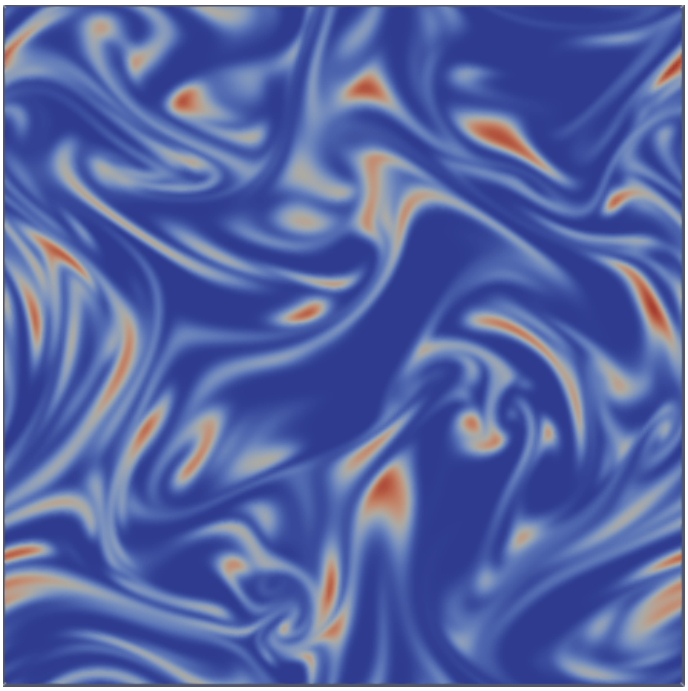
Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Energy and Process Engineering

Tommy Hellen

A multiple-resolution scheme for DNS of scalar fields in turbulent flows

Master's thesis in Mechanical Engineering
Supervisor: Prof. Luca Brandt
June 2022



NTNU
Kunnskap for en bedre verden

Tommy Hellen

A multiple-resolution scheme for DNS of scalar fields in turbulent flows

Master's thesis in Mechanical Engineering
Supervisor: Prof. Luca Brandt
June 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of Energy and Process Engineering

NTNU
Norwegian University of
Science and Technology

Master`s Agreement / Main Thesis Agreement

Faculty	**Faculty of Engineering**
Institute	**Department of Energy and Process Engineering**
Programme Code	**MTPROD**
Course Code	**TEP4925**

Personal Information

Surname, First Name	**Hellen, Tommy**
Date of Birth	**16.02.1996**
Email	**tommyhel@stud.ntnu.no**

Supervision and Co-authors

Supervisor	**Luca Brandt**
Co-supervisors (if applicable)	
Co-authors (if applicable)	

The Master`s thesis

Starting Date	**15.01.2022**
Submission Deadline	**11.06.2022**
Thesis Working Title	**Multi-resolution CFD for scalar turbulence**
Problem Description	**Try to develop a mulit-resolution scheme, using interpolation, for scalar turbulence to make DNS a quicker method.**

This Master`s agreement must be signed when the guidelines have been reviewed.

Signatures

Tommy Hellen
Student

12.01.2022
Digitally approved

Luca Brandt
Supervisor

12.01.2022
Digitally approved

Anita Yttersian
Department

21.01.2022
Digitally approved

Abstract

For turbulent flows with the presence of an active or passive scalar field, similar to the Kolmogorov scale η, the smallest scales where transport of scalar concentration happens is defined as the Batchelor scale λ_B. When solving the momentum and advection-diffusion equations for turbulent flows with direct numerical simulation (DNS), with a high Schmidt (or Prandtl) number, the necessary grid spacing increases as the Batchelor scale gets smaller and smaller. This implies that the method might be over-resolving the momentum equations, making DNS more computationally expensive than necessary. In this thesis, it's presented and implemented a method to reduce the computational time needed to resolve the fine-scale scalar transport in turbulent flows, by interpolating the coarse velocity field onto a finer grid, obtained by enforcing conservation of mass. This ensures that the interpolated velocity field on the fine grid is divergence free, and the advection-diffusion equation can be solved on a finer grid, while the momentum and pressure equations can be solved on the base grid. For evaluating the proposed method several simulations using the high-performance computing system Betzy were carried out. The total divergence of the interpolated velocity field was estimated to be on the order of $O(10^{-10})$, and the maximum divergence on the order of $O(10^{1})$, in the turbulent region. The large values for the divergence are likely to be related to the MPI parallelization for the interpolation module developed. The time evolution of the temperature field variance for different resolutions was compared and found the multiple-resolution method to over-solve the temperature variance. The inaccurate results suggest that possible sources of error are likely due to the implementation of the transient linear interpolation, and possibly the large divergence. The reduction of overall CPU time was found to be approximate 61.30 % when using a coarse grid for the turbulent velocity field, and a fine grid for the scalar field.

Sammendrag

For turbulente strømninger med tilstedeværelse av et aktivt eller passivt skalarfelt, tilsvarende Kolmogorov-skalaen η, er de minste skalaene hvor skalarkonsentrasjon transporteres definert som Batchelor-skalaen λ_B. Når moment- og adveksjon-diffusjonslikningene for turbulente strømninger løses med direkte numerisk simulering (DNS), med et høyt Schmidt (eller Prandtl), øker den nødvendige rutenettavstanden etter hvert som Batchelor-skalaen blir mindre og mindre. Dette innebærer at metoden kan overløse moment-ligningene, noe som gjør DNS mer beregningsmessig kostbart enn nødvendig. I denne oppgaven presenteres og implementeres en metode for å redusere den numeriske beregningstiden som trengs for å løse den finskalerte skalartransporten i turbulente strømninger, ved å interpolere det grove hastighetsfeltet til et finere rutenett, gjennom bevaring av masse. Dette sikrer at det interpolerte hastighetsfeltet på det fine rutenettet er divergensfritt, og adveksjons-diffusjonsligningen kan løses på et finere rutenett, mens moment- og trykkligningene kan løses på det grove rutenettet. For å evaluere den foreslåtte metoden ble det utført flere simuleringer ved hjelp av høyytelses datasystemet Betzy. Den totale divergensen til det interpolerte hastighetsfeltet ble estimert til å være i størrelsesorden $O(10^{-10})$, og maksimal divergens på orden $O(10^{1})$, i det turbulente området. De store verdiene for divergensen er sannsynligvis relatert til MPI-parallelliseringen for interpolasjonsmodulen utviklet. Tidsutviklingen av temperaturfeltvariansen for forskjellige oppløsninger ble sammenlignet og funnet at metoden med fler-oppløsning overestimerte temperaturvariansen. De unøyaktige resultatene tyder på at mulige feilkilder skyldes sannsynligvis implementeringen av transient lineær interpolasjon, og muligens den store divergensen. Reduksjonen av samlet CPU-tid ble funnet å være omtrent 61.30 % ved bruk av et grovt rutenett for det turbulente hastighetsfeltet, og et fint rutenett for det skalare feltet.

Acknowledgment

I would like to give my thanks to Prof. Luca Brandt who has been very helpful in supervising the work and for finding an interesting topic for this master thesis. It has been both challenging and educational. Also a huge thanks to Marco Crialesi-Esposito, together with Nicolò Scapin and Shahab Mirzareza for being patient, and helping me along the way with developing the code and finding relevant research literature. I would also like to show my appreciation to The Department of Energy and Process Engineering (EPT) at NTNU for providing me with the necessary tools and facilitating a great, inspirational and educational environment.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

ABC Arnold-Beltrami-Childress

CFD Computational fluid dynamics

CFL Courant-Friedrichs-Lewy

DNS Direct numerical simulation

FFT Fast Fourier transform

FluTAS Fluid Transport Accelerated Solver

HIT Homogeneous isotropic turbulence

HPC High-performance computing

MPI Message Passing Interface

MTHINC Multi-dimensional tangent hyperbola interface capturing

VOF Volume of fluid

Symbols

α Thermal diffusivity

ϵ Kinetic energy dissipation rate

η Kolmogorov scale

κ Thermal conductivity

λ_B	The Batchelor scale
μ	Dynamic viscosity
ν	Kinematic viscosity
Φ	Volume of Fluid function
ρ	Density
σ_T	Temperature variance
τ	Turbulent characteristic time
$\vec{V}s$	Interpolated velocity field
$\vec{V}$	Velocity field
C	Courant number
C_p	Specific heat capacity
H	The color function
L	Length of spatial domain
M	Number of refinement
N	Number of grid cells
p	Pressure
Pr	Prandtl number
Re	Reynolds number
Sc	The Schmidt number
T_c	Carrier phase temperature
T_d	Disperse phase temperature

1 Introduction

Scalar fields which are diffused and advected in turbulent flows are a phenomenon that is all around in both nature and many different industries and is a crucial part of how fluids interact with their surroundings and other fluids. Direct numerical simulation (DNS) of scalar turbulent flow is becoming a more and more important tool, with cases such as pollution of air and water [4], or turbulent convection when heating and cooling devices [5]. This means that both the experimental and numerical study of scalar transport in turbulence is important to understand and model the underlying mechanisms and processes. This thesis will explain how the numerical study of scalar turbulence can be computationally costly, and what attempts can be done to reduce the necessary computational time it requires.

When using DNS on turbulent flows, the cascading energy down to the smallest scales, the Kolmogorov scale η, needs to be captured. Equally, with the presence of either a passive or active scalar field, the grid spacing needs to be as small as the Batchelor scale λ_B. In scalar turbulent flows with high Schmidt number Sc (or Pr) the Batchelor scale becomes even smaller than the Kolmogorov scale. This results in the need for even more grid points for DNS to be accurate and thus making the computational cost even higher.

This research work attempts to create and implement an algorithm to reduce the overall computation time for resolving a scalar turbulent flow. The method proposed in this paper is to a large extent inspired by the work done by Chong et al. (2018) and Ostilla-Mónico et al. (2015), and is a continuation of the project work at NTNU (2021) [6–8]. The method consists of an interpolation scheme that converts the divergence free velocity field to a finer grid, such that the scalar field can be solved on a finer grid, but the pressure and momentum equations can be solved on a coarser grid. Solving these equations on a coarser grid allows for less computational time as these are the more costly equations to solve. This multiple-resolution scheme makes it so that the need for high resolution in flows with high Schmidt number Sc (or Pr) is reduced, while still preserving the accuracy of the resolved scalar field.

To test the proposed method, the interpolation scheme was implemented in the Fluid Transport Accelerated Solver (FluTAS), an existing DNS code developed for multiphase flows [9]. Several simulations were done using the high-performance computing system Betzy, to test and quantify potential computational savings, while still resolving the scalar field accurately [10].

The flow that will be studied is a homogeneous isotropic turbulent (HIT) flow, using the Arnold-Beltrami-Childress (ABC) conditions [11]. The initial aim was to

simulate emulsion in turbulent flow, however, due to time-limitation and technical challenges, the flow studied is essentially a single-phase flow with a droplet-like temperature distribution, to facilitate mixing and transport of a scalar field. In other words, the properties of the flow are uniform, and the temperature distribution emulates an initial emulsion condition and secures heat transfer occurring, to test the accuracy of resolving the scalar field for different resolutions.

This thesis contains 7 chapters. First, Section 2 presents relevant theory and previous research work within the given topic. Section 3 gives a detailed description of the methods used in this thesis work, and the numerical scheme implemented. Further, Section 4 presents the obtained results, where the analysis is discussed in Section 5. Lastly, Section 6 and Section 7 contains conclusions and suggestions for further work.

