5.15d.



(a) Well separated vortex rings ($t^* = 5.03$, $t = 10[s]$)

(b) Vortex ring collision ($t^* = 35.22$, $t = 70[s]$)

(c) Vortex ring spreading ($t^* = 65.42$, $t = 130[s]$)

(d) Vortex ring breakdown ($t^* = 150.96$, $t = 300[s]$)

Figure 5.15: Evolution of colliding vortex rings at different simulation phases

Despite both vortex rings having the same strength, and theoretically colliding symmetrically, in reality one of the vortex rings eventually overcame the other, and the vortex ring collision propagated slightly in in the positive x direction. This was attributed to small differences in the initialization of the two vortex rings. Given that they both were initialised with a hexagonal structure and filtered by particle strength, small differences in the positions of the particles in both rings lead to one ring being slightly stronger than the other.

5.3.1. Accuracy of FMM estimates

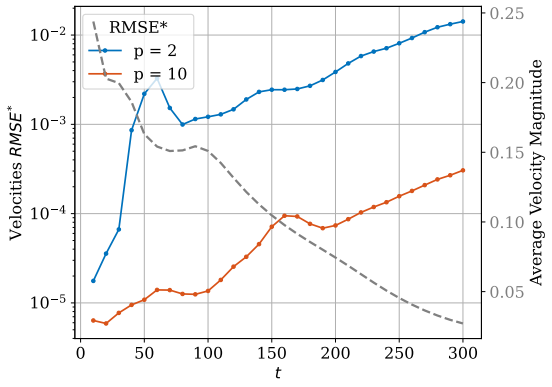


Figure 5.16: Velocities RMSE* for $Re = 1000$ colliding vortex rings VPM simulation

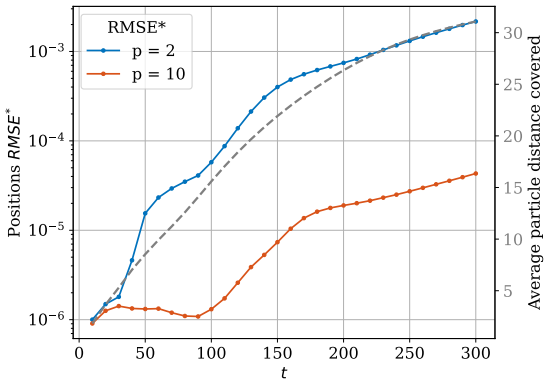


Figure 5.17: Positions RMSE* for $Re = 1000$ colliding vortex rings VPM simulation

From the results in Figure 5.16 it is evident that the largest increase in error does not occur when the two vortices are closest during the collision, which occurs at around $t = 70[s]$. Rather, it occurs

slightly earlier at $t = 60[s]$, where the far field interactions become stronger due to the proximity of the rings. Thus, the result could be mostly attributable to the far field component being significantly more dominant for $t = 60[s]$. During the collision itself, much more particles exist in each other's near field, which is handled through direct interactions and thus the accuracy improves. The reduction of RMSE* for the $p = 2$ case after $t = 60[s]$ is more pronounced than for the $p = 10$, highlighting the ability of higher order Taylor approximation in bridging the gap between far field and near field calculations.

The velocities and positions RMSE* in the late stages of the simulation does not plateau, contrary to the behaviour in the Lamb-Oseen and vortex ring cases, despite the reduction in average velocity magnitude. The FMM estimate continues to worsen in the breakdown regime. This is believed to be a result of the unstable chaotic nature of the breakdown regime, where small disturbances in velocity result in significant changes in the shapes of the ensuing secondary flow structures. Small velocity errors from FMM are expected to contribute to this, thereby increasing the error between it and the direct method.

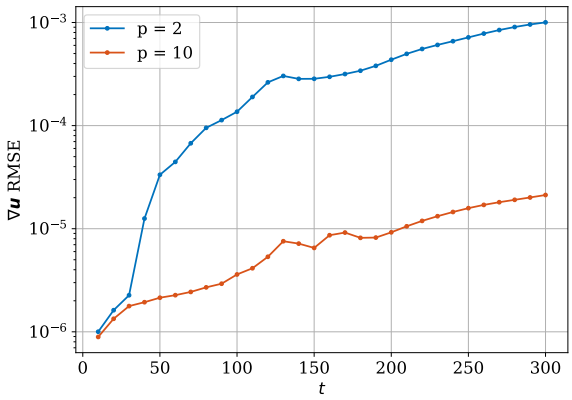


Figure 5.18: Velocity gradients RMSE for $Re = 1000$ colliding vortex rings VPM simulation

For the velocity gradients in Figure 5.18, the $p = 2$ and $p = 10$ FMM estimates are similar at the initial phase of the simulation ($t \leq 30$), when the influence between the vortex rings is insignificant. As the rings approach each other and their influence on each other becomes more significant through the FMM far field, a discrepancy between the two p values arises as the $p = 2$ error grows rapidly. This is then later stabilized as the vortex rings merge into each other, causing their most significant interactions to be directly computed through the near field. It was expected for the error in the velocity gradients to increase significantly as the flow becomes chaotic. The opposite was observed as the error rises more steadily after $t = 70[s]$. A possible explanation for this finding is the increased distance between particles in the far field as the ring spreads. The induced velocities from particles in the far field are thus reducing, resulting in a lower FMM error. Furthermore, as the induced velocities have an inverse square relationship with the distance between source and target, the velocity gradients have a higher order inverse relationship, resulting in a rapid decay of the velocity gradient influence from the far field.

5.3.2. Conformity with Diagnostics

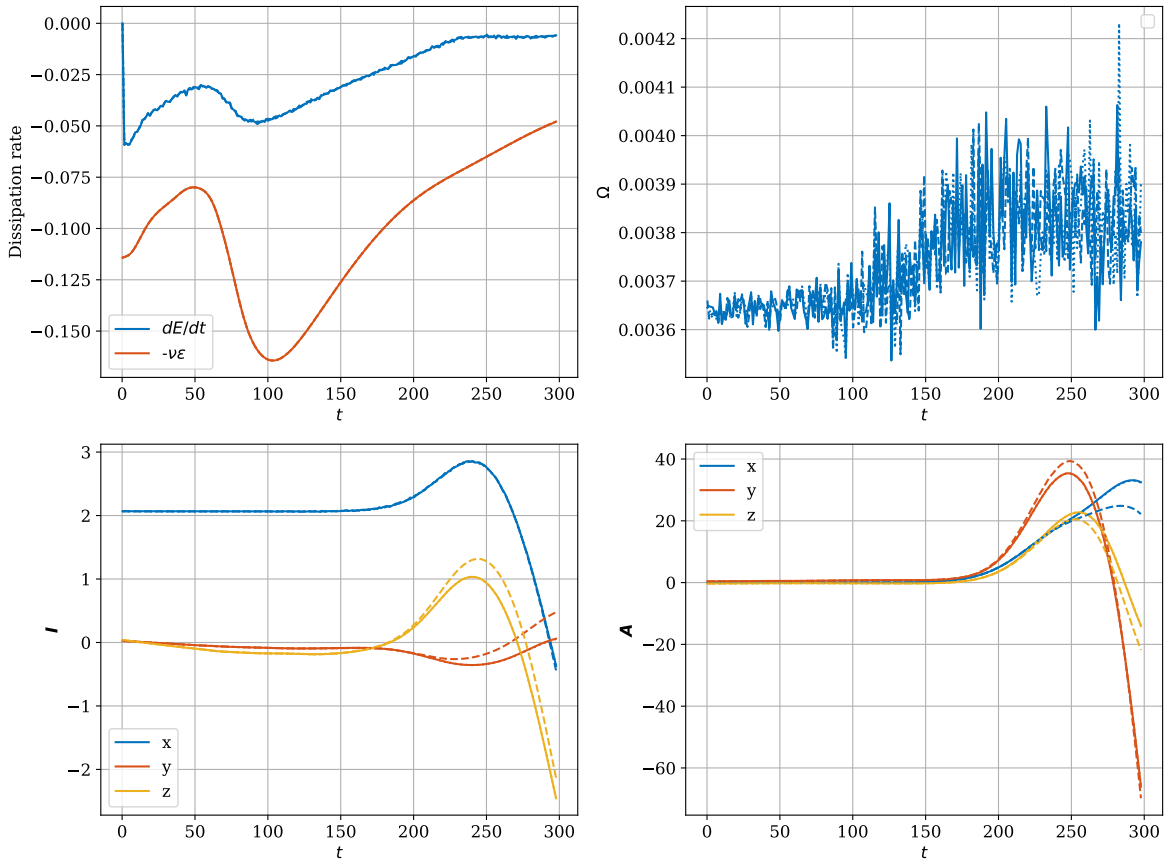


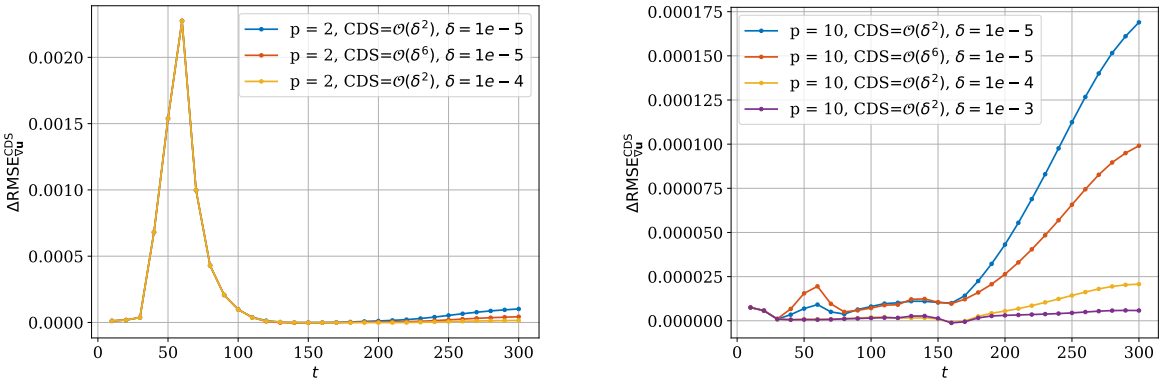
Figure 5.19: VPM diagnostics for $Re = 1000$ colliding vortex rings test case (**Clockwise from top left:** measured kinetic energy decay rate contrasted to viscosity-enstrophy decay rate, total circulation, angular impulse, linear impulse)
Legend: —— Direct - - - - FMM $[p = 2]$ ····· FMM $[p = 10]$

Figure 5.19 shows the diagnostics for the colliding vortex rings test case. The simulation is underresolved as evidenced by the discrepancy between the dE/dt and $-\nu\mathcal{E}$ values. The circulation remains conserved but rises slightly at the $t > 70[s]$ mark, as the ring begins to develop secondary structures.

Initially, there is a small positive component of $\mathbf{I_x} = 2$, contrasted to $\mathbf{I_x} \approx 40$ for the single vortex ring case, which supports the observation that the colliding rings propagate slightly in the positive x direction, as the ring impulses do not fully cancel.

The distinguishing feature of the diagnostics in this test case, is the divergence of the linear and angular impulse components in the vortex ring breakdown regime after $t = 70[s]$. This is attributed to the reduced physical meaningfulness of the VPM results due to the insufficient energy dissipation in the flow. Despite this divergence in diagnostics, the FMM estimate remains close to the direct method calculated results. The FMM with $p = 2$ diverges slightly from the direct method once the flow becomes chaotic, whereas the $p = 10$ case continues to stay close to the direct method throughout the simulation. This highlights the stability of using FMM in maintaining the physical meaningfulness of the flow. Firstly, FMM did not affect the time at which the flow transitioned into the chaotic regime, and kept a relatively similar flow field to the direct method, despite the chaotic flow nature.