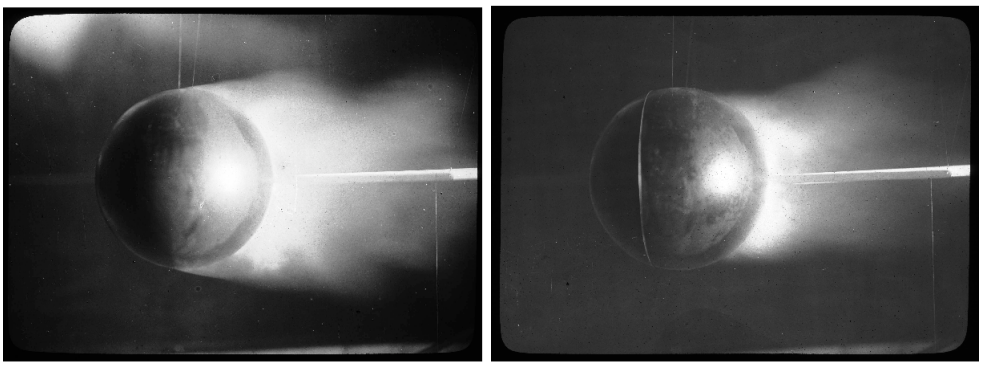
2 Background and Theory

Understanding the physics and behaviour of turbulent flow interactions through simulation is of great interest in various fields. In computational fluid dynamics (CFD) there are several different numerical methods to simulate turbulence and this section will give an overview of how turbulent flow can be modelled, DNS and it's governing equations, and lastly various work done within scalar turbulence and multiphase flow. Several portions of this section (2.1 - 2.3) have been gathered from what was presented in the project work [8].

2.1 Turbulence modelling

Turbulence or turbulent flow is of chaotic and irregular nature, and is characterized by the significant change in pressure and velocity. The highly unpredictable behaviour of turbulent flow makes it a challenge to fully describe in detail and is usually categorized in three different length scales in a cascading manner. The larger scales affect the smaller scales, but the smaller scales can also impact the larger scales [12].

In 1914 Ludwig Prandtl performed experiments with a sphere and a tripwire around the sphere [1]. The experiment aims to show how the boundary layer at the sphere surface affects the turbulence behind the sphere. As Figure 1 shows, the presence of a small diameter around the sphere heavily affects the turbulence occurring behind.



(a) Without a trip wire. (b) With a trip wire.

Figure 1: A reproduction of the original experiment done in 1914 by Prandtl [1].

This experiment shows how even the small turbulent scales, called the Kolmogorov scales η, can affect the larger scales.

Furthermore, looking at the Navier-Stokes equation describing turbulent flow [13], the governing momentum equation reads:

$$\rho \frac{D\vec{V}}{Dt} = -\nabla p + \mu \nabla^2 \vec{V} + \rho \vec{g} \ , \tag{1}$$

where the first term is the material derivative of the velocity field $\vec{V}$, ρ is the density, p is the pressure and $\vec{g}$ is the gravitational acceleration acting on the fluid. The third term is the viscous term, where μ is the dynamic viscosity, and can't be neglected in turbulent flow. When solving the momentum equation, assuming the viscosity term to be neglected would mean that $\mu = 0$. However by definition, this means that there would be zero drag in the fluid, but this can not be the case as shown in Figure 1. So that means that viscosity impacts the behaviour of the flow, and it is necessary to calculate this term at every scale, even the smallest ones:

$$\eta = (\frac{\nu^3}{\epsilon})^{1/4} \ , \tag{2}$$

where η is the Kolmogorov scale, ν is the kinematic viscosity and ϵ is the kinetic energy dissipation rate [14]. The dissipation rate of turbulent kinetic energy can be described using Reynolds average equation [15]:

$$\epsilon = \frac{1}{2}\nu \overline{(\frac{\partial u_i'}{\partial x_j} + \frac{\partial u_j'}{\partial x_i})^2} \ , \tag{3}$$

where u_i' is the fluctuating velocity. As seen from Equation (3) the dissipation rate is determined from the velocity gradients and the viscosity.

For statistically steady state and incompressible flows, where homogenous isotropic turbulent (HIT) conditions are obtained, it can be shown that the turbulent kinetic energy equation results in [16]:

$$-\overline{u_i' u_j'} \frac{\partial \overline{U_i}}{\partial x_j} = \epsilon = \nu \overline{\frac{\partial u_i'}{\partial x_j} \frac{\partial u_i'}{\partial x_j}} \ , \tag{4}$$

where $\bar{U}_i$ is the mean velocity in each direction. The left side of the equation is the production term P, and as the equation shows should be equal to dissipation when statistical steady state is achieved.

2.2 Numerical methods

For turbulent flows, there are three main different ways to calculate the flow: either by using large-eddies simulation (LES), Reynolds-averaged Navier-Stokes model (RANS) or direct numerical simulation (DNS). Although experiments on turbulence have given important insight, numerical simulation is also an important tool in fluid mechanics, as it allows for studying the flow behavior only by simulations.

The DNS directly solves the Navier-Stokes Equation (1), while the RANS method models the flow. The difference in simulation can be seen in Figure 2, where the difference in accuracy can be seen for a simulation of jet flow [2].

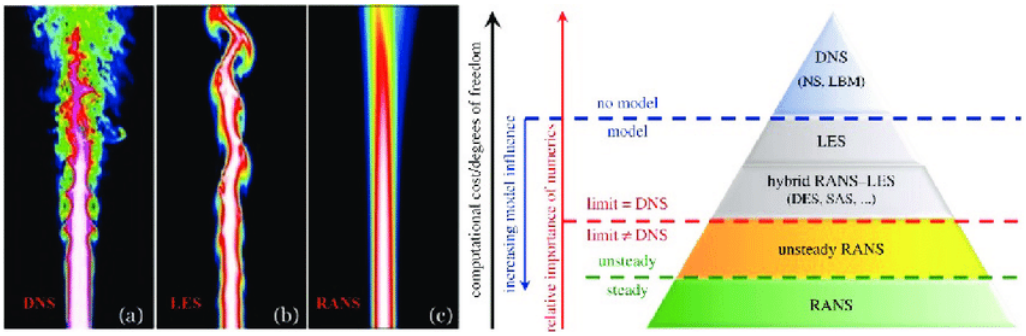


Figure 2: A comparison of the different numerical simulation in turbulent flow [2].

Although DNS is a costly method, the advancements of high-performance computing (HPC) renders DNS a more and more viable option for studying turbulence.

2.2.1 DNS

Compared to the two other main categories of simulating turbulent flow, DNS do not model the flow in any way but tries to capture all of the flow properties at all scales. Since the DNS method is a way to directly simulate the flow, the method needs to resolve the Navier-Stokes equation for all scales in both time and space. This means that the method must solve the Kolmogorov scales, with a spatial increment $\Delta x \leq \eta$. Since the Kolmogorov scales are given by Equation (2), the number of grid cells N^3 that is necessary to properly resolve the flow depends on the Reynolds number Re:

$$N^3 = Re^{9/4} \, . \tag{5}$$

From this relation, it can be observed that the the necessary number of grid points increases substantially with increasing Reynolds number.

Considering the necessary increment in time Δt, this must be small enough such that the Courant-Friedrichs-Lewy (CFL) condition is satisfied [17]. For a 3-dimensional grid with velocities in all three directions, the condition yields as follows:

$$C = \Delta t \sum_{i=1}^{3} \frac{u_{x_i}}{\Delta x_i} < C_{max} \ , \tag{6}$$

where C is the Courant number and C_{max} is a value depending in whether the method used is explicit or implicit. This means when trying to solve the Navier-Stokes equation for turbulence, it's necessary to have both a high time and space resolution. Consequently, this makes DNS is a costly method and usually only simple flows are studied when using this numerical method.

2.3 Scalar turbulence

Although there have been extensive experimental work on the study of scalar turbulence, DNS renders to be a more and more useful tool for studying these types of flows [18, 19]. A comprehensive overview of the studies of simulations of passive scalar field can be reviewed in the work done by Warhaft (2000) [20].

There are two different types of scalar fields in turbulent flows: either passive or active. An active scalar field means that the scalar values are coupled with the vector field. An example is the buoyancy force due to changes in temperature in the fluid, which would affect the flow [21]. A passive scalar field is when the scalar values are decoupled and thus do not affect the vector field [20]. In the method put forth in this paper, both passive and active scalar fields can be solved.

As discussed in Section 2.1 the corresponding length scale to viscous dissipation rate is the Kolmogorov scale η. Likewise, the Batchelor scale λ_B is the corresponding scale to the scalar field [22]:

$$\lambda_B = (\frac{\mu D^2}{\epsilon})^{1/4} = \frac{\eta}{Sc^{1/2}} \ , \tag{7}$$

where D is the mass diffusivity and Sc is the Schmidt number. For the time evaluation of the scalar field, the governing equation is given by the advection-

diffusion equation [23]:

$$\frac{\partial c}{\partial t} + \vec{V} \cdot \nabla c = \alpha \nabla^2 c + R \ , \tag{8}$$

where c is the scalar value e.g temperature, R is a source/sink term, and α is the diffusivity coefficient. In the case where the scalar field is a temperature field, the Schmidt number translate to the Prandtl number Pr and α is the thermal diffusivity.

From Equation (7) it's possible to see that for large Schmidt numbers (or Prandtl number), the Batchelor scale where the scalar transport happens, becomes increasingly smaller than the Kolmogorov scale. For single-resolution methods, one therefore needs a grid spacing Δx small enough to solve the advection-diffusion equation at the Bachelor scale λ_B, and as a result, the single-resolution method tends to over resolve for momentum and pressure equations corresponding to the Kolmogorov scale. This is the main reason why the computational resources to resolve scalar turbulent flow are such high, and a method to combat this is presented Section 3.3.

Previous work on passive scalar fields have involved the use of LES-type filter in the viscous-convective sub-range to filter the scalars in turbulent flow with high Schmidt numbers [24]. Another study done by Ostilla-Mónico et al. (2015) uses a multiple-resolution scheme to improve simulations with active scalar field [7]. This method uses the gradient to find the interpolated velocities at the finer grid, and solves the coupled temperature and velocity for the coarse and dense grid separately. However, this method can not readily be adopted to a finite-volume algorithm. The method proposed by Chong et al. (2018) will be presented in Section 3.3, and will work on both an active or passive scalar field for a finite-volume code [6].

2.4 Multiphase flow

As mentioned in Section 1, the initial aim was to simulate a multiphase flow. Though the flow is a single-phase, the simulation uses some parts of the multiphase solver FluTAS regarding multiphase modelling, which will be explained further in Section 3.1.3. The underlying modelling used in these portions of the code will therefore be briefly explained.

In fluid mechanics a flow can be single-phase or multiphase, where multiphase flow have the presence of two or more thermodynamic phases. A separated multiphase flow has the attributes of two or more continuous flowing fluids with an interface, e.g. laminar flow of oil and water. A dispersed multiphase flow is a case where it's typically a carrier phase and a dispersed phase, where the latter is

occupying the former discontinuously throughout the fluid, e.g bubbles in water [25].

One type of multiphase flow is called emulsion, which will be discussed further, where there are two or more immiscible liquid phases, such as the case with oil in water. Here the dispersed phase can either be partially or totally emulsified in the carrier phase. These types of flow are highly relevant in industrial applications such as food processing and oil production [26, 27]. One recent research study on emulsion in turbulence is the work done by Crialesi-Esposito et al. (2022), which is a numerical study on how different properties affect the HIT in emulsion [28].

2.4.1 Modelling multiphase flow

When modelling and discretizing multiphase flow the type of flow needs to be considered. In the simulation done in this study the free-surface Volume of Fluid (VOF) method has been used for the initialization of the scalar field. The VOF method can track the advection of the interface in the flow, and along with solving the Navier-Stokes equation resolve the multiphase flow [29]. The advection equation for the VOF method can be written as

$$\partial_t \Phi + \partial_i u_i H = \Phi \partial_i u_i \ , \tag{9}$$

where H is the color function with a value of either 1 or 0, depending on which phase Ω_i is occupying the cell:

$$H(\vec{x}, t) = \begin{cases} 1 & \text{if } \vec{x} \in \Omega_1 \\ 0 & \text{if } \vec{x} \in \Omega_2 \end{cases} \ , \tag{10}$$

The VOF function Φ is the cell-averaged of H and can track the interface between the two phases in both time and space. Once Φ is known, the thermal properties can be evaluated for each phase.