5.3.3. Accuracy of velocity gradients using CDS scheme



(a) $p = 2$ **(b)** $p = 10$

Figure 5.20: $\Delta\text{RMSE}^{\text{CDS}}_{\nabla\mathbf{u}}$ for $Re = 1000$ colliding vortex rings VPM simulation

Lastly, the velocity gradient accuracy using CDS is shown in Figure 5.20. A peak is observed at $t \approx 60[s]$ when the vortex rings influence on each other is significant. This peak can also be seen for the velocity estimates of FMM in Figure 5.16. The peak indicates lowered accuracy of the CDS scheme in face of strong velocity gradients. As the rings approach each other, one might expect the far field gradients caused by a ring on another to have a larger local variation, which could be the cause of this inaccuracy. The $p = 10$ case better attenuates this peak by accounting for the local variation in velocity gradients better than $p = 2$. Finally, the velocity gradients can be seen to diverge as the ring transitions into chaotic secondary structures. This can be explained by the rapid change of velocities of the particles as the structures develop. However, due to the non-deterministic nature of these chaotic structures, estimate divergence in this case can be considered somewhat acceptable. A CDS scheme with a larger δ performs better in this case as well. Based on this, a step size $\delta \approx 1 \times 10^{-3}$ can be considered a good value to use for the CDS, regardless of the flow in consideration.

6

Performance Analysis

This chapter provides an overview of the expected performance characteristics of the FMM operations when run settings are varied. Implementing an FMM run involves selecting a balance of the accuracy predictions discussed in chapter 4 and chapter 5 and the estimated duration of the FMM runs, which this chapter aims to provide. For the results of this chapter, focus is given to the time complexity of FMM operations, as the speed of performing runs is highly dependent on machine specifications and operating systems. For reference, the simulations in this section were performed on an NVIDIA GeForce RTX 3050 4GB Laptop GPU device and a 13th Gen Intel Core i9-13900H CPU device.

6.1. Time complexity of FMM and Direct method

In this section, the performance of the FMM code is compared to the direct method. For this, a profiling of the time taken to complete a single timestep of velocity computations is performed using both the FMM and direct method. The direct method is fully implemented within a single P2P taichi kernel to ensure the direct computations benefit fully from device acceleration and to minimise any latency caused by communication between the host and device. This study also indicates how well the FMM code implemented meets its theoretical $\mathcal{O}(N)$ complexity.

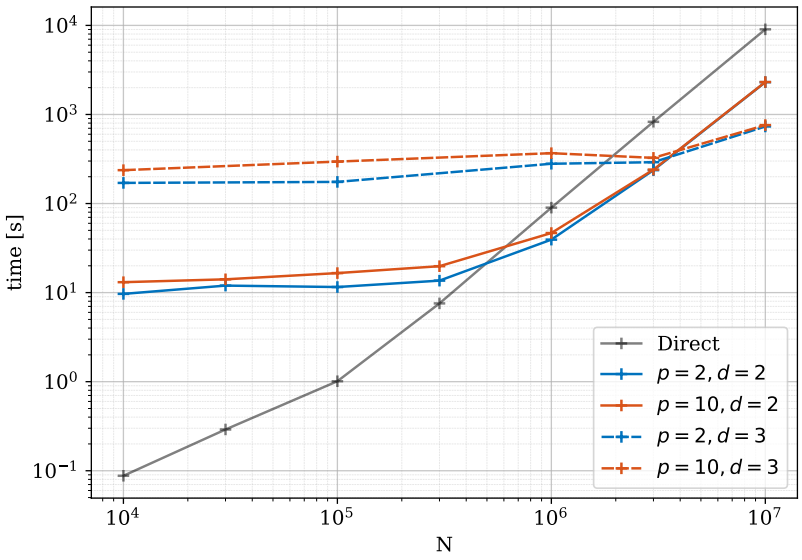


Figure 6.1: Single timestep calculation time plot for a saturated domain on CUDA backend (GPU)

Figure 6.1 shows the computational time taken to run different configurations of FMM for different numbers of particles. This is contrasted with the direct method. Looking at the direct method, it shows complexity of $\mathcal{O}(N)$ for a small number of particles, but then transitions into the expected $\mathcal{O}(N^2)$ complexity. This is mainly attributed to the time taken for compiling the taichi kernel, which consumes a significant portion of the runtime for a small number of particles, but becomes insignificant as N increases.

For FMM, the overhead is much higher than the direct method. This is evidenced by its low sensitivity to the number of particles at small N and the significantly higher computational time required by FMM at small N as a result. This overhead consists of all the necessary FMM processes such as octree generation and management and the computationally expensive operations, such as M2L, which are required by FMM regardless of the number of particles in the system. Host-device communication latency also highly contributes to this overhead, which is caused by the transfer of information from the python scope to the taichi scope and vice-versa.

For the cases with a depth of 2, the complexity appears to be $\mathcal{O}(N^2)$ for high N, which is caused by the large number of P2P interactions that still have to be performed in the near-field. Considering that the leaf clusters are all in level 2, the near field for each cluster takes up a large region of the domain. In contrast, the cases with depth of 3 perform more efficiently due to the smaller near field for each cluster, despite being slowed down by more M2L operations. Because of this, the $\mathcal{O}(N)$ complexity of the FMM algorithm can only be achieved if the octree depth is increased with N as to mitigate the ratio of P2P interactions.

The transition point where FMM becomes more efficient than the direct method occurs around $N \approx 5 \times 10^5$. In comparison with an example from literature, this transition point is an order of magnitude higher than the GPU FMM code developed by Yokota et al. [53][55] where FMM becomes more efficient around $N \approx 5 \times 10^4$. Evidently, there is a large variation in the device specification in both works. Furthermore, the work of Yokota et al. included a parallelization strategy to ensure efficient parallel operations on the GPU clusters, which is not a main objective of this thesis. Nonetheless, this comparison indicates that there is potential for improving the computational efficiency of the FMM developed in this thesis.

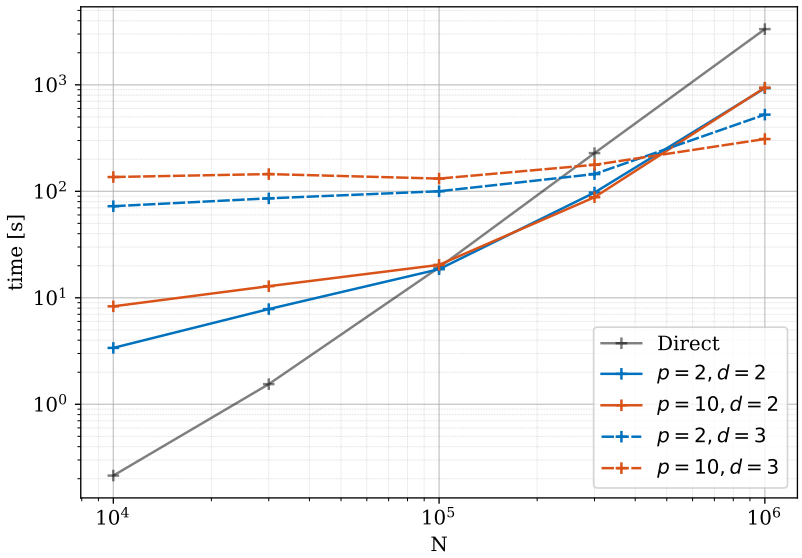


Figure 6.2: Single timestep calculation time plot for a saturated domain on CPU backend

For reference, the same analysis is performed using the CPU as the device, as shown in Figure 6.2. Due to the significantly lower parallel processing ability of CPU's, the P2P operation in particular is much more expensive. Because of this, the efficiency of FMM overcomes that of the direct method earlier when using the CPU as the device, at $N = 1 \times 10^5$.

6.2. Performance breakdown of FMM operations on GPU device

To better understand the performance of the FMM solver, a profiling is performed on single timestep calculation of FMM. In this analysis, the time taken to perform all FMM operations (P2P, P2M, M2M, M2L, L2L, L2P) is given. It was found that the time for other overhead, such as building the octree, finding cluster neighbours or interaction lists, was significantly less time consuming than the FMM operations themselves, and so was omitted for this analysis.

First, the time taken for each FMM operation is shown for a depth of 2 in Figure 6.7 and for a depth of 3 in Figure 6.4. This is performed for a random particle distribution which results in a saturated domain, where all leaf clusters contain a roughly equal number of particles n_p. The performance breakdown is contrasted for runs with different numbers of particles ($N = 1 \times 10^4$, $N = 1 \times 10^6$) and trunction orders ($p = 2$, $p = 10$).

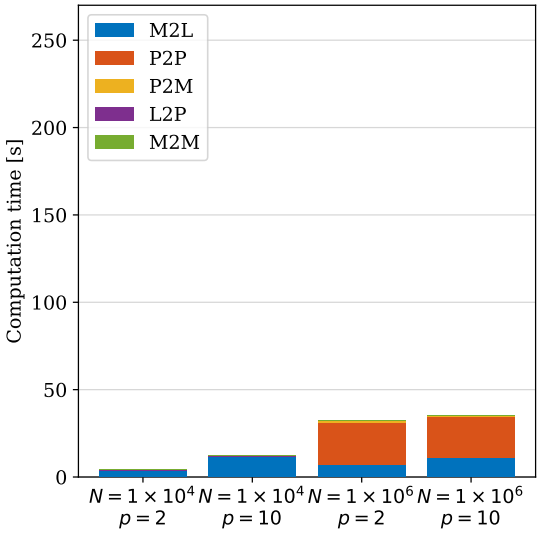


Figure 6.3: Computation time breakdown of FMM operations for a saturated domain at $d = 2$ (GPU)

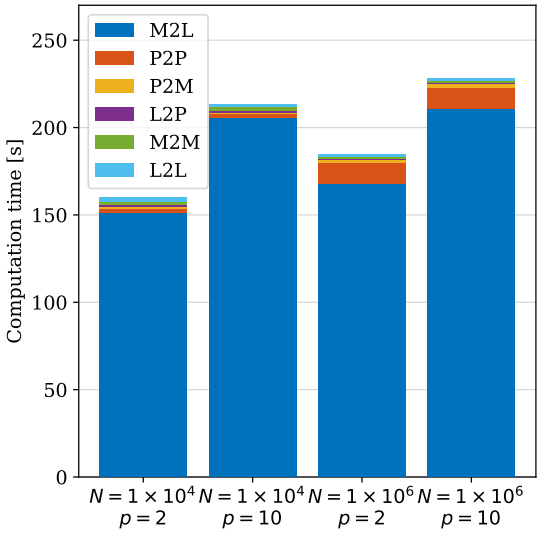


Figure 6.4: Computation time breakdown of FMM operations for a saturated domain at $d = 3$ (GPU)

The main observation from Figure 6.3 and Figure 6.4 is that the P2P and M2L operations take up the largest portion of computational time. Based on this the effect of other operations can be considered negligible for the subsequent analysis. For P2P, this results from it being performed for all pairwise interactions in the near field, which for $d = 2$, amounts to slightly more than $N^2/4$ interactions. The M2L interactions, as shown by Equation 3.5, are expensive double summations which are repeated for each cluster with every cluster in its interaction list. They also require the computation of a large number of derivatives per operation, which are run serially for this FMM code.

The M2L computational time increases greatly when moving from $d = 2$ to $d = 3$. The M2L operation is performed for clusters at all levels starting from level 2 to the level of the leaves. For a depth of 2 this would only be performed for 64 clusters, while for the depth of 3 it is performed for 576 clusters. Furthermore, M2L computational time increases with p as more derivatives have to be computed, but this increase is not proportional to the size of the $\boldsymbol{k}$ vector nor the number of derivatives computed. This is most likely a result of the overhead of host-device communication and taichi kernel compilations. The M2L operation shows a slight sensitivity to N, which was an unexpected observation, as it is an operation independent of the number of particles. It is possible that the memory used by the preceding P2M operation, which increases with N, reduces the computational resources available to the M2L step. On the contrary, P2P shows no sensitivity to the value of p used, which is expected.

For this test case, FMM with $d = 2$ is more performant than $d = 3$ in all given scenarios. Yokota et al. indicates an optimal depth on the GPU of $d = 3$ to $d = 4$ for $N = 5 \times 10^5$ for their FMM code [53]

while here it is $d = 2$, which indicates that the M2L operation in this FMM code is the main operation to be optimised to obtain a more performant FMM code.

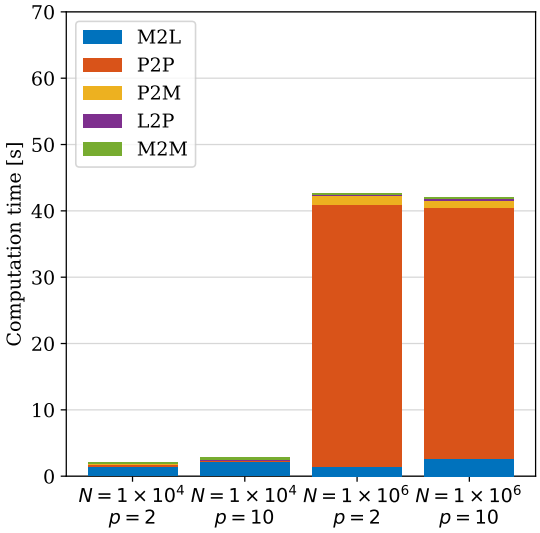


Figure 6.5: Computation time breakdown of FMM operations for a toroidal particle concentration at $d = 2$ (GPU)

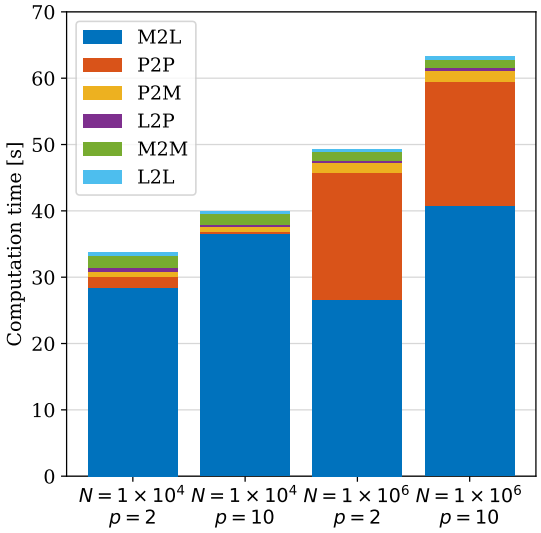


Figure 6.6: Computation time breakdown of FMM operations for a toroidal particle concentration at $d = 3$ (GPU)

Figure 6.5 and Figure 6.6 show the same performance breakdown for a vortex ring case with particles positioned in a torus. One can see that the M2L operation is faster for this case because, unlike the saturated case, there are empty clusters in the domain which do not contribute to the M2L computations. The P2P operation time for high N increases compared to the saturated case due to the presence of more particles in each other's near field. When considering the use of a higher depth, the improvement of performance due to the decrease in M2L operations offsets the increase in P2P operations. One can see that the difference in performance between $d = 2$ and $d = 3$ in this case is less pronounced than that of the saturated case. A saturated domain can be considered a worst case scenario for this FMM code, and in many cases the use of higher depth will not impact the performance of FMM as significantly.

6.3. Performance breakdown of FMM operations on CPU device

Finally, the same saturated case used in Figure 6.3 and Figure 6.4 is repeated using the CPU as the device. The performance breakdown is shown for both depths in Figure 6.7 and Figure 6.8. Compared to the GPU, CPU computations of P2P are significantly slower. This is expected due to the limited number of cores in the CPU compared to the GPU. As the P2P operations can be performed independently they can better utilise the GPU cores with minimal serialization. The M2L operations are however $\approx 2\times$ faster on the CPU. CPUs are more suited to performing logic-heavy operations serially such as the M2L.

Another advantage of using the CPU for the heavy FMM operations such as M2L is that CPUs are generally more suited to performing double precision computations compared to GPUs, thus the rounding error bottlenecks of FMM seen in chapter 4 can be bypassed by using the CPU for FMM operations in the far field, and using the GPU with single precision for the P2P operations in the near field. Nonetheless, an improved parallelisation strategy for the M2L operation, such as computing the kernel derivatives in parallel can result in M2L being more efficient on the GPU.

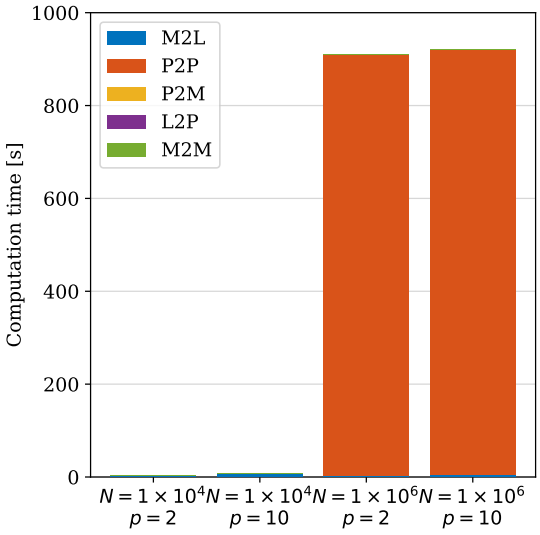


Figure 6.7: Computation time breakdown of FMM operations for a saturated domain at $d = 2$ (CPU)

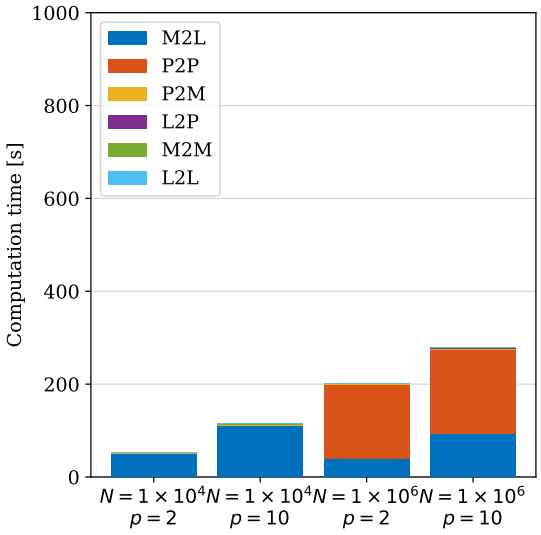


Figure 6.8: Computation time breakdown of FMM operations for a saturated domain at $d = 3$ (CPU)

6.4. A rudimentary approach to depth selection

From the performance analysis in this chapter, it became evident that the P2P and M2L operations are the most computationally expensive. Thus, one can estimate the optimal depth d for achieving the best computational performance out of the FMM algorithm. If we assume:

- Very large number of particles N such that the considerations of compilation and threading overhead for parallelization of the P2P operation is insignificant compared to the number of operations performed, such that $t_{\text{P2P}} \to \mathcal{O}(N^2)$.
- Sufficient depth such that the majority of clusters can be considered to have 189 clusters in their interaction list. In reality, clusters at a low depth have a smaller interaction list because there is a larger proportion of clusters at the border of the octree.
- A saturated domain where all leaf clusters contain particles.

The total time of an FMM run can be approximated as:

$$t_{\text{FMM}} \approx t_{\text{M2L}} + t_{\text{P2P}} \ . \tag{6.1}$$

Where t_{P2P} can be assumed to be related to the depth as:

$$t_{\text{P2P}} \approx t_{\text{P2P}}^{'} \cdot \frac{N^2 \cdot 27}{8^d} \ , \tag{6.2}$$

Where $t_{\text{P2P}}^{'}$ is the approximate time needed to compute one P2P interaction. And t_{M2L}, assuming all clusters have a neighbour list of 189:

$$t_{\text{M2L}} \approx 189 \cdot t_{\text{M2L}}^{'} \cdot \sum_{l=2}^{d} 8^d = 189 \cdot t_{\text{M2L}}^{'} \cdot \frac{8^{d+1} - 8^2}{7} \ , \tag{6.3}$$

Where $t_{\text{M2L}}^{'}$ is the time to compute one M2L interaction between a source and target cluster. The total FMM time can be approximated as:

$$t_{\mathrm{FMM}} \approx t'_{\mathrm{P2P}} \cdot \frac{N^2 \cdot 27}{8^d} + 216 \cdot t'_{\mathrm{M2L}} \cdot (8^d - 8) \ . \tag{6.4}$$

Through equating the derivative to 0 to find the optimal depth:

$$d_{\mathrm{optimal}} \approx \frac{1}{2} \log_8 \left[\frac{N^2 \ t'_{\mathrm{P2P}}}{8 \ t'_{\mathrm{M2L}}} \right] \ . \tag{6.5}$$

It is important to note that this parameter might give a poor estimate for d_{optimal} for FMM at lower depths and with a smaller N. It is best used as a first guess for the depth and adjusted on a case-per-case basis. Furthermore, the tests in chapter 4 only involved depths of up to $d = 4$. It is unclear whether the effect of depth on FMM accuracy will remain insignificant beyond this depth.

7

Conclusions

In this work, a Cartesian FMM solver written in python was developed with the aim of accelerating particle interactions in VPM from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ and integrated within a pre-existing VPM solver. The solver handles octree creation and manipulation on the host, while expensive computational FMM operations handled by the device, an approach used by Kailasa et al. [26]. The solver was verified by means of ensuring convergence of the RMSE of induced velocities in contrast with the direct method. Results show convergence of the FMM solver for all test cases used. Validation of the solver was performed in the integrated VPM-FMM framework using a Lamb-Oseen vortex, vortex ring and colliding vortex rings as test cases, providing an overview of the accuracy of FMM under different physical flows. A central differencing scheme used to obtain the necessary vortex-stretching through velocity gradients was also verified. The performance of this FMM solver was later profiled based on its constituent operations to obtain an understanding of the trade-offs required to ensure an accurate and performant FMM run can be obtained by users of this solver. More detailed conclusions on the results obtained through this thesis are given as answers to the formulated research questions.

Research Question 1A: ***How does a change in FMM parameters impact the accuracy of Cartesian FMM when applied to VPM?***

To answer this research question, a set of best practice guidelines are given to the users of the FMM solver, based on the findings of this thesis.

Taylor series truncation order p

Results showed an increase in accuracy with Taylor series truncation order p where the accuracy would plateau afterwards when FMM was performed in single precision. Cases with a larger number of particles showed a higher error at the plateau, which is believed to result from accumulation of rounding errors through the P2M and L2P operations when using single precision. Users are advised to use a value of p ranging from $p = 2$ to $p = 10$. Generally, it was found that values of $p > 10$ were too dominated by rounding errors displaying the plateau behaviour where accuracy of FMM did not improve beyond that point. If double precision is used, one can expect to see improved accuracy well beyond $p > 10$, but this is generally discouraged for computations on the GPU. With higher Re simulations, a larger p is advised due to the diminished effect of viscous damping in counteracting the accumulation of FMM error. This is also recommended if the simulation consists of a large number of timesteps. With simulations where large velocity gradients or chaotic behaviour dominates, users are recommended to increase p as the accumulation in position and velocity error can become unbounded.

Octree depth d

The octree depth was found to have minimal effect on the accuracy of FMM. The depth is therefore recommended to be tuned to the number of particles and extent of particle spread within the domain. Larger values of N incentivize using a larger depth to counteract the large number of P2P interactions with M2L interactions. For saturated domains, such as in modelling isotropic

turbulence, the use of a lower depth is recommended as the presence of particles in all clusters necessitates a large number of costly M2L interactions. Less saturated cases, such as simulation of a vortex ring, allow for a larger depth to be used as there are less M2L interactions to be performed.

Other remarks

The distribution of particles in a saturated domain had negligible effect on the accuracy of FMM. It was found however, that concentration of particles in some clusters resulted in reduced errors for low N and increased errors for high N. Additionally, little to no sensitivity of FMM error to the overlap ratio σ was found.