3 Method

In this section the governing equations, along with the multiphase flow solver will be presented. The numerical methods used for this research work will also be discussed. Lastly, the developed algorithm will also be presented.

As mentioned earlier the main idea behind this numerical scheme is gathered from previous work done on active scalar fields in turbulence, with a finite-volume method code [6]. This method was chosen because of the ability to compare results to the work previously done. In order to test this, a code using modern Fortran 90 as programming language was developed, so that it could be incorporated into an already existing Fluid Transport Accelerated Solver (FluTAS) for multiphase simulations [9]. This code was developed by the research group of Prof. Luca Brandt and will also be briefly presented. Most of the figures and equations presented in Section 3.3 have been gathered from the previous work done in the project work [8].

3.1 Numerical solver

The numerical method and equations used in FluTAS will be briefly described in the following section. For the full description of the numerical method for solving the Poisson and momentum equations the detailed description can be reviewed in the work done with FluTAS [9].

3.1.1 Governing equations

In this study, an incompressible Newtonian fluid was studied, meaning the governing equations are the Navier-Stokes equations (1) and the continuity equation [30]:

$$\nabla \cdot \vec{V} = 0. \tag{11}$$

The momentum equations are solved with a second-order pressure correction algorithm, and the resulting Poisson equation is solved with a FFT-based finite-difference direct solver as presented in the work by P. Costa [31]. Lastly the correction of the velocity field is done by imposing the divergence free constrain, and the pressure is updated from the previous time-step.

As mentioned in section 2.4.1 the VOF method can be used for multiphase flow. In the work done by Sugiyama et al. [29] its presented a Multi-dimensional Tangent Hyperbola Interface Capturing (MTHINC) algorithm, which can be used

in conjunction with the advection Equation (9). The full description of the VOF method will not be thoroughly discussed here, since the case study does not essentially use the VOF function throughout the simulation. It is only used for the initialization of the scalar field, in this case the temperature field, at the start of the simulation. This is to emulate a multiphase flow and ensure scalar mixing occurring, where the dispersed fluid has a different initial temperature. This was done to make the code and the simulation as simple as possible, as one of the objectives of this research work is to test if the interpolation scheme works well when integrated with the multiphase solver.

3.1.2 Heat transfer in flow

In the presence of an active temperature field, the FluTAS code solves the energy equation explicitly and coupled with the momentum equations using the Oberbeck-Boussinesq approximation [32]. As previously stated, the method proposed works for both active and passive scalar fields. Here, a passive temperature field was chosen due to simplicity reasons.

The aim of the multiple-resolution scheme is to solve the advection-diffusion Equation (8) for a scalar field in a turbulent flow as accurate as possible. Since the scalar field in this case is a passive temperature field, the resulting convection-diffusion equation can be evaluated separately on the fine grid. The temperature field can be discretized using the explicit second-order Adams-Bashforth method [33]:

$$T^{n+1} = T^{n} + \Delta t^{n+1}(f_{t,1} M_T^{n} - f_{t,2} M_T^{n-1}) \ , \tag{12}$$

where the coefficients of the Adams-Bashforth scheme is given by:

$$\begin{aligned} f_{t,1} &= 1 + 0.5\Delta t^{n+1}/\Delta t^{n} \\ f_{t,2} &= 0.5\Delta t^{n+1}/\Delta t^{n}. \end{aligned}$$

The operator M_T in Equation (12) equates to the advection and diffusion terms and can be semi-discretized as:

$$M_T^{n} = -\nabla \cdot (\vec{u}^{n} T^{n}) + \frac{1}{\rho^{n+1} C_p^{n+1}} \nabla \cdot (\kappa^{n+1} \nabla T^{n}) \tag{13}$$

where C_p is the specific heat capacity, ρ is the density and κ is the thermal diffusivity.

For estimating the accuracy of resolving the scalar field with Equation (12) in regards to the spatial grid spacing, the variance of the temperature field was studied. In the absence of a source or sink, the variance of a scalar field with constant diffusion coefficient D can be evaluated as [34]:

$$\frac{\partial \Theta}{\partial t} = -D \int (\nabla c)^2 dx dy dx \ , \tag{14}$$

where Θ is the variance over the spatial domain. Equation (14) shows that the variance of the temperature field can only decrease over time. The temperature field was estimated throughout the simulation to check if this is the case:

$$\sigma_T = \sqrt{\frac{\sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \sum_{k=1}^{N_z} (T_{i,j,k} - \bar{T})^2}{N_x N_y N_z}} \ , \tag{15}$$

where σ_T is the variance of the temperature field and $\bar{T}$ is the mean value of the temperature.

3.1.3 Code structure

As mentioned the multiphase solver used in the simulation is the module based FluTAS code, which can be tailored to different physical scenarios. To incorporate the interpolation scheme, an interpolation module in Fortran F90 was made so that it either can be switch on or of depending on if it's a single or multiple-resolution grid.

The code uses different flags or switches such as turbulent forcing, VOF, Boussinesq approximation, etc. These inputs are used for the preprocessor program before the main code is run. Depending on what flow is studied, the different features can be switch on or off.

The essential structure of the code can be seen in the flow chart below, see Fig. 3. Since the temperature field in this case is a passive scalar field, the momentum equations can be solved directly from the initial VOF and velocity field. Then, the velocity field can be interpolated, and the convection-diffusion equation can be solved using the VOF and the thermal diffusivity α on the finer grid.

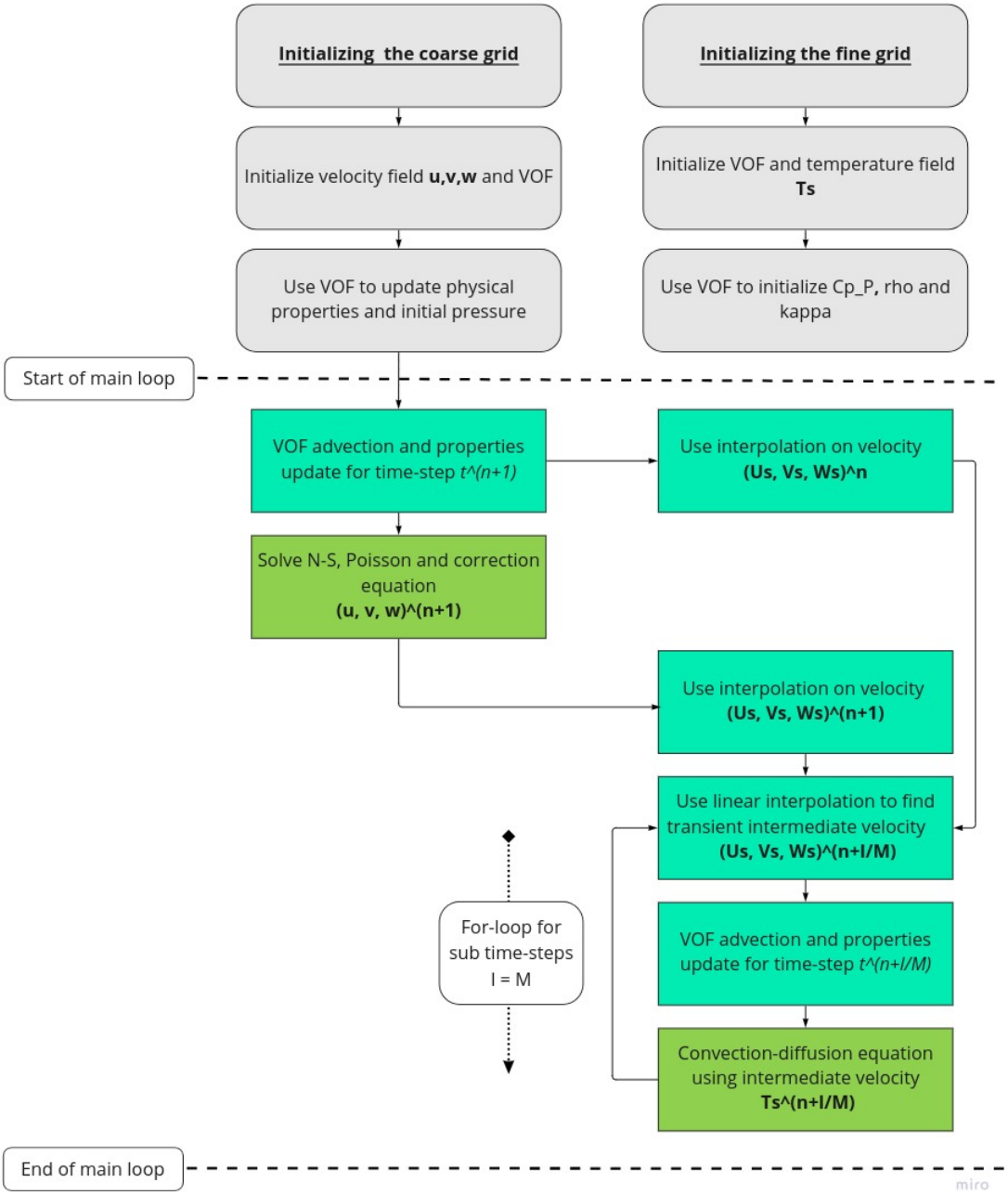


Figure 3: Shows the code structure of the multiphase solver with a passive temperature field.

As Figure 3 above shows, the code is structured for a passive scalar field, however in the case of an active scalar field, the only change would be to extrapolate the scalar field to the coarse field, before solving the momentum equation with the Boussinesq approximation. This will be discussed further in Section 3.3.4.

3.1.4 Parallelization

The FluTAS code takes advantage of MPI task parallelization [35]. In short, the spatial grid domain is divided in "sub-domains" by using 2-dimensional pencil decomposition of the grid. More specifically, here the domain is decomposed in y- and z-direction. Throughout the simulation, the different sub-domains communicate via MPI tasks, and an *all-to-all* communication is required for the transposition of the 2D decomposition.

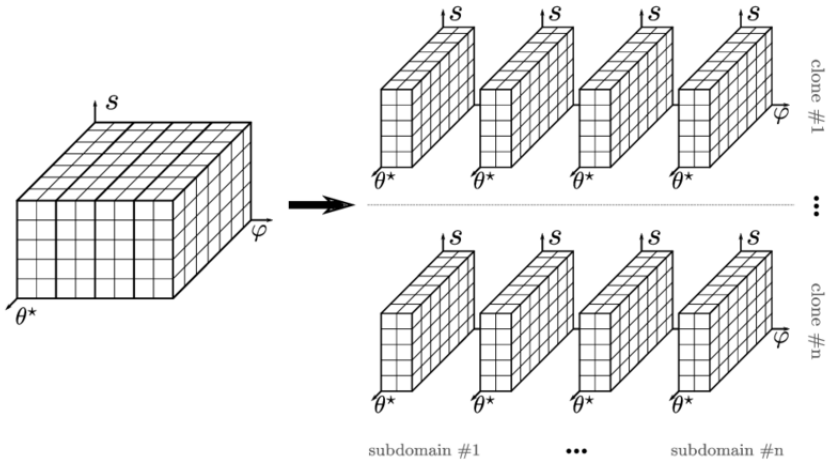


Figure 4: Illustrates how the parallelization works using the pencil decomposition [3].

By decomposing the domain with the usage of MPI and an FFT interface, illustrated in Fig. 4, the simulation can be done on n processor cores. This allows for a more efficient code and a much faster simulation time.