Validation of FMM in the VPM framework showed an effect of accumulating FMM error with increase in timestep, with a plateau of FMM error aided by viscous diffusion. The FMM showed stability for both the Lamb-Oseen and vortex ring cases, but showed signs of possible divergence for the colliding vortex ring test case in the vortex breakdown regime. This was found to be acceptable due to the chaotic nature of secondary flow structures in this regime. A drawback of the validation cases is that they used a modest number of particles $N = 5 \times 10^4$ compared to those for which FMM becomes computationally beneficial. Also, as seen in the verification section, the errors of FMM increase significantly as N increases, due to the influence of rounding errors. To obtain a rough estimate of the order of error of FMM, one should scale the errors seen in the validation case, with 5×10^4 particles, to any number of particles N up until 3×10^6. The errors obtained for the cases with $p = 2$ are more representative of the magnitude of error expected for such large number of particles.

Research Question 1B: ***How do the different operations of FMM implementation scale with a change in FMM parameters and number of particles?***

Citing the large overhead in FMM, FMM computations began to be more efficient at $N \approx 5 \times 10^5$ using the GPU as device and at $N \approx 1 \times 10^5$ using the CPU as device. By a large margin, the most expensive operations were found to be the M2L and P2P operations. An increase in depth significantly increased overhead due to the increase in M2L computations, but proved to become more efficient with an increase in N. The P2P computational time was influenced mostly by d and N, while the M2L computational time was influenced greatly by d and by the particle distribution in the domain. Lastly, M2L was found to be more $\approx 2\times$ performant on the CPU than the GPU, while P2P operations offset this by being orders of magnitude more performant on the GPU.

Research Question 2: ***Can a central-differencing scheme be reliably incorporated with Cartesian FMM to compute the velocity gradient tensor?***

Overall, the errors contribution of the CDS to the velocity gradient tensor were found to be of a comparable order of magnitude to the FMM contribution. CDS was thus found to be an acceptable method of computing the velocity gradient tensor for DNS computations. Using a higher order accurate CDS stencil was generally found to improve estimates, while a larger step size δ was found to provide more accurate results through attenuating rounding errors. It is still uncertain whether such a scheme would significantly affect the results of an LES turbulence model through the strain rate tensor $\boldsymbol{S}_{ij}$, which would require validation through an LES test case.

8

Future Recommendations

The analysis performed in this project highlighted many potential improvements to the FMM solver developed. Further research is required to further establish the contexts for which FMM can be suitably used. These findings are compiled as a list of recommendations for future research to build upon the work done in this project.

8.1. Improvements to the FMM solver

Parallel computation of kernel Taylor coefficients in M2L In this work, the computation of kernel derivatives through the Taylor coefficients obtained using recurrence relations is done serially due to the dependencies in recurrence relations. This greatly contributes to the computational expense of M2L operations. This can however be parallelised if the derivatives are obtained if a diagonal plane is taken, such that each derivative can access derivatives in all (-x, -y and $-z$) directions which have already been computed.

Automatic Differentiation on GPU Another possibility is to investigate potential for the theoretically exact automatic differentiation [14] method on GPU, which could bypass the need for recurrence relations, and significantly speed up computation of derivatives in M2L. Furthermore, it could also be used to bypass CDS for computing the velocity gradients, providing a fast and exact alternative which would reduce uncertainty.

Including support for other kernels The Gaussian kernel is more widely used in VPM simulations than the high-order algebraic kernel used in this thesis. Adding support for the Gaussian kernel would greatly improve the applicability of this FMM solver. This would mean that the recurrence relation approach used would not be suitable for the Gaussian kernel, in which case other differentiation techniques would be required.

Combining CPU and GPU as device From the performance analysis performed it became evident that while the GPU were more efficient in computing P2P operations, the CPU was more efficient in performing the M2L operation. Combining the use of CPU for M2L and GPU for P2P would improve the efficiency of the FMM solver, while providing the opportunity for performing double precision computations on the CPU to reduce the impact of rounding errors.

8.2. Further Research

Handling variable particle radii in an LES + CSM context In some LES models, such as the Smagorinsky model, the introduction of a an additional local turbulent viscosity term ν_t results in a different effective viscosity $\nu_{eff} = \nu + \nu_t$ for each particle. As a consequence, with CSM updating particle radii as a function of particle viscosity, the use of CSM in an LES context results in particles with different radii. This is not the case for CSM in a DNS context, or for PSE in both contexts. The current FMM implementation is thus unsuited for LES using the

CSM viscous diffusion method as it assumes the same radius for all particles. It can be useful to investigate the effect of averaging particle radii in each cluster and using a different kernel for each cluster in the M2L operation on the accuracy of the LES subfilter-scale turbulence model.

Investigating the effect of FMM and CDS on an LES test case Because FMM only computes different induced velocities and velocity gradient tensor to the direct method, it can be used as is in both the Smagorinsky and SFS stretching models provided in OpenONDA, which use the full strain rate tensor $\boldsymbol{S}_{ij}$ and do not modify particle cutoff radii σ. While FMM can be potentially used in LES, it is unknown what the effect of computing the velocity gradients using FMM and CDS would have on the accuracy of the LES turbulence model. The Smagorinsky model for example, uses the strain-rate tensor $\boldsymbol{S}_{ij}$ to compute the turbulent viscosity, which is affected by FMM and CDS errors in computing the velocity gradient tensor $\nabla\mathbf{u}$. To investigate this, an analysis on the results and kinetic energy cascade resulting from an LES VPM test case using FMM can be performed.

Further validation using a real scenario test case at high Reynolds number Validation cases in this project were done on theoretical flows a Reynolds number of 1000. It was seen that an increase in Reynolds number has a negative impact on FMM accuracy. Therefore, validation using a test case with a higher Reynolds number can provide more context on the applicability of FMM to real engineering applications. Flows over wind turbines, for example, can have Reynolds numbers of $Re = \mathcal{O}(1 \times 10^6)$.

References

[1] Eduardo Alvarez. "Reformulated Vortex Particle Method and Meshless Large Eddy Simulation of Multirotor Aircraft". phD thesis. Brigham Young University, June 2022. URL: https://scholarsarchive.byu.edu/etd/9589.

[2] Andrew W. Appel. "An Efficient Program for Many-Body Simulation". In: *SIAM Journal on Scientific and Statistical Computing* 6.1 (Jan. 1985). Publisher: Society for Industrial and Applied Mathematics, pp. 85–103. ISSN: 0196-5204. DOI: 10.1137/0906008.

[3] Lorena A. Barba. "Vortex Method for Computing High-Reynolds Number Flows: Increased Accuracy with a Fully Mesh-Less Formulation". en. phd. California Institute of Technology, 2004. DOI: 10.7907/TSR5-DE67. URL: https://resolver.caltech.edu/CaltechETD:etd-05282004-030854.

[4] Josh Barnes and Piet Hut. "A hierarchical O(N log N) force-calculation algorithm". In: *Nature* 324 (Dec. 1986), pp. 446–449. ISSN: 0028-0836. DOI: 10.1038/324446a0.

[5] T Berdowski et al. "Derivation and analysis of the analytical velocity and vortex stretching expressions for an O(N log N)-FMM". en. In: *Journal of Physics: Conference Series* 753.8 (Sept. 2016). Publisher: IOP Publishing, p. 082023. ISSN: 1742-6596. DOI: 10.1088/1742-6596/753/8/082023.

[6] T. J. Berdowski. "3D Lagrangian VPM-FMM for Modelling the Near-Wake of a HAWT". en. MA thesis. Delft University of Technology, 2015. URL: https://repository.tudelft.nl/record/uuid:99174fca-9568-4ed4-9876-b7771b6560c9.

[7] Pierre Blanchard et al. "ScalFMM: A Generic Parallel Fast Multipole Library". en. In: Salt Lake City, United States, Mar. 2015. URL: https://inria.hal.science/hal-01135253 (visited on 05/10/2025).

[8] Richard L. Burden and John Douglas Faires. *Numerical analysis*. eng. 9. ed., International ed. Boston, MA: Brooks/Cole, Cengage Learning, 2011. ISBN: 978-0-538-73351-9 978-0-538-73564-3.

[9] Alexandre Joel Chorin. "Numerical study of slightly viscous flow". en. In: *Journal of Fluid Mechanics* 57.4 (1973), pp. 785–796. ISSN: 1469-7645, 0022-1120. DOI: 10.1017/S0022112073002016.

[10] Georges-Henri Cottet and Petros D. Koumoutsakos. *Vortex Methods: Theory and Practice*. en. Cambridge University Press, Apr. 2008. ISBN: 978-0-521-06170-4.

[11] R. Courant. "Variational methods for the solution of problems of equilibrium and vibrations". In: *Bulletin of the American Mathematical Society* 49.1 (Jan. 1943). Publisher: American Mathematical Society, pp. 1–23. ISSN: 0002-9904, 1936-881X. URL: https://projecteuclid.org/journals/bulletin-of-the-american-mathematical-society/volume-49/issue-1/Variational-methods-for-the-solution-of-problems-of-equilibrium-and/bams/1183504922.full.

[12] Felipe A. Cruz and Lorena A. Barba. "Characterization of the accuracy of the fast multipole method in particle simulations". en. In: *International Journal for Numerical Methods in Engineering* 79.13 (2009). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.2611, pp. 1577–1604. ISSN: 1097-0207. DOI: 10.1002/nme.2611.

[13] Felipe A. Cruz and Lorena A. Barba. *pyfmm*. Feb. 2008. URL: https://github.com/barbagroup/pyfmm (visited on 05/10/2025).

[14] Hend Dawood and Nefertiti Megahed. "Automatic differentiation of uncertainties: an interval computational differentiation for first and higher derivatives with implementation". en. In: *PeerJ Computer Science* 9 (Mar. 2023). Publisher: PeerJ Inc., e1301. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.1301.

[15] P. Degond and S. Mas-Gallic. "The Weighted Particle Method for Convection-Diffusion Equations. Part 1: The Case of an Isotropic Viscosity". In: *Mathematics of Computation* 53.188 (1989). Publisher: American Mathematical Society, pp. 485–507. ISSN: 0025-5718. DOI: 10.2307/2008716.

[16] P. Degond and S. Mas-Gallic. “The Weighted Particle Method for Convection-Diffusion Equations. Part 2: The Anisotropic Case”. In: *Mathematics of Computation* 53.188 (1989). Publisher: American Mathematical Society, pp. 509–525. ISSN: 0025-5718. DOI: 10.2307/2008717.

[17] Michael A. Epton and Benjamin Dembart. “Multipole Translation Theory for the Three-Dimensional Laplace and Helmholtz Equations”. In: *SIAM Journal on Scientific Computing* 16.4 (July 1995). Publisher: Society for Industrial and Applied Mathematics, pp. 865–897. ISSN: 1064-8275. DOI: 10.1137/0916051.

[18] William Fong and Eric Darve. “The black-box fast multipole method”. In: *Journal of Computational Physics* 228.23 (Dec. 2009), pp. 8712–8725. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2009.08.031.

[19] Bengt Fornberg. “Numerical Differentiation of Analytic Functions”. en. In: *ACM Transactions on Mathematical Software* 7.4 (Dec. 1981), pp. 512–526. ISSN: 0098-3500, 1557-7295. DOI: 10.1145/355972.355979.

[20] Leslie Greengard and V Rokhlin. “A fast algorithm for particle simulations”. In: *Journal of Computational Physics* 73.2 (Dec. 1987), pp. 325–348. ISSN: 0021-9991. DOI: 10.1016/0021-9991(87)90140-9.

[21] Leslie Greengard and V Rokhlin. *On the Efficient Implementation of the Fast Multipole Algorithm.* en. Tech. rep. 602. Yale University Department of Computer Science, Feb. 1988. URL: https://apps.dtic.mil/sti/citations/ADA191167.

[22] Tsuyoshi Hamada et al. “42 TFlops hierarchical N-body simulations on GPUs with applications in both astrophysics and turbulence”. In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. SC ’09. New York, NY, USA: Association for Computing Machinery, Nov. 2009, pp. 1–12. ISBN: 978-1-60558-744-8. DOI: 10.1145/1654059.1654123.

[23] Noha A. Al-Harthi. “Performance Benchmarking of Fast Multipole Methods”. en. MA thesis. Thuwal, Kingdom of Saudi Arabia: King Abdullah University of Science and Technology, June 2013. URL: http://hdl.handle.net/10754/293890.

[24] A. Hrennikoff. “Solution of Problems of Elasticity by the Framework Method”. In: *Journal of Applied Mechanics* 8.4 (Mar. 2021), A169–A175. ISSN: 0021-8936. DOI: 10.1115/1.4009129.

[25] Yuanming Hu et al. “Taichi: a language for high-performance computation on spatially sparse data structures”. In: *ACM Trans. Graph.* 38.6 (Nov. 2019), 201:1–201:16. ISSN: 0730-0301. DOI: 10.1145/3355089.3356506.

[26] Srinath Kailasa et al. “PyExaFMM: An Exercise in Designing High-Performance Software With Python and Numba”. In: *Computing in Science & Engineering* 24.5 (Sept. 2022), pp. 77–84. ISSN: 1558-366X. DOI: 10.1109/MCSE.2023.3258288.

[27] *Kernels and Functions | Taichi Docs*. en. May 2023. URL: https://docs.taichi-lang.org/docs/kernel_function (visited on 11/04/2025).

[28] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. “Numba: a LLVM-based Python JIT compiler”. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. Austin Texas: ACM, Nov. 2015, pp. 1–6. DOI: 10.1145/2833157.2833162.

[29] A Leonard. “Vortex methods for flow simulation”. In: *Journal of Computational Physics* 37.3 (Oct. 1980), pp. 289–335. ISSN: 0021-9991. DOI: 10.1016/0021-9991(80)90040-6.

[30] Dhairya Malhotra and George Biros. “PVFMM: A Parallel Kernel Independent FMM for Particle and Volume Potentials”. In: *Communications in Computational Physics* 18.3 (2015). Publisher: Global Science Press, pp. 808–830. ISSN: 1815-2406, 1991-7120. DOI: 10.4208/cicp.020215.150515sw.

[31] N. N. Mansour, J. H. Ferziger, and W. C. Reynolds. *Large-eddy simulation of a turbulent mixing layer*. Tech. rep. TF-11. NTRS Author Affiliations: Stanford Univ. NTRS Document ID: 19780014084 NTRS Research Center: Legacy CDMS (CDMS). Apr. 1978. URL: https://ntrs.nasa.gov/citations/19780014084.

[32] Flavio A. C. Martins. *OpenONDA: Operator for Numerical Design and Fluidynamics*. 2025. DOI: 10.5281/ZENODO.15111460. (Visited on 05/19/2025).

[33] Aaron Meurer et al. “SymPy: symbolic computing in Python”. en. In: *PeerJ Computer Science* 3 (Jan. 2017). Publisher: PeerJ Inc., e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103.

[34] Chloé Mimeau and Iraj Mortazavi. “A Review of Vortex Methods and Their Applications: From Creation to Recent Advances”. en. In: *Fluids* 6.2 (Feb. 2021). Number: 2 Publisher: Multidisciplinary Digital Publishing Institute, p. 68. ISSN: 2311-5521. DOI: 10.3390/fluids6020068.

[35] D. Moore. “Finite Amplitude Waves on Aircraft Trailing Vortices”. In: *Aeronatical Quarterly* 23.4 (Nov. 1972), pp. 307–314. DOI: https://doi.org/10.1017/S000192590000620X.

[36] Guy M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company, 1966.

[37] Susanne Pfalzner and Paul Gibbon. *Many-Body Tree Methods in Physics*. Cambridge: Cambridge University Press, 1996. ISBN: 978-0-521-49564-6. DOI: 10.1017/CBO9780511529368.

[38] S. B. Pope. *Turbulent flows*. Cambridge ; New York: Cambridge University Press, 2000. ISBN: 978-0-521-59125-6 978-0-521-59886-6.

[39] Louis Rosenhead. “The formation of vortices from a surface of discontinuity”. In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 134.823 (Nov. 1931), pp. 170–192. ISSN: 0950-1207. DOI: 10.1098/rspa.1931.0189. URL: https://doi.org/10.1098/rspa.1931.0189 (visited on 12/10/2025).

[40] Louis Rosenhead and Harold Jeffreys. “The spread of vorticity in the wake behind a cylinder”. In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 127 (Jan. 1997). Publisher: Royal Society, pp. 590–612. DOI: 10.1098/rspa.1930.0078.

[41] Enrico Sabatino. “Development of an aerodynamic panel code and a vortex particle wake with the Fast Multipole Algorithm”. en. MA thesis. Politecnico di Milano, July 2020. URL: https://www.politesi.polimi.it/retrieve/447218fc-68de-491b-8265-7c3d0a6c78c9/Sabatino_VPMwithFMM.pdf.

[42] B. Shanker and H. Huang. “Accelerated Cartesian expansions – A fast method for computing of potentials of the form R−ν for all real ν”. en. In: *Journal of Computational Physics* 226.1 (Sept. 2007), pp. 732–753. ISSN: 00219991. DOI: 10.1016/j.jcp.2007.04.033.

[43] P. R. Spalart and A. Leonard. “Computation of separated flows by a vortex-tracing algorithm”. en. In: *14th Fluid and Plasma Dynamics Conference*. Palo Alto,CA,U.S.A.: American Institute of Aeronautics and Astronautics, June 1981. DOI: 10.2514/6.1981-1246.

[44] Mark Stock, Adrin Gharakhani, and Christopher Stone. “Modeling Rotor Wakes with a Hybrid OVERFLOW-Vortex Method on a GPU Cluster”. In: *28th AIAA Applied Aerodynamics Conference*. Chicago, Illinois: American Institute of Aeronautics and Astronautics, June 2010. DOI: 10.2514/6.2010-4553.

[45] Matteo Tugnoli et al. “Mid-fidelity approach to aerodynamic simulations of unconventional VTOL aircraft configurations”. In: *Aerospace Science and Technology* 115 (Aug. 2021), p. 106804. ISSN: 1270-9638. DOI: 10.1016/j.ast.2021.106804.

[46] Lei Wang, Robert Krasny, and Svetlana Tlupova. “A kernel-independent treecode based on barycentric Lagrange interpolation”. In: *Communications in Computational Physics* 28.4 (Oct. 2019). arXiv:1902.02250 [math], pp. 1415–1436. ISSN: 1991-7120, 1815-2406. DOI: 10.4208/cicp.OA-2019-0177.

[47] Tingyu Wang, Rio Yokota, and Lorena A. Barba. “ExaFMM: a high-performance fast multipole method library with C++ and Python interfaces”. en. In: *Journal of Open Source Software* 6.61 (May 2021), p. 3145. ISSN: 2475-9066. DOI: 10.21105/joss.03145.

[48] M. S. Warren and J. K. Salmon. “A parallel hashed Oct-Tree N-body algorithm”. In: *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*. Supercomputing ’93. New York, NY, USA: Association for Computing Machinery, Dec. 1993, pp. 12–21. ISBN: 978-0-8186-4340-8. DOI: 10.1145/169627.169640.

[49] D. Wee et al. “Convergence Characteristics and Computational Cost of Two Algebraic Kernels in Vortex Methods with a Tree-Code Algorithm”. In: *SIAM Journal on Scientific Computing* 31.4 (2009). Publisher: Society for Industrial and Applied Mathematics, pp. 2510–2527. ISSN: 1064-8275. DOI: 10.1137/080726872.

[50] G. S. Winckelmans. *Some Progress in Large-Eddy Simulation using the 3-D Vortex Particle Method*. 1995. URL: https://ntrs.nasa.gov/citations/19960022324 (visited on 05/19/2025).

[51] G. S. Winckelmans and A. Leonard. “Contributions to Vortex Particle Methods for the Computation of Three-Dimensional Incompressible Unsteady Flows”. In: *Journal of Computational Physics* 109.2 (Dec. 1993), pp. 247–273. ISSN: 0021-9991. DOI: 10.1006/jcph.1993.1216.

[52] Lexing Ying, George Biros, and Denis Zorin. “A kernel-independent adaptive fast multipole algorithm in two and three dimensions”. In: *Journal of Computational Physics* 196.2 (May 2004), pp. 591–626. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2003.11.021.

[53] R. Yokota et al. “Fast multipole methods on a cluster of GPUs for the meshless simulation of turbulence”. In: *Computer Physics Communications* 180.11 (Nov. 2009), pp. 2066–2078. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2009.06.009.

[54] R. Yokota et al. “Petascale turbulence simulation using a highly parallel fast multipole method on GPUs”. In: *Computer Physics Communications* 184.3 (Mar. 2013). arXiv:1106.5273 [cs], pp. 445–455. ISSN: 00104655. DOI: 10.1016/j.cpc.2012.09.011.

[55] Rio Yokota and Shinnosuke Obi. “Vortex Methods for the Simulation of Turbulent Flows: Review”. In: *Journal of Fluid Science and Technology* 6.1 (2011), pp. 14–29. DOI: 10.1299/jfst.6.14.

[56] Wen Zhang and Stephan Haas. “Adaptation and performance of the Cartesian coordinates fast multipole method for nanomagnetic simulations”. en. In: *Journal of Magnetism and Magnetic Materials* 321.22 (Nov. 2009), pp. 3687–3692. ISSN: 03048853. DOI: 10.1016/j.jmmm.2009.07.016.

[57] Feng Zhao. “An O(N) Algorithm for Three-Dimensional N-Body Simulations”. MA thesis. Cambridge, MA: MIT Artificial Intelligence Laboratory, Oct. 1987. URL: https://dspace.mit.edu/handle/1721.1/6962.

A

Supplementary statistics from FMM error quantification

A.1. Lamb-Oseen Vortex: Random Particle Distribution

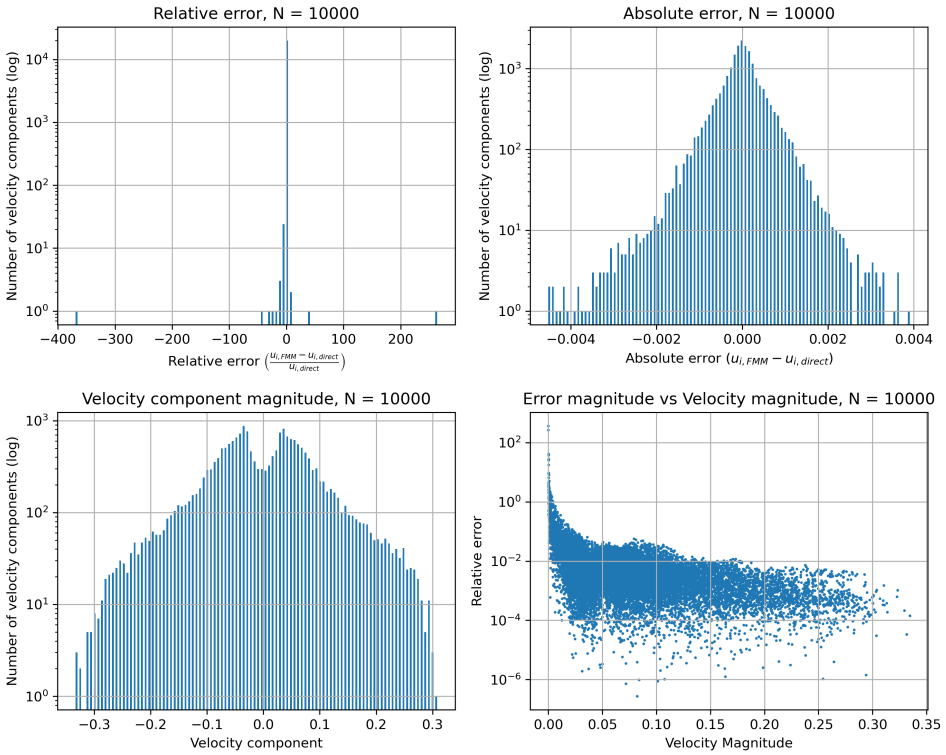


Figure A.1: Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^4$, $d = 2$, $p = 2$, random particle distribution