3.2 Flow configuration

3.2.1 Governing equations

To sustain turbulence, a forcing term f_i^T is added to the Navier-Stokes Eq. (1). In the simulations done in this research the Arnold-Beltrami-Childress (ABC) forcing was chosen, to sustain homogeneous isotropic turbulence (HIT) [11]. The forcing term is governed by:

$$\begin{aligned} f_x &= A \sin k_0 z + C \cos k_0 y \\ f_y &= B \sin k_0 x + A \cos k_0 z \\ f_z &= C \sin k_0 y + B \cos k_0 x \end{aligned} \tag{16}$$

where $\{x, y, z \in [0, 2\pi]\}$, k_0 is the forcing wavelength and A, B and C are constants. Here the constants have been put to $A = B = C = 1$ and the wavelength to $k_0 = 2$. The velocity field $V(x, y, z)$ subsequently is also governed by the ABC equations, and tri-periodic boundary condition is used. This flow was chosen since it both can sustain turbulence and satisfy the Navier-Stokes equation, and the parameters were chosen such that the coarse grid spacing Δx is able to capture the dissipation rate ϵ at the Kolmogorov scale η.

3.2.2 Numerical and physical parameters

Below is an overview of the parameters used in all the simulations. Table 1 shows both the numerical and physical parameters.

Table 1: Table shows parameters used for the different simulations.

Description	Variable	Value
Number of cells	N^3	$128^3 - 256^3$
Number of refinement	M	2
CFL condition	C_{max}	0.25
Length	L^3	2π m
Density	ρ	1 kg/m^3
Dynamic viscosity	μ	0.02 kgm^{-1}s^{-1}
Specific heat capacity	Cp_P	4000 Jkg^{-1}K^{-1}
Case 1	Variable	Value
Thermal conductivity	κ	80 Wm^{-1}K^{-1}
Prandtl number	Pr	1
Case 2	Variable	Value
Thermal conductivity	κ	8 Wm^{-1}K^{-1}
Prandtl number	Pr	10

The parameters chosen for this study is purely based on comparing the resolutions for $Pr = 1$ and $Pr = 10$, and is not based on any physical data.

The characteristic time used for studying the different properties of the scalar field and the turbulent flow can be evaluated as [36]:

$$\tau = \frac{L}{\overline{V}_{rms}} \quad , \text{ where } \overline{V}_{rms} = \frac{U_{rms} + V_{rms} + W_{rms}}{3} \tag{17}$$

The $U_{rms}, V_{rms}, W_{rms}$ is the root mean square of the velocity field. Because of the homogeneous condition, the root mean square velocities in each direction

should be equal.

The initial temperature distribution have been constructed as a droplet-like field using n_b number of "droplets" with a volume ratio $\beta = 0.15$, which can be seen in Figure 5. This was done first off to create a scalar field that can be evaluated as it advects and diffuses throughout the simulation, to see if the multiple-resolution method is able to accurately resolve the scalar field. And secondly, to emulate a multiphase flow where the carrier and dispersed phases have different initial temperatures while keeping the condition as simple as possible.

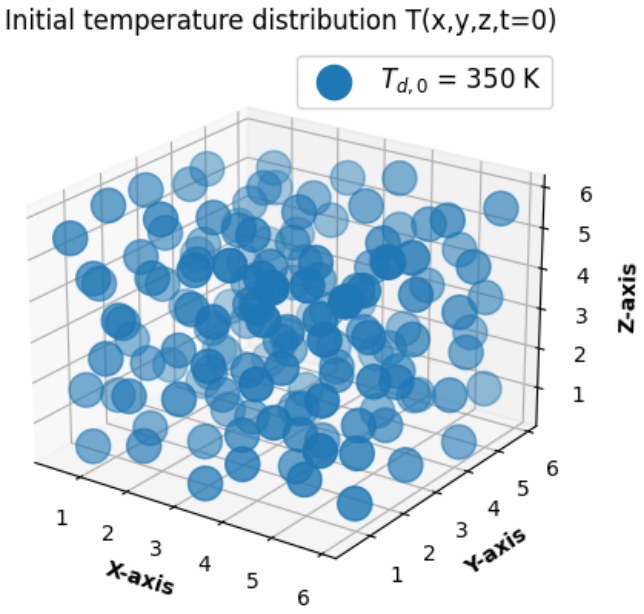


Figure 5: The initial temperature distribution of the flow, where the temperature of the dispersed "droplets" are $T_{d,0} = 350$ K and the carrier phase is $T_{c,0} = 250$ K .

Depending on the parameters governing the ABC equations, the flow eventually reaches the turbulent region, and HIT conditions are reached. Some different testing were done to find when the flow reaches turbulent conditions, and then the initialization time t_{T_0} of the temperature field was chosen. The different parameters for the initialization of the temperature field can be seen in Table 2 below.

Table 2: Table of initial parameters for the droplet-like temperature distribution.

Description	Variable	Value
Volume fraction	β	0.15
Droplet diameter	d_b	0.7854 m
Cells per diameter	ppd	32
Number of droplets	n_b	146
Initial temperature of carrier	$T_{c,0}$	250 K
Initial temperature of dispersed	$T_{d,0}$	350 K
Initial time	t_{T_0}	9.10τ

3.2.3 Assumptions

For this case study, there have been several assumptions regarding the flow. First off, the field is essentially assumed a single-phase, where the initial temperature of the dispersed phase are different from the carrier phase. The thermal properties Cp_p and κ together with the viscosity ν are assumed constant throughout the simulation. Further, gravity is neglected and the only force sustaining the flow is the ABC forcing. This is to create a simple flow study, with HIT conditions, to see whether the multiple-resolution scheme is able to accurately resolve the scalar field while minimizing the number of possible sources of error.

After the initialization of the temperature field, the VOF function is set to $\psi = 1$ for the rest of the simulation. As can be seen from Eq. 13 the convection-diffusion equation is dependent on the properties ρ, κ and Cp_p. In the simulations for the multiple-resolution method these properties have been assumed constant. However, in a multiphase scenario the properties are decided based on ψ, and it is therefore necessary to either solve the advecting VOF Eq. 9 or use interpolation from coarse to fine grid.

3.3 Numerical interpolation method

3.3.1 Staggered grid

For codes that uses a finite-volume approach when solving the turbulent flow, the domain is diveded into N control volumes or grid cells. The velocity is evaluated at the cell surface, while the scalar values are assumed an average in the middle, as can be seen in Figure 6 below.

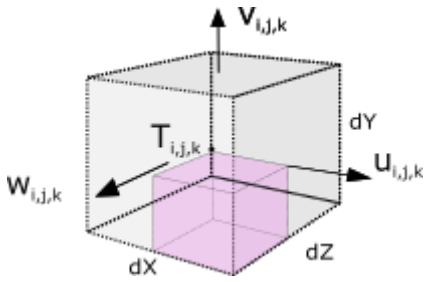


Figure 6: Control volume for dense and coarse mesh, evaluating the velocity at the surfaces and scalar value at the cell center.

This means that grid is a staggered grid with the pressure also evaluated in the center. For the flow studied in this research work, the number of grid cells have been assumed equal in every direction $N_x = N_y = N_z$ for simplicity reasons, but is not a necessary condition for the interpolation scheme.

3.3.2 Interpolation

As mentioned in Section 1, the method uses the velocity from the coarse grid to interpolate new velocities on the finer grid. For the interpolation, a third order interpolation scheme is used, where the distribution of velocity at the i, j, k surface can be expressed as

$$f(x) = \sum_{n=1}^{3} \zeta_n x^{n-1} \ , \tag{18}$$

where ζ_n is determined by applying mass-conservation at the cell surface, as well as the neighbouring surfaces. This interpolation was chosen, because a first order scheme cannot guarantee smoothness, and an even order tends to favour neighbouring grid points [6]. This means that given the three coefficients, it is necessary to evaluate three points. Looking at a velocity in a given direction gives a set of three equations, where in the case of y-direction: $\Delta X = X_i - X_{i-1}$, and the only unknowns are the coefficients:

$$\begin{aligned} V_{i-1,j,k}\Delta X &= \textstyle\sum_{n=1}^{3} \zeta_n \int_{X_{i-2}}^{X_{i-1}} x^{n-1} dx \\ V_{i,j,k}\Delta X &= \textstyle\sum_{n=1}^{3} \zeta_n \int_{X_{i-1}}^{X_{i}} x^{n-1} dx \\ V_{i+1,j,k}\Delta X &= \textstyle\sum_{n=1}^{3} \zeta_n \int_{X_{i}}^{X_{i+1}} x^{n-1} dx \end{aligned} \tag{19}$$

Figure 7 below shows the velocity $V_{i,j,k}$ in y-direction for the 2-dimensional plane $x - y$.

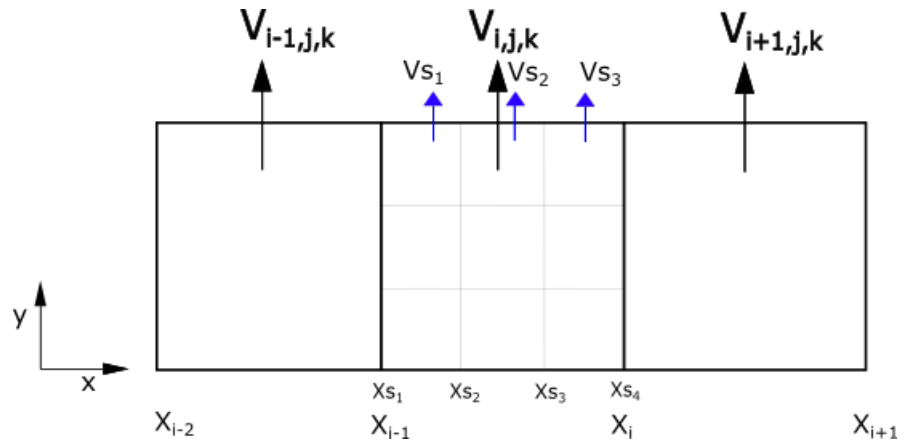


Figure 7: Shows the interpolation using the neighbouring velocities and the law of mass conservation.

Further, Eq. (19) can be reduced to a liner equation $\vec{V} = \vec{\zeta}\mathbf{A}$ where $\mathbf{A}$ is only determined by the grid spacing:

$$\begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{2}(X_{i-1} + X_{i-2}) & \frac{1}{3}(X_{i-1}^2 + X_{i-1}X_{i-2} + X_{i-2}^2) \\ 1 & \frac{1}{2}(X_i + X_{i-1}) & \frac{1}{3}(X_i^2 + X_iX_{i-1} + X_{i-1}^2) \\ 1 & \frac{1}{2}(X_{i+1} + X_i) & \frac{1}{3}(X_{i+1}^2 + X_{i+1}X_i + X_i^2) \end{bmatrix}. \tag{20}$$

This means that the coefficients can be found by $\vec{\zeta} = \mathbf{A}^{-1}\vec{V}$. The obtained velocity distribution $f(x)$ will insure that the the velocity field is divergence free, since mass conservation has already been applied as a condition.

Furthermore the refinement of the grid is determined by M, here the refinement has been assumed equal in all direction for simplicity, i.e. $M_x = M_y = M_z$, but it is not a necessary condition for the method to work [6]. The corresponding velocities Vs can be determined in the same manner, applying mass conservation over the refined surfaces. This yields (here $M = 3$):

$$\begin{aligned} Vs_1 \Delta Xs &= \sum_{n=1}^{3} \zeta_n \int_{Xs_1}^{Xs_2} x^{n-1} dx \\ Vs_2 \Delta Xs &= \sum_{n=1}^{3} \zeta_n \int_{Xs_2}^{Xs_3} x^{n-1} dx \\ Vs_3 \Delta Xs &= \sum_{n=1}^{3} \zeta_n \int_{Xs_3}^{Xs_4} x^{n-1} dx \\ &\vdots \end{aligned} \tag{21}$$

where $\Delta Xs = Xs_1 - Xs_0$, and Vs_i are the refined velocities for the $V_{i,j,k}$ surface. The number of equations are determined by M. From Eq. (21) the problem can

also be reduced to a linear equation $\vec{Vs} = \vec{\zeta}\mathbf{As}$ where $\mathbf{As}$ is only determined by the refined grid spacing. However the size of $\vec{Vs}$ is $[M \times 1]$ and the corresponding matrix $\mathbf{As}$ is $[M \times 3]$. Similar to Eq. (20) this yields:

$$\begin{bmatrix} Vs_1 \\ Vs_2 \\ Vs_3 \\ \cdots \end{bmatrix} = \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \\ \cdots \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{2}(Xs_1 + Xs_2) & \frac{1}{3}(Xs_1^2 + Xs_1Xs_2 + Xs_2^2) \\ 1 & \frac{1}{2}(Xs_2 + Xs_3) & \frac{1}{3}(Xs_2^2 + Xs_2Xs_3 + Xs_3^2) \\ 1 & \frac{1}{2}(Xs_3 + Xs_4) & \frac{1}{3}(Xs_3^2 + Xs_3Xs_4 + Xs_4^2) \\ \cdots & \cdots & \cdots \end{bmatrix}. \tag{22}$$

Thus the new interpolated velocities $\vec{Vs}$ can be found from the linear equations (20) and Eq. (22):

$$\vec{Vs} = \mathbf{As}\vec{\zeta} = \mathbf{AsA}^{-1}\vec{V} = \mathbf{R}\vec{V}\ , \tag{23}$$

where $\mathbf{R}$ is a $[M \times 3]$ matrix, only determined by the coarse and refined grid spacing. In other words the $\mathbf{R}$ matrix only needs to be calculated once, and is independent of the time-evolution of the velocity field.