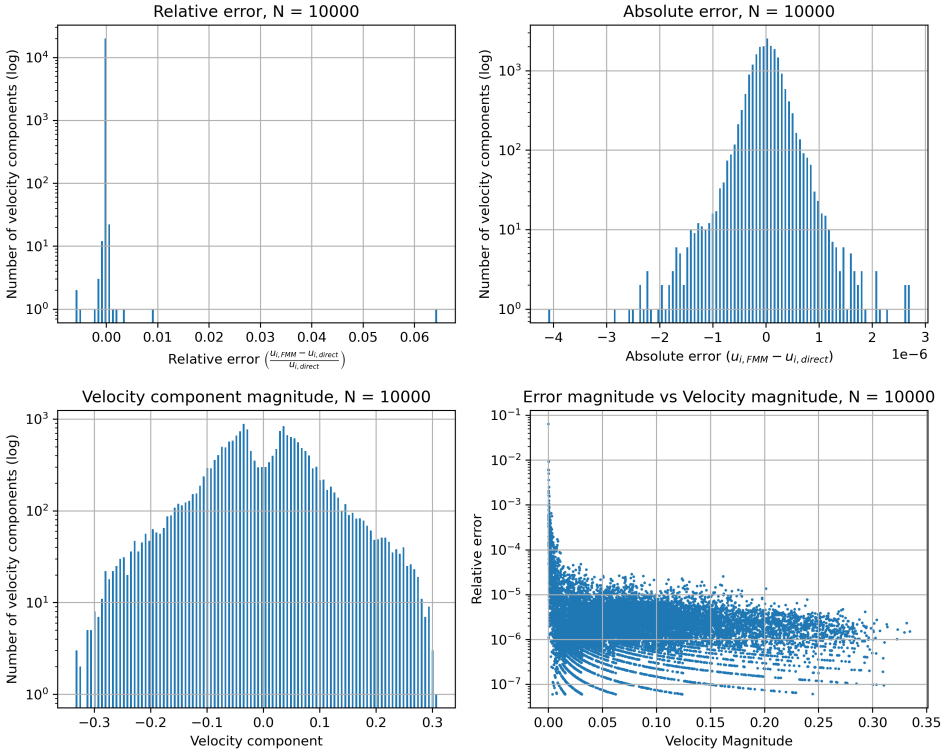


Figure A.2: Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^4$, $d = 2$, $p = 10$, random particle distribution

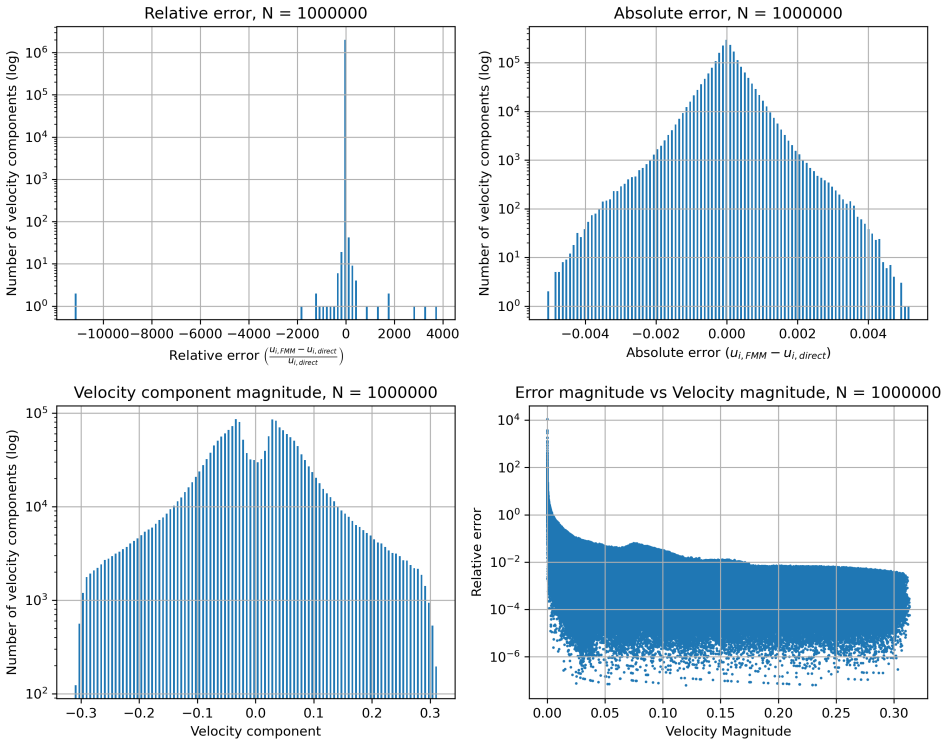


Figure A.3: Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^6$, $d = 2$, $p = 2$, random particle distribution

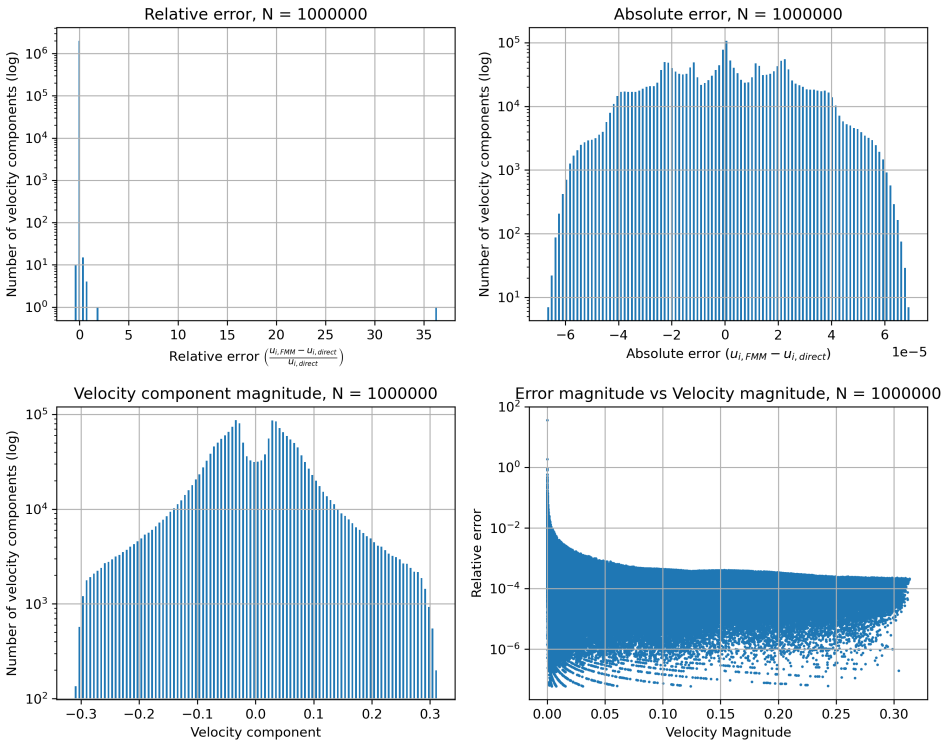


Figure A.4: Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^6$, $d = 2$, $p = 10$, random particle distribution

A.2. Lamb-Oseen Vortex: Uniform Particle Distribution

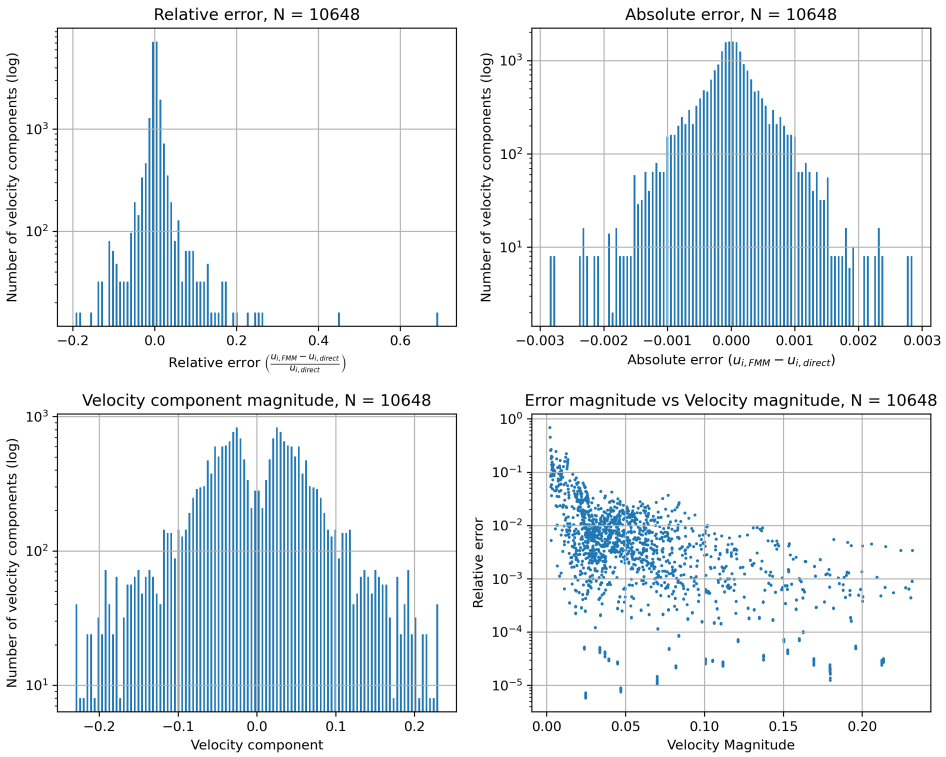


Figure A.5: Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^4$, $d = 2$, $p = 2$, uniform particle distribution

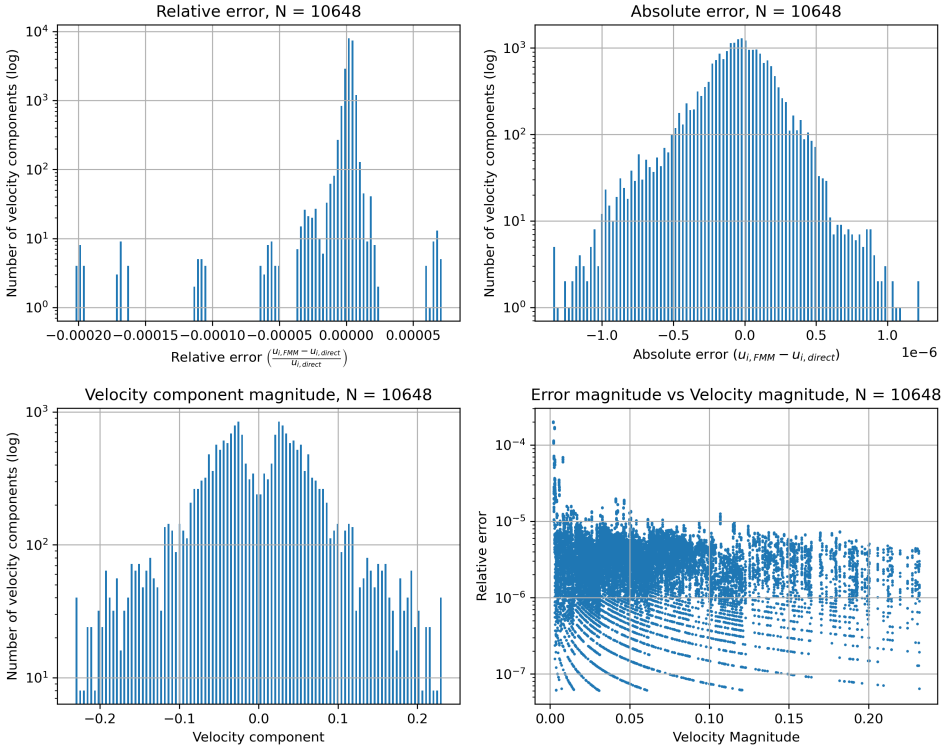


Figure A.6: Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^4$, $d = 2$, $p = 10$, uniform particle distribution

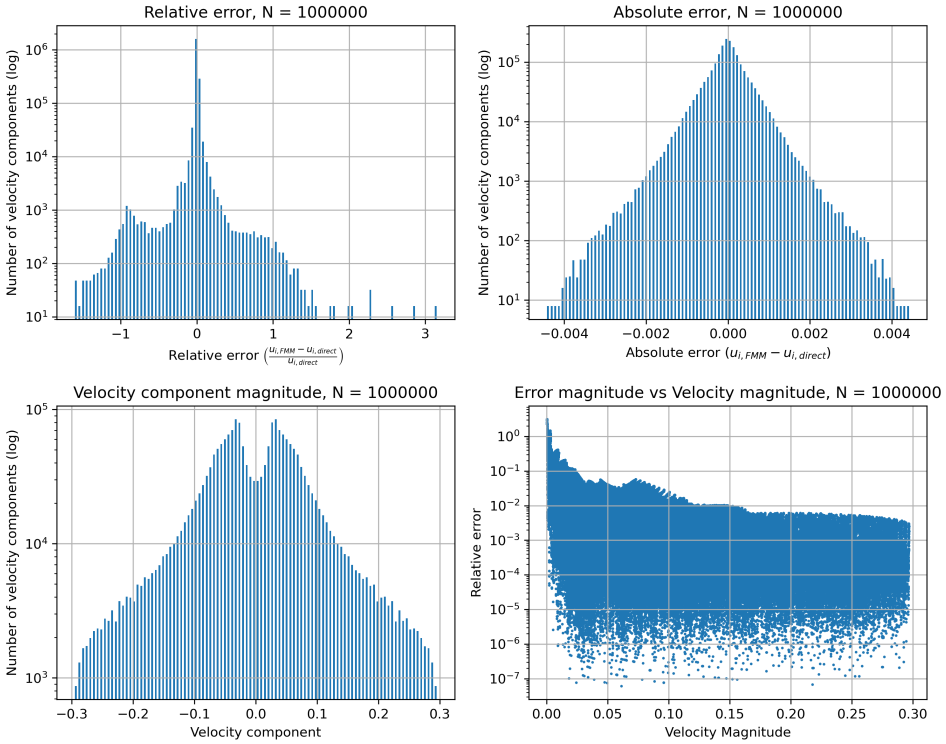


Figure A.7: Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^6$, $d = 2$, $p = 2$, uniform particle distribution

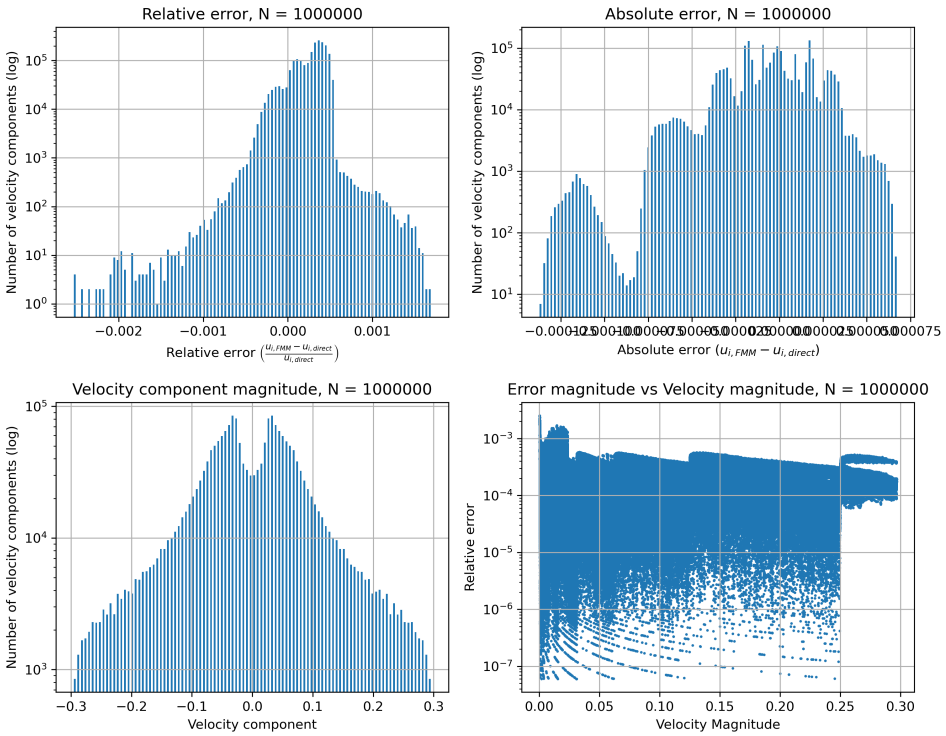


Figure A.8: Statistical figures for Lamb–Oseen vortex, $N = 1 \times 10^6$, $d = 2$, $p = 10$, uniform particle distribution

A.3. Vortex Ring: Random Particle Distribution

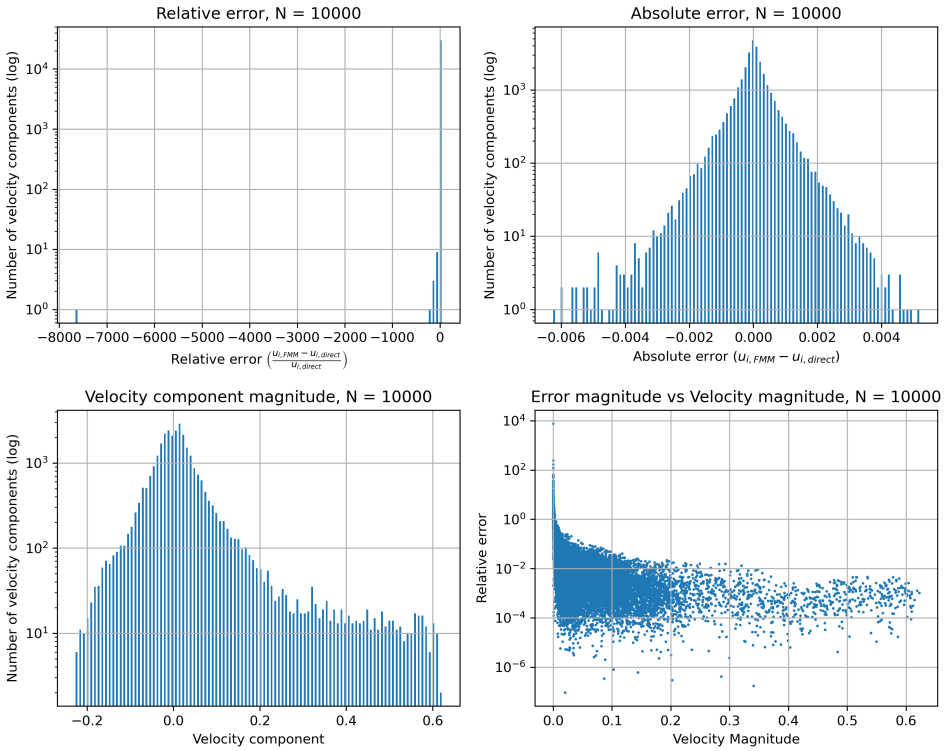


Figure A.9: Statistical figures for vortex ring, $N = 1 \times 10^4$, $d = 2$, $p = 2$, random particle distribution

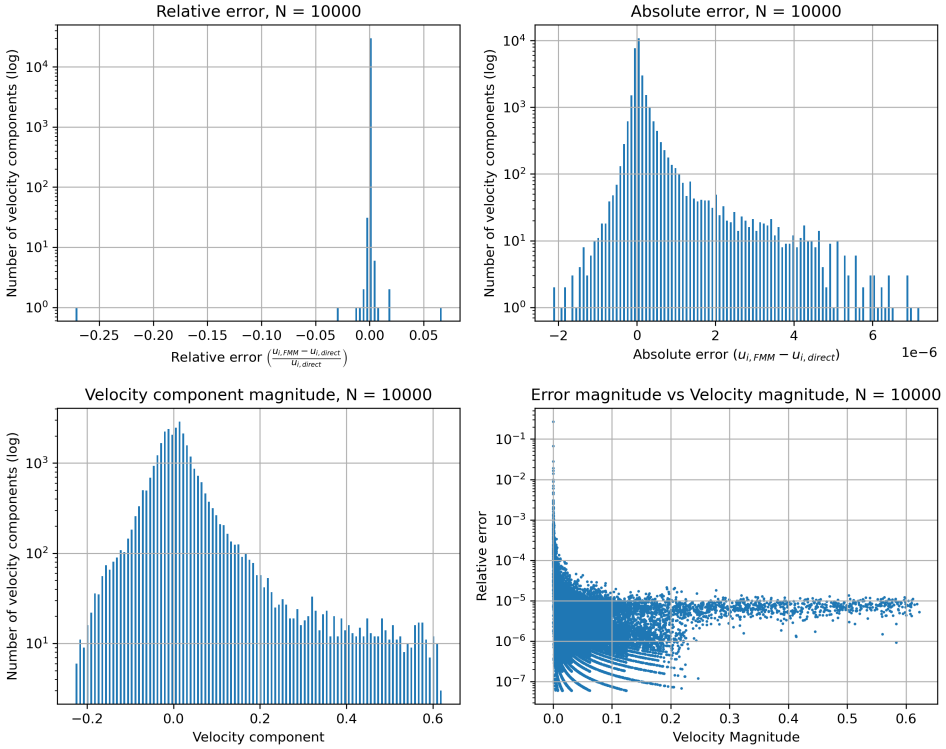


Figure A.10: Statistical figures for vortex ring, $N = 1 \times 10^4$, $d = 2$, $p = 10$, random particle distribution

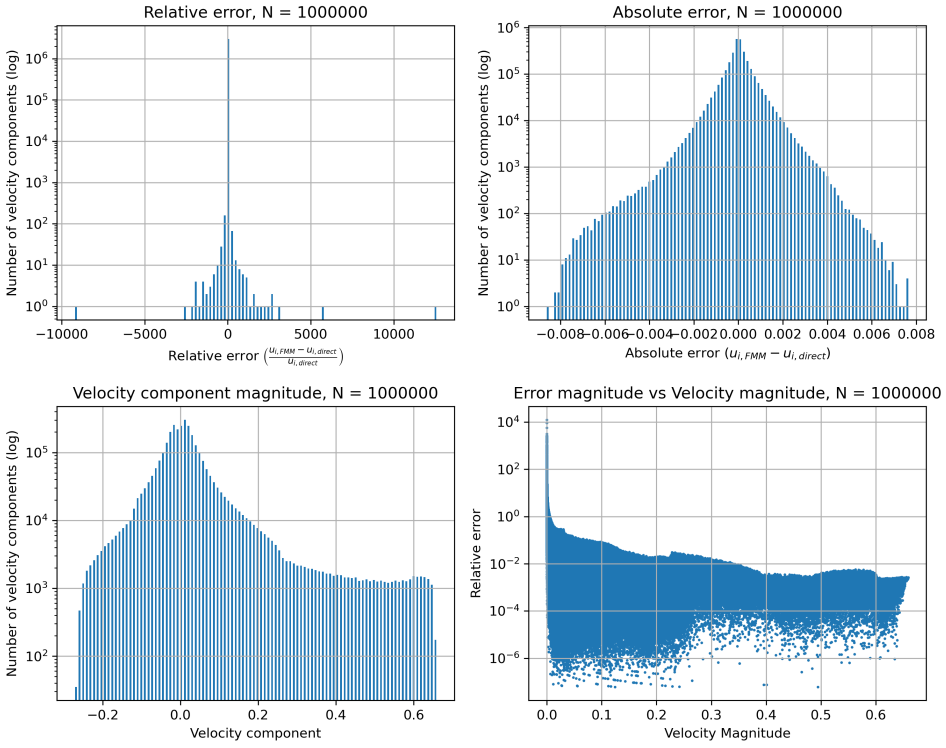


Figure A.11: Statistical figures for vortex ring, $N = 1 \times 10^6$, $d = 2$, $p = 2$, random particle distribution