3.3.3 Interior velocities

Until now, only the refined surface velocities at the control volume has been calculated . To calculate the remaining interior velocities the following procedure can be done: Defining an inner control volume for the coarse cell, where the cell is dived into M sub-cells, as can be seen in Fig. 8 below.

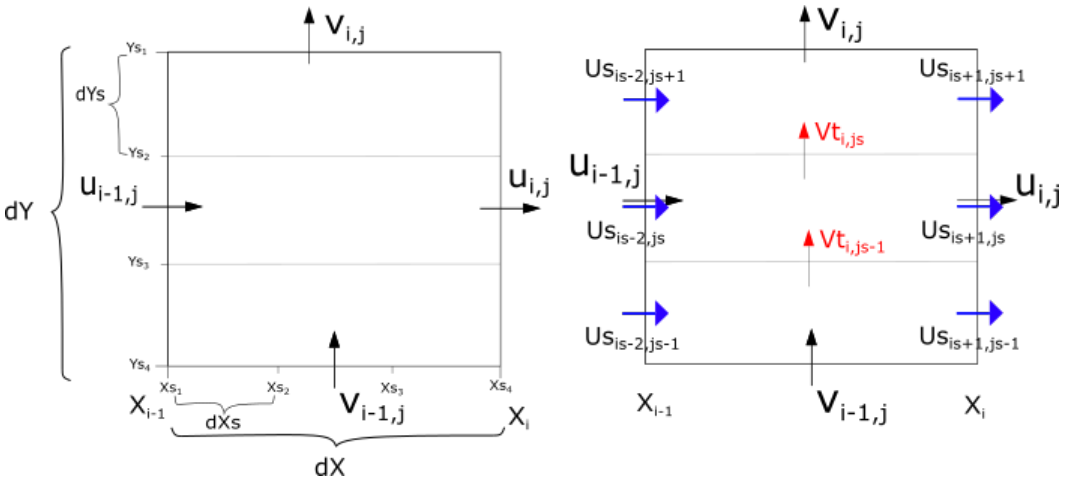


Figure 8: Shows how the coarse cell can be divided into $M = 3$ sub-cells to find the interior velocities.

Looking at the y-direction and applying conservation of mass to the top sub-cell yields the following:

$$Vt_{i,js,k} = V_{i,j,k} + (Us_{is+1,js+1,k} - Us_{is-1,js+1,k})\frac{\Delta Ys}{\Delta Y} \ , \tag{24}$$

where Us is the surface velocities in the horizontal direction and Vt is an intermediate velocity. Then the same condition can be applied for $Vt_{i,js-1,k}$. Finally, using the interpolation Equation (23), and $\vec{Vt} = [Vt_{i-1,js,k}, Vt_{i,js,k}, Vt_{i+1,js,k}]^T$ as the new velocity vector, the new interpolated interior velocities can be obtain. This procedure also secures the conservation of a divergence free field through mass conservation.

3.3.4 Scalar extrapolation

As explained in Section 3.1.3 the temperature needs to be extrapolated to the coarse field, in the case where there is an active scalar field. In finite-volume method the temperature is just the average over the grid cell, meaning that the extrapolation is purely a summation of the refined cells over the coarse grid cell.

$$T_{i,j,k} = \frac{\sum_{is=1}^{M_x}\sum_{js=1}^{M_y}\sum_{ks=1}^{M_z} Ts_{is,js,ks}}{M_x M_y M_z} \ , \tag{25}$$

where Ts represents the temperature field on the refined grid.

3.3.5 Transient interpolation

Since the scalar field is being resolved on a finer grid, the spacial increment Δx naturally has to decrease. This has a direct impact on the aforementioned CFL condition from Eq. (6). In fact, this means that the incremental time-step Δt must decrease by the same factor as the number of refinement M. However by solving the more expensive momentum and pressure equations with a larger Δt, the scalar equations can be solved separately with a smaller $\Delta t/M$ using an intermediate interpolated velocity. The linear interpolation can be expressed as [7]:

$$\vec{Vs}^{\,n+l/M} = \frac{L-l}{L}\vec{Vs}^{\,n} + \frac{l}{L}\vec{Vs}^{\,n+1} \ , \tag{26}$$

where L is the amount of sub-steps, l is the intermediate time level between t^n and t^{n+1} where the velocity $\vec{Vs}^{\,n+l/M}$ is evaluated. By using the maximum possible C_{max} when solving the momentum equations and setting $L \geq M$, ensures that the

CFL condition is conserved. Since both $\vec{V}s^{n}$ and $\vec{V}s^{n+1}$ are divergence free, the intermediate velocity $\vec{V}s^{n+l/M}$ should also be a divergence free field.

3.4 Algorithm

Since the multiphase solver is a module-based code, the algorithm developed in this research work is a module that is called in the main-loop, as can be seen in the flow chart of the code in Fig. 3. The input arguments are the 3-dimensional velocity field $\vec{V}s$, number of refinements M, number of grid cells in each direction N, length of the spatial domain L and the number of halo points nh.

The code developed takes on the assumption that the number of refinements in each direction is equal. This has been done partially to make the algorithm less complicated and also to shorten the number of necessary for-loops. However, this is not a requirement for the method to work [7].

As mentioned earlier the $\mathbf{R}$ matrix only needs to be calculated at the start of the simulation. This is done with input data from Table 1. Once the $[M \times 3]$ matrix has been calculated, this can be used throughout the simulation without being recalculated.

A pseudo-code of the developed algorithm can be seen below in Alg. 1. Efforts have been made to shorten the number of needed for-loops, however the iteration in one direction is needed to calculate the intermediate velocities $\vec{V}t$ in the same direction. Meaning the code needs at least 3 for-loops for every velocity in each direction. The full code can be seen in Appendix A.

Algorithm 1 Interpolation module

```
Input data: M, u, v, w, L, N, nh
Calculate matrix R from matrix A and As
for i in range(NI) do
    for j in range(NJ) do
        for k in range(NK) do
            Interpolate surface velocity for Us, Vs, Ws from Eq. (23)
        end for
    end for
end for
Impose boundary conditions
for k in range(NK) do
    for j in range(NJ) do
        for i in range(NI) do
            Calculate the intermediate velocity Vt using Eq. (24)
        end for
        Impose boundary conditions
        for i in range(NI) do
            Interpolate remaining velocities for Vs
        end for
    end for
end for
Impose boundary conditions
Do the same for Us and Ws
for i in range(NI) do
    for j in range(NJ) do
        for k in range(NK) do
            Interpolate Us, Vs, Ws with Eq. (23) for remaining refined grid cells
        end for
    end for
end for
Impose boundary conditions
return Us, Vs, Ws
```

The last for-loop in Alg. 1 refers to the velocities for the remaining refined grid cells. E.g. in the case of calculating the vertical velocity Vs with a refinement of $\mathsf{M} = 2$, the interpolated velocities are found in both y- and x-direction. However, the velocity is only estimated at the coarse grid center line. Thus it is necessary to interpolate to find the velocities at the fine grid cell center.

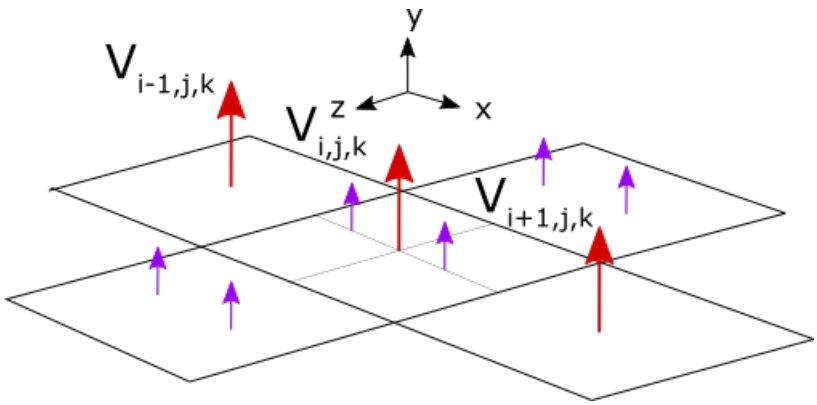


Figure 9: Interpolation of remaining velocities at the fine cell center.

Figure 9 shows how the velocities in y-direction are interpolated for every coarse cell. Thus the remaining velocities for the refined grid cells can be interpolated using the same Eq. (21) as before with the already interpolated velocities, here the neighbouring velocities in z-direction.

3.5 Verification

As verification for the interpolation scheme, the divergence free constraint on the original velocity field must also be satisfied for the new interpolated velocity field. This means that both the total and the maximum divergence of the field must be close to machine error.

When interpolating between different grids with regards to DNS it is crucial not to lose any energy, as turbulent energy is based on cascading energy down to the Kolmogorov scale η. As long as the coarse grid is able to capture the momentum equation well enough, there will be close to no loss of energy when applying interpolation. The sufficiency of the spatial grid spacing can be studied by estimating the energy spectrum, however this was not done in this research work [15]. Here, only the Kolmogorov scale was studied, by estimating the dissipation rate, given in Eq. (3).

Regarding the temperature field, the grid spacing needs to be sufficient enough to resolve the low thermal diffusivity, and thus Eq. (15) can be used to evaluate how the different grids are able to capture the evolution of the temperature field in time.

3.6 Computing system

For running the multiphase solver FluTAS, the High Perfomance Computing (HPC) system Betzy was used [10]. This is a supercomputer provided by NTNU and maintained by the Norwegian Research Infrastructure Services (NRIS). In the simulations done there were used 4 computing nodes with a total of $n = 64$ CPU cores at a clock speed of 2.25 GHz. The CPU type is AMD® Epyc™ 7742.

When measuring the total computational time of the simulation, the CPU time in this case, the MPI function *MPI_ Wtime*() is called [37]. This function evaluates the elapsed time or wall-clock time between the different subroutines called inside the main-loop.