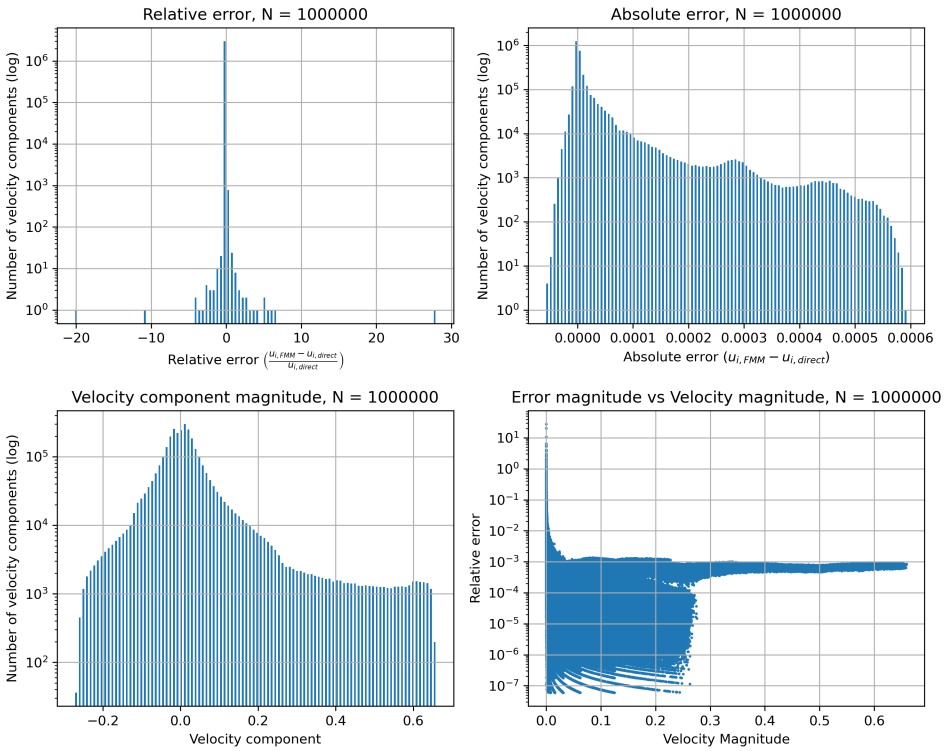


Figure A.12: Statistical figures for vortex ring, $N = 1 \times 10^6$, $d = 2$, $p = 10$, random particle distribution

A.4. Vortex Ring: Toroidal Particle Distribution

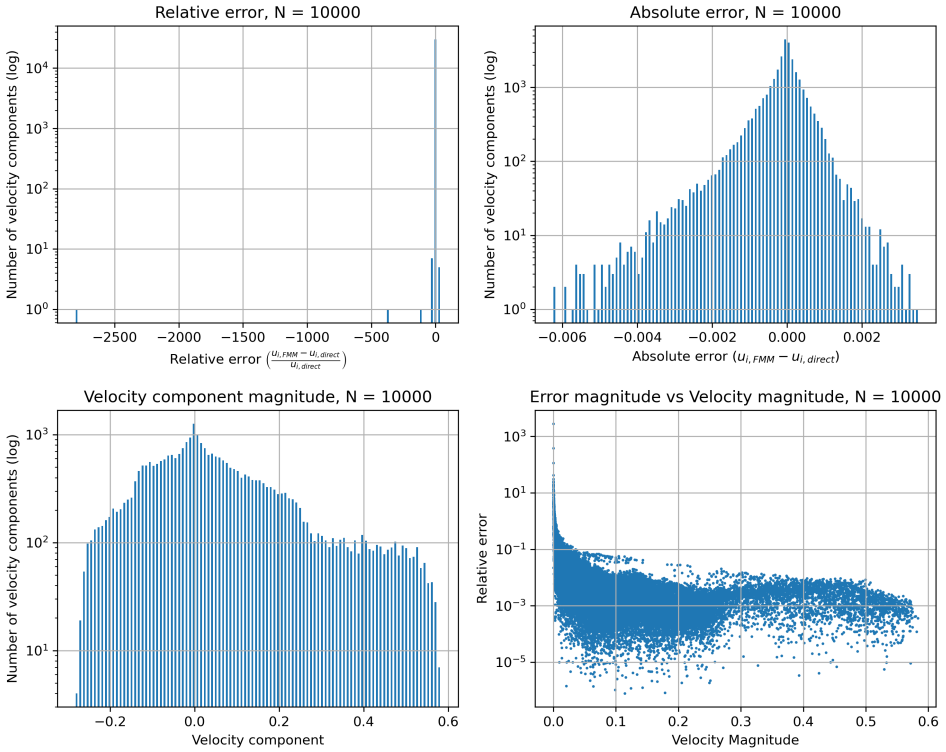


Figure A.13: Statistical figures for vortex ring, $N = 1 \times 10^4$, $d = 2$, $p = 2$, toroidal particle distribution

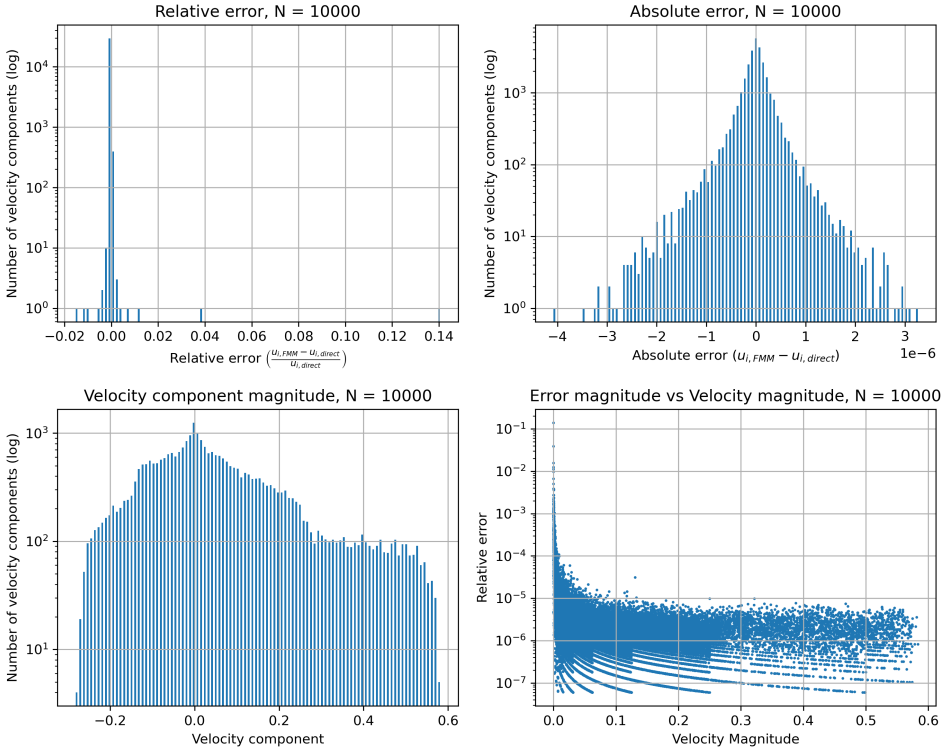


Figure A.14: Statistical figures for vortex ring, $N = 1 \times 10^4$, $d = 2$, $p = 10$, toroidal particle distribution

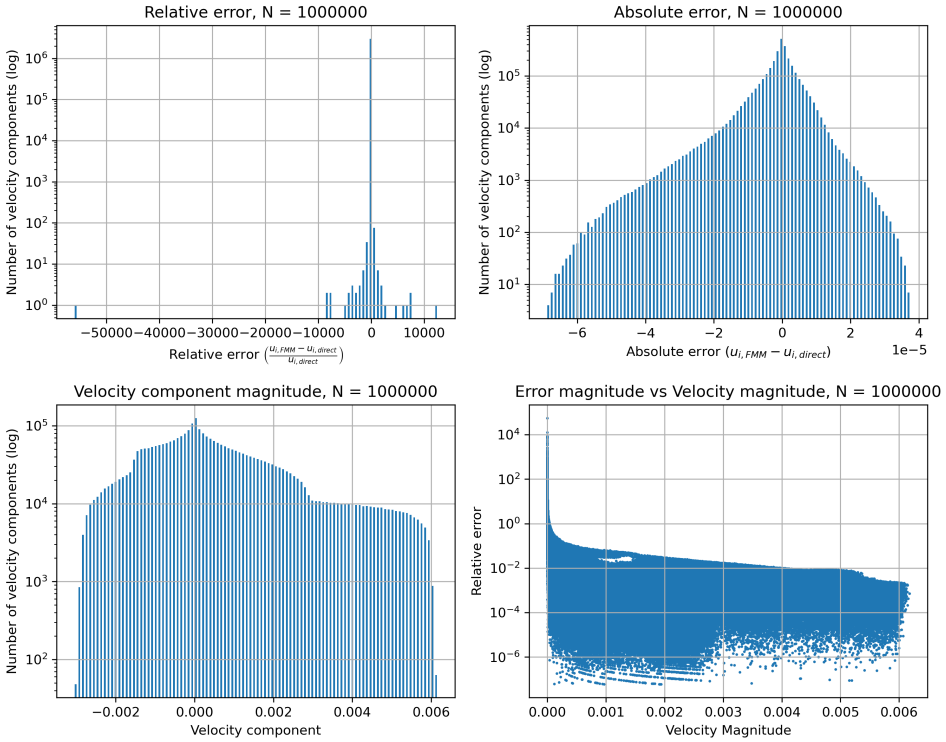


Figure A.15: Statistical figures for vortex ring, $N = 1 \times 10^6$, $d = 2$, $p = 2$, toroidal particle distribution

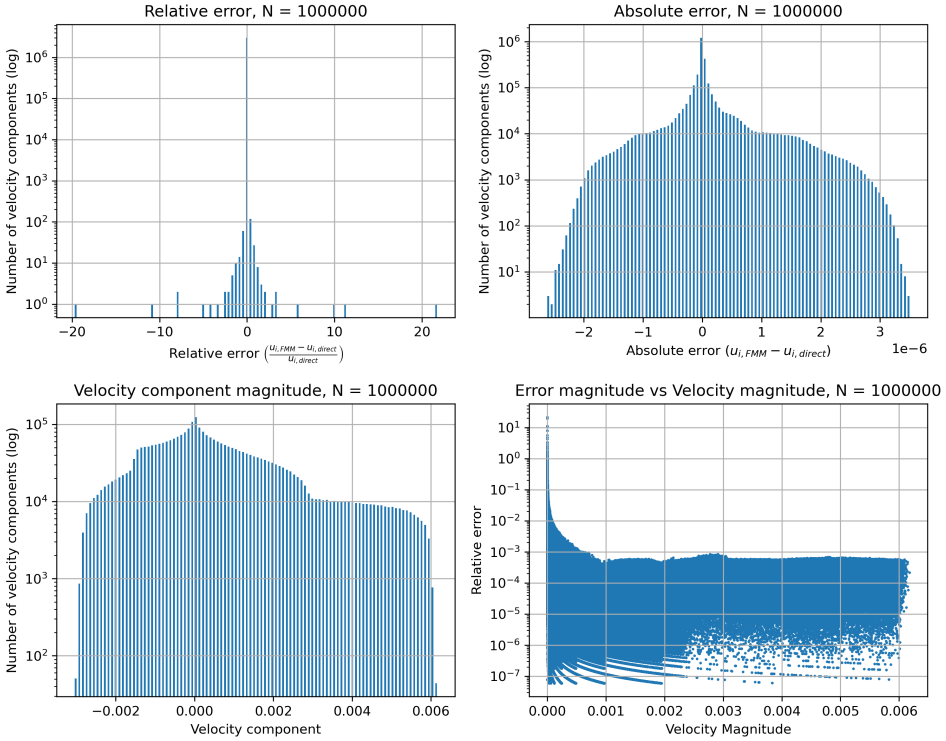


Figure A.16: Statistical figures for vortex ring, $N = 1 \times 10^6$, $d = 2$, $p = 10$, toroidal particle distribution