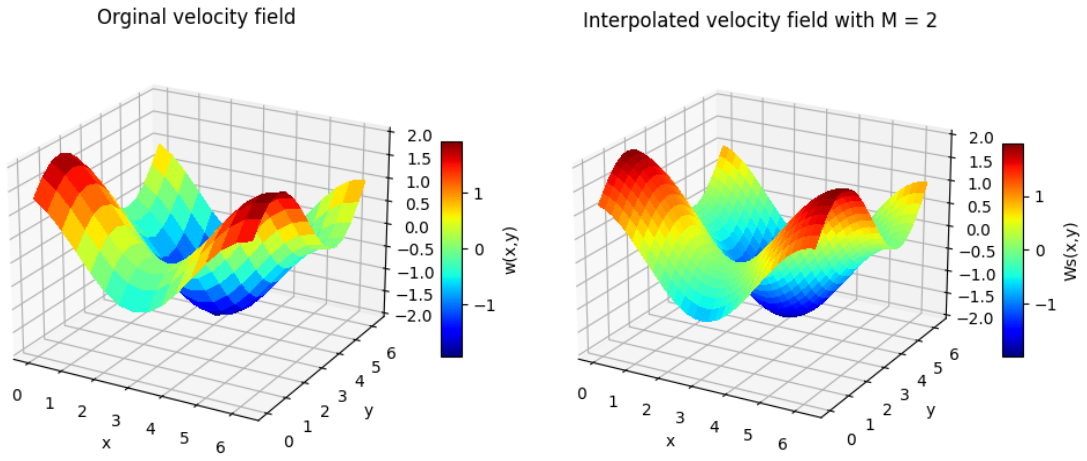
4 Results

In this section the results obtained from the code developed, using the method described in section 3.3, are given. Further the results and visualization of the different simulations are presented. Lastly the difference in computational time using different resolutions will be shown.

For the simulations discussed in the next sections, the different resolutions will be referred to as SR and MR, single and multiple-resolution respectively. For the coarse grid a total of $N^3 = 128^3$ grid cells are used, and for the finer grid $N^3 = 256^3$. The number of refinement used is in the simulations is $M = 2$. All the results presented are from the simulations on the HPC system Betzy using $n = 64$ processors (see Sec. 3.6), if not otherwise explicitly stated.

4.1 Divergence of the velocity field

For the interpolation module created, the divergence of the interpolated velocity $\vec{Vs}$ was compared to the original velocity field.



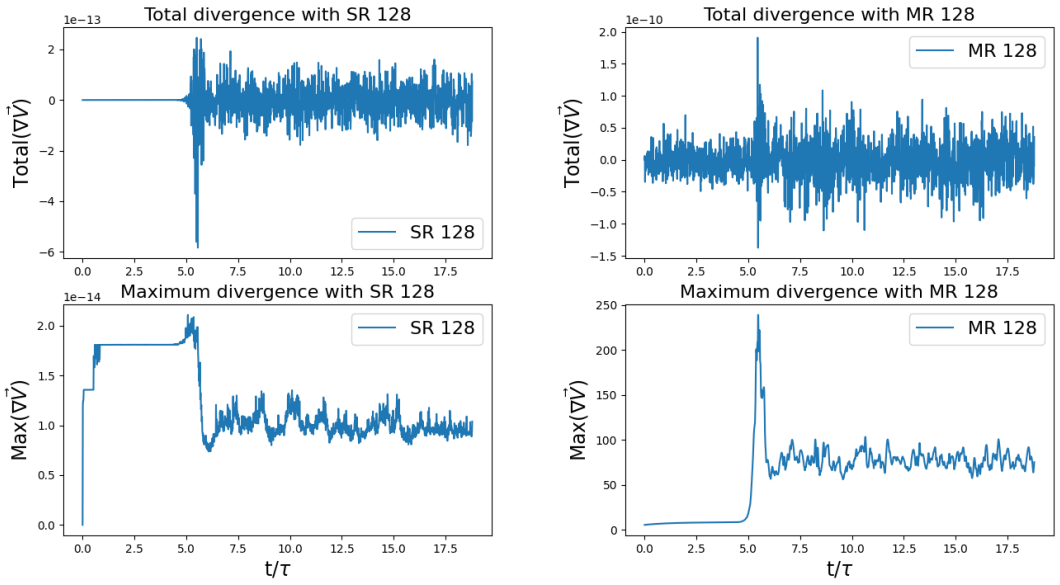
(a) $w(x, y, z = L_z/2)$ (b) $Ws(x, y, z = L_z/2)$

Figure 10: Comparison of the original velocity w to the interpolated velocity Ws using $M = 2$.

Figure 10 shows the interpolated velocity in z-direction $Ws(x, y, z = L_z/2, t = 0)$ for the initial velocity field using the created algorithm proposed in Section 3.4.

For verifying the interpolation scheme, the divergence of the new interpolated velocity field must be as close to zero as possible as well. Throughout the

simulation the divergence of both the coarse and fine velocity field was evaluated.



(a) SR 128

(b) MR 128

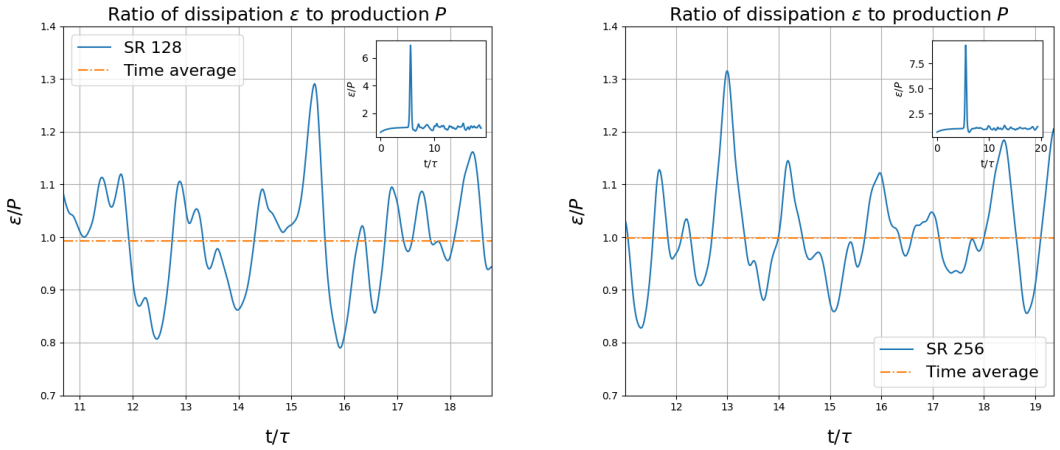
Figure 11: Comparison of the divergence of the multi resolution velocity field to original velocity field.

Figure 11 shows the maximum and the total divergence of the original and the interpolated velocity field for the total duration of the simulation.

4.2 Simulations

4.2.1 Turbulent properties

To see when the flow reaches steady state, the production to dissipation ratio was estimated as an integral over the computational domain, for the turbulent region.



(a) $\epsilon/P(t)$ for SR 128 (b) $\epsilon/P(t)$ for SR 256

Figure 12: Ratio of turbulent kinetic energy dissipation ϵ to the production P in the turbulent region.

Figure 12 above shows the turbulent kinetic energy dissipation rate ϵ to the production P given by Eq. (4) for both the coarse and the fine grid.

Table 3: Results for resolving the turbulence for the different resolutions. The values are estimated for the same time-period as shown in Fig. 12.

Properties	SR 128	MR 128	SR 256
Average dissipation $\overline{\epsilon(t)}$	0.1303 m^2/s^3	-	0.1371 m^2/s^3
Average production $\overline{P(t)}$	0.1316 m^2/s^3	-	0.1372 m^2/s^3
Average $\overline{\epsilon/P(t)}$	99.36 %	-	99.86 %
$\overline{V}_{rms}$	0.5964 m/s	-	0.6082 m/s
Kolmogorov scale η	0.0885 m	-	0.0874 m

Table 3 shows the different turbulent properties related to resolving the momentum equations properly, for different resolutions. The time average values are from the turbulent region right after the temperature field is initialized. Efforts were done to try to also calculate and study the energy spectrum using Fourier transformation for the coarse and fine grid, however due to technical problems with the code, this was not made possible as of now.

4.2.2 Temperature field

To see if the temperature field is resolved accurately, the variance of the temperature have been plotted, as described in Section 3.1.2.

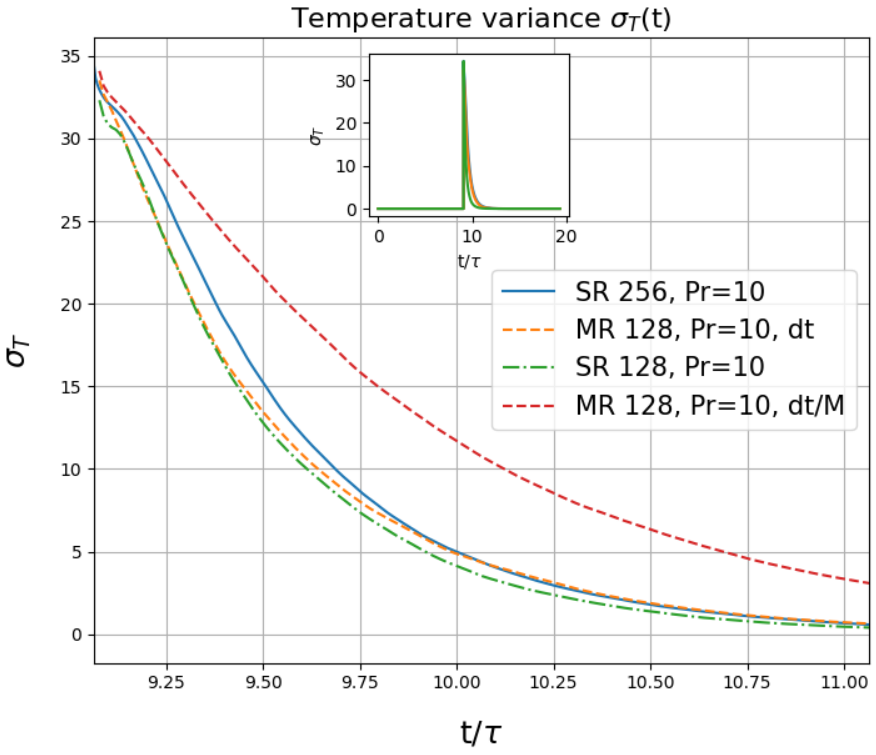


Figure 13: The temperature variance σ_T for different resolutions and Δt with $Pr = 10$.

Figure 13 shows the temperature variance after the initialization of the droplet-like temperature field. The simulations were done with the parameters given in Table 1. The $\Delta t/M$ notation refers to which Δt is used in Equation 12 for the multiple-resolution scheme. Figure 14 below shows $\sigma_T(t)$ for $Pr = 1$.

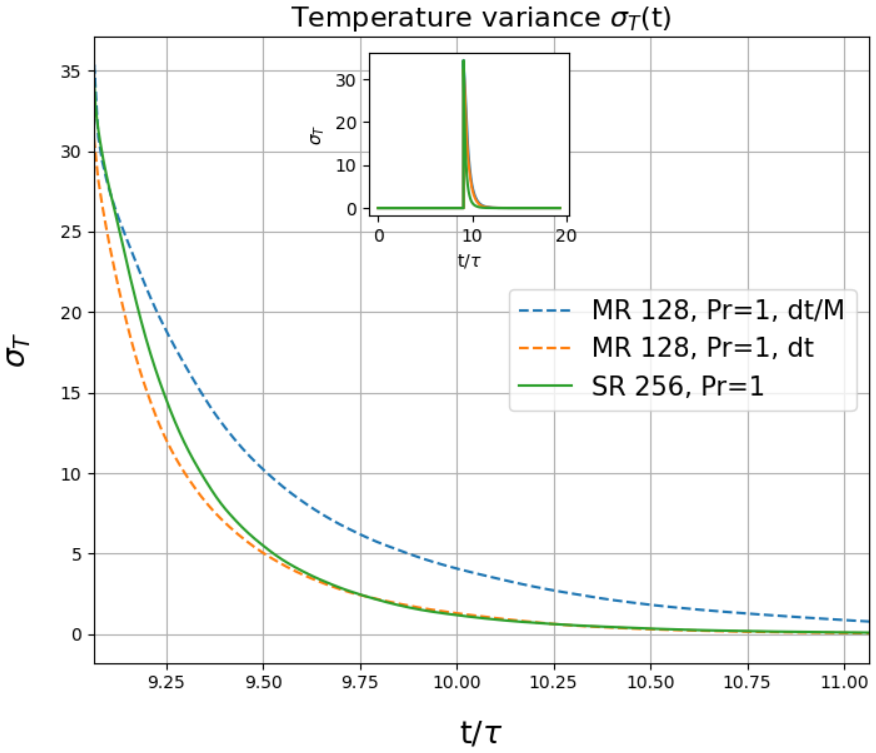


Figure 14: The temperature variance σ_T for different resolutions and Δt with $Pr = 1$.

The different results for when the temperature field reaches uniform temperature with $Pr = 10$, can be seen in Table 4.

Table 4: Results related to the temperature field for $Pr = 10$ from the different resolutions.

Property	SR 128	MR 128	SR 256
Batchelor scale λ_B	0.0280 m	0.0280 m	0.0276 m
t_{T_0}/τ	8.921	8.921	9.097
t_{T_f}/τ	13.033	13.195	13.203
$\Delta t_f/\tau$	4.113	4.275	4.107
T_f	265.313 K	265.976 K	265.092 K

Further, the average spatial temperature have been studied. Figure 15 shows the different mean temperatures after the initialization of the temperature field, with $Pr = 1$.

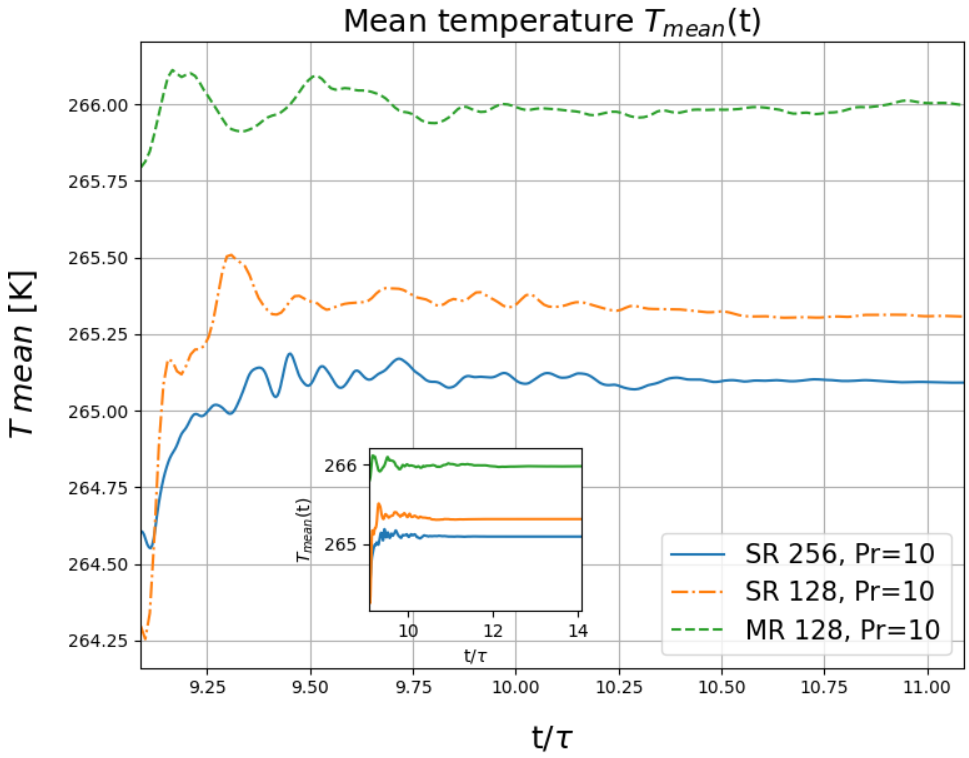


Figure 15: The mean temperature $\bar{T}$ after the initialization, for different resolutions.

The visualization of the temperature field can be seen in Figure 16. The 2-dimensional temperature field for the coarse and fine single resolution grid is compared right after the initialization of the temperature, $t = 0.092\tau$ after t_{T_0}. Efforts have been made to try to visualize the temperature field for the multiple-resolution field as well, however due to problems with the post processing section handling different grid sizes in the FluTAS code, this is not presented here.

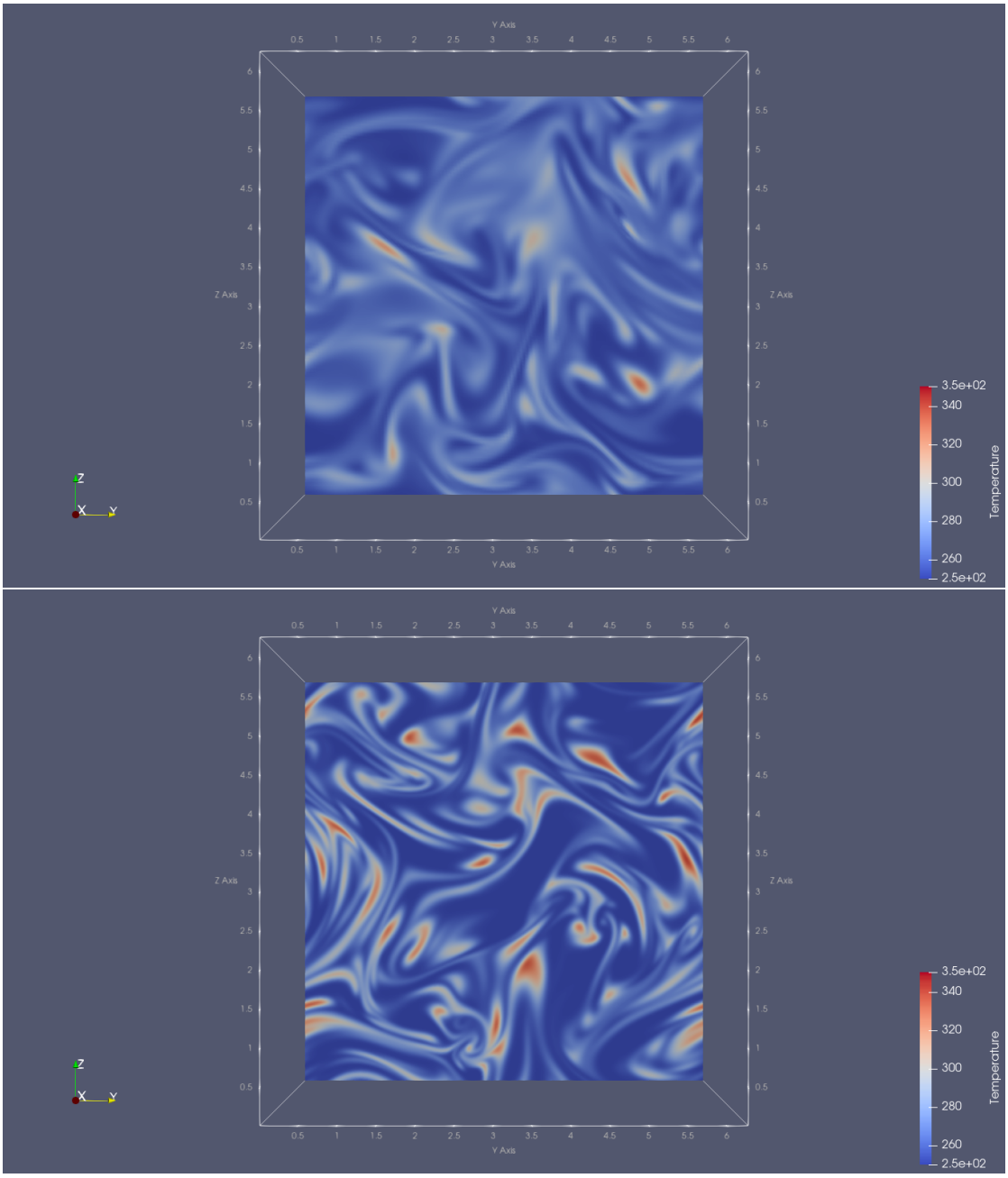


Figure 16: Shows the temperature field at $\mathrm{t} = 0.092\tau$ after the initialization. The upper plane is from SR 128 and bottom SR 256.

4.3 Computational time

The computational time was measured for the different resolutions. Figure 17 shows the different CPU wall-clock times for different parts of the code.

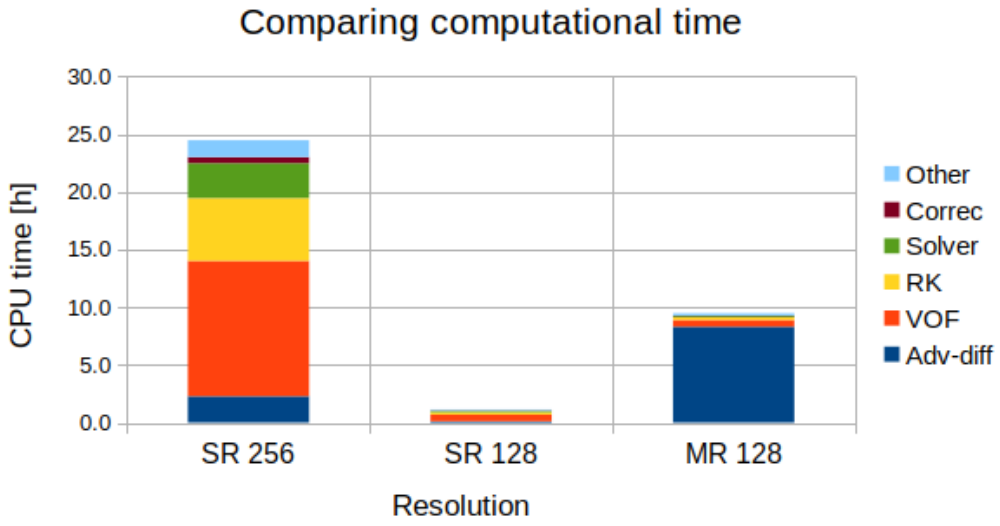


Figure 17: The CPU time for different resolutions, using the same configuration, with a refinement $M = 2$. Correc is the correction term, Solver is the Poisson eq. solver, and RK is the Runge-Kutta method used when solving the momentum equations. Adv-diff is the total time for both the interpolation and the Adams-Bashforth Eq. 12.

Here the total CPU time of both the interpolation module and the Adams-Bashforth equation have been measured for the multiple-resolution, denoted as Adv-diff.

5 Discussion

This section discusses the results found and presented in Section 4, and also some indications to what could be done differently and potential further work.

5.1 Divergence of velocity field

As discussed in Section 4, the method requires that the interpolated velocity field must also be divergence free. This is a way to verify that the interpolation method is valid, and that the conservation properties of the velocity field are not lost when interpolating from coarse to fine grid. The maximum divergence and the total divergence of the coarse grid and the multiple-resolution can be seen in Figure 11. For the coarse grid, with $N^3 = 128^3$, the integral of the divergence in the turbulent region is on the order of $O(10^{-13})$ and the maximum divergence is on the order of $O(10^{-14})$. In the region leading up to turbulent conditions, the total divergence is on the order of $O(10^{-16})$. This is what is expected, as it should be close to machine error, and is expected to be somewhat larger in the turbulent region where the velocity field is fluctuating.

When compared to the multiple-resolution method, the total divergence is of slightly higher order $O(10^{-10})$, and the maximum divergence is on the order of $O(10^{1})$ in the turbulent region. First off, the maximum divergence is clearly non-physical and is incorrect. The order should be close to machine error, as with the single resolution. However, while trying to understand the cause of this issue, the code was tested using only $n = 1$ processor. In this case, the total divergence of the interpolated velocity field was found to be on the order of $O(10^{-16})$ and the maximum divergence of $O(10^{-6})$ at the start of the simulation. Due to time-limitation, the code was stopped before it could reach turbulent conditions.

It seems that the problem is partially connected to the parallelization of the code. The order of magnitude for the divergence seems to be related to the number of processors n the code utilises when simulating. Worth taking into consideration is how the divergence is computed for the finer velocity field, and could also be a potential source of error. However, the order of maximum divergence is still substantially higher than what is expected, even when using only $n = 1$ processor. Previous work suggests that maximum divergence of order $O(10^{-17})$ can be achieved [6]. This suggests that there might be something incorrect with both the implementation of the module, and the parallelization of the code. Efforts have been made to try to investigate these errors, however, the time-limitation of this work has restricted further investigations.

5.2 Simulations

5.2.1 Resolving the momentum equations

For evaluating the HIT conditions, the ratio of dissipation rate ϵ and production P can be studied. At statistical steady state production to dissipation should balance in time. As can be seen in Fig. 12, the dissipation to production ratio for the coarse grid is oscillating around 0.9936 in the turbulent region. The difference in dissipation can be seen in Table 3, for the coarse and dense grid spacing. In this case, the temperature field is assumed a passive scalar field, and resolving the momentum equation should yield the same as for the multiple-resolution and the single resolution on the coarse grid. This is indeed the case, as Tab. 3 shows.

For the mutliple-resolution method to be viable, it is crucial that the base grid is able to capture and resolve the momentum equations accurately. The Kolmogorov scale was found to be $\eta = 0.0874$ m for the SR 256, and since the grid spacing for the coarse field is $\Delta x = 0.0491$ m, the base grid should be able to resolve the turbulence accurately. To further verify the sufficiency of the resolution, the energy spectrum of the turbulence can be estimated and studied. As previously mentioned, this was not done in this study, but could readily be implemented to the code to compare the different resolutions. The rest of the discussion takes on the assumption that the estimated Kolmogorov scale is indicative of the adequacy of the resolution.

5.2.2 Resolving the temperature field

To verify that the multiple-resolution method is able to resolve the scalar field accurately, and quantify the degree of mixing in the flow, the time evolution of the temperature variance can be studied. The variance of the temperature field $\sigma_T(t)$ can be estimated using Eq. (15). Figure 13 and 14 shows the different time evolutions with both different resolutions and different Prandtl numbers. As expected, the gradient of the temperature variance $\partial\sigma_T(t)/\partial t$ becomes steeper with lower Pr, because of the increased thermal diffusivity which causes increased mixing.

Further, it can be observed that with the lower resolution, SR 128, the grid is not able to capture the change in $\sigma_T(t)$ compared to the higher resolution SR 256. Due to the sharp gradients of the temperature field, the lower resolution is not small enough to capture the rapid change, as can be seen in the visual comparison in Fig. 16. This is also to be expected, as the coarse grid spacing $\Delta x = 0.0491$ m is larger than the Batchelor scale, which was found to be $\lambda_B = 0.0276$ m with $Pr = 10$. As the fine grid spacing here is $\Delta Xs = 0.0245$ m, the higher resolution should be able to capture the temperature field accurately. However, the fine grid

spacing, SR 256, has not been compared to other studies or higher resolutions, and is assumed to be correctly resolving the temperature. Comparing to an even higher resolution is something that could be considered for future work, to further verify the accuracy.

For the multiple-resolution method, two different approaches have been compared. The first one uses $\Delta t/M$ in the Adams-Bashforth Eq. (12), which is following the method described in Section 3.3.5. This is expected to be the same as the SR 256 solution. However, the results show a much slower diffusion than the higher resolution SR 256, as can be seen by the $\sigma_T(t)$ gradient. Compared to simulations with $Pr = 1$ the base grid should be small enough to capture both the momentum and temperature field accurately. As Fig. 14 shows, the multiple-resolution overestimates the temperature variance, i.e the method estimates a slower thermal diffusion compared to the higher resolution.

To investigate the problem of underestimating thermal diffusivity, a second approach was tested, using the original Δt from the coarse grid. Since Δt should be refined along with the refined velocity field, it is expected that this approach should give an inaccurate result. Although this yields a more accurate diffusion of the temperature field, it seems to not be able to capture the $\sigma_T(t)$ as well as the higher resolution, even with $Pr = 1$.

This suggests that the potential source of error is related to how the transient interpolation from Eq. (26) is implemented together with the Adams-Bashforth method. Since the Batchelor scale λ_B is given by Eq. (7), the base grid in this case is already small enough to capture the scalar field. Therefore, a refinement of Δt should not change the diffusivity captured by the resolution. The results discussed regarding the error in magnitude of the divergence, could likely also be a source of error in the time evolution of $\sigma_T(t)$.

Due to how the parameter of the initialization of the temperature is set, the initialization time t_{T_0} is slightly different for the high resolution SR 256, as seen in Tab. 4. However, the difference of around $0.176\ \tau$ should not heavily affect the difference in the advection of the temperature field.

Lastly the mean temperature $\overline{T(t)}$ was studied, as shown in Fig. 15. The time evolution of $\overline{T(t)}$ for the different resolutions can be observed, where the multiple-resolution scheme has a slightly higher mean temperature, although small difference in percentage. The mean temperature should be constant throughout the simulation. Although there seem to be some slight variations immediately after the initialization, the MR 128 conserves the mean temperature after some initial adjustments.

5.3 Computational performance

To estimate how the multiple-resolution method performance compared to the single resolution, the total computational time for the different resolutions was measured. Here the total CPU wall-clock time was measured, for different sections of the code, (see Fig. 17). The results are from the simulations with $\mathsf{Pr} = 10$ and show that the total CPU time for the main-loop for SR 256 was 24.47 h. For the multiple-resolution scheme, the total CPU time was measured to 9.47 h. This is approximately a reduction of 61.30 %. The adv-diff CPU time represents both the interpolation and the Adams-Bashforth equation for the multiple-resolution method. In MR 128 this was measured to be 87.79 % of the total CPU time.

Previous results with multiple-resolution scheme simulating Rayleigh-Benard flow suggest a computational speed-up of a factor of 7 with $\mathsf{Pr} = 10$ [7]. Here it is worth mentioning that most of the computational time cost for the solvers used are the momentum and Poisson solver. The results suggest that there might be some improvements to be made, and that further reduction in computational time could be possible. The reduction of the RAM memory usage is something that would also be worth measuring but was not done in this study.

As mentioned in Section 3.2.3, the properties Cp_P, ρ and κ are assumed constant for the multiple-resolution scheme. Thus, there is no need to advect the VOF function ψ on the fine grid. As can be seen by the results, solving the VOF equations is computationally expensive, whereas for the single resolution case it was measured to be 47.93 % of the total CPU time. In a multiphase flow, there will be a need for interpolating these properties, thus making the multiple-resolution method slightly more computationally expensive. However, by avoiding solving the VOF equations on the finer grid there is still substantial CPU time to be saved.

Though the simulations are not fully representative of a multiphase flow, the comparison between single and multiple-resolution is still valid in the case of a passive scalar field. Assuming that the source of error lies with the implementation of the transient interpolation, the possible fix to this error will not heavily affect the required computational time, provided that the momentum and pressure equations are resolved correctly on the base grid. The results are not an accurate representation of the saved CPU time but is an indication of how much can potentially be saved.

5.4 Challenges

Due to the nature of this work, there have been several challenges along the way, such as familiarizing with Fortran programming language, the multiphase solver

FluTAS and connecting to the HPC system Betzy. Much of the time has been dedicated to understanding how the FluTAS code is structured, and how the different sections are organized. Large portions of the problems faced involve how the MPI and parallelization work together. Several different simulations that are not included here have been done with efforts to try to understand the aforementioned errors without success.

6 Conclusions

The research question in this work is to see if a multiple-resolution scheme using an interpolation method can be used to significantly reduce the computational time needed to resolve scalar transport in a turbulent flow with low scalar diffusivity. To test the interpolation method proposed, the algorithm developed was implemented in the multiphase solver FluTAS. Several simulations using the high-performance computing system Betzy were done to test the accuracy of the multiple-resolution grid compared to a single resolution. The interpolation method has been tried verified by computing the divergence of the interpolated velocity field. The total divergence was found to be on an order of $O(10^{-10})$. The maximum divergence was found to be $O(10^{1})$, and seems to vary with the number of processors n used in the simulation, which likely implies a source of error related to the implementation of the MPI parallelization.

From the simulations carried out, the results show that although the multiple-resolution is able to substantially reduce the CPU time with 61.30 %, the accuracy of the method still yields inaccurate results for resolving the passive scalar field. Further work investigating the implementation of the transient interpolation is suggested, as although the method proves to be inaccurate, the comparison of computational time to single resolution is still representative of the possible reduction in CPU time. On the assumption that a correct implementation of the interpolation method yields an accurate solution of the scalar field, there is promising potential in reducing the necessary computational time substantially.

7 Further work

As discussed in Section 5.1 the maximum divergence of the interpolated velocity field is clearly incorrect, which suggests that further investigations of the interpolation scheme is needed, to understand what causes this behaviour. The results involving the evolution of the temperature field suggest that the transient linear interpolation is not incorporated correctly, and for the method to work correctly further investigation needs to be done regarding this implementation.

Although the results discussed in 5.2.2 shows inaccuracy compared to the single resolution, the comparison in computational time for the different resolutions still shows promising results regarding the potentially saved CPU time, on the assumption, based on previous work done, that the multiple-resolution correctly implemented will resolve the scalar field accurately. The study of how different Prandtl numbers affect the solution of the multiple-resolution method would be interesting to further investigate. The computational investigation of implementing this method to a multiphase solver is of high interest as it could lead to drastically reducing the computational time needed. The further development of this interpolation method could open up possibilities to save substantial computational time for DNS of scalar fields in turbulence.

References

[1] Eberhard Bodenschatz and Michael Eckert. A voyage through turbulence. pages 40 – 100, 2011.

[2] Kai Velten, William Lubitz, and Alwin Hopf. *Simulation of airflow within horticulture high-tunnel greenhouses using open-source CFD software*. PhD thesis, 02 2018.

[3] E. Lanti and et al. Orb5: a global electromagnetic gyrokinetic code using the pic approach in toroidal geometry. 2019.

[4] Ming Li and Chris Garret. The relationship between oil droplet size and upper ocean turbulence. *Marine Pollution Bulletin*, pages 961–970, 1998.

[5] R. Bessaih and M. Kadja. Turbulent natural convection cooling of electronic components mounted on a vertical channel. *Applied Thermal Engineering*, 20(2):141–154, 2000.

[6] Kai Leong Chong, Guangyu Ding, and Ke-Qing Xia. Multiple-resolution scheme in finite-volume code for active or passive scalar turbulence. *Journal of Computational Physics*, 09 2018.

[7] R.Ostilla-Monicoa, Yantao Yanga, E.P.van der Poela, D. Lohsea, and R.Verzicco. A multiple-resolution strategy for direct numerical simulation of scalar turbulence. *Journal of Coputational Physics*, 11 2015.

[8] Tommy Hellen. Interpolation scheme for dns of scalar fields in turbulence. 2021.

[9] Marco Crialesi-Esposito, Nicolo Scapin, Andreas D. Demou, Marco Edoardo Rosti, Pedro Costa, Filippo Spiga, and Luca Brandt. Flutas: A gpu-accelerated finite difference code for multiphase flows. 2022.

[10] Sigma2. Betzy. `https://documentation.sigma2.no/hpc_machines/betzy.html`, 2022.

[11] O. Podvingina and A. Pouquet. On the non-linear stability of the 1:1:1 abc flow. pages 471–508, 1994.

[12] A. Kolmogorov. The local structure of turbulence in incompressible viscous fluid for very large reynolds' numbers. *Akademiia Nauk SSSR Doklady*, 30:301–305, January 1941.

[13] John G. Heywood, Masuda Kyuya, R. Rautmann, Vsevolod, and A. Solon-

nikov. *Theory Of The Navier-stokes Equations.* Number v. 47 in Series on Advances in Mathematics for Applied Sciences. World Scientific, 1998.

[14] M. T. Landahl and E. Mollo-Christensen. *Turbulence and Random Processes in Fluid Mechanics.* Cambridge University Press, 1992.

[15] J. O. Hinze. *Turbulence.* McGraw-Hill, 1975.

[16] S.B Pope. Turbulent flows. pages 112–134, 2000.

[17] H. Lewy R. Courant, K. Friedrichs. On the partial difference equations of mathematical physics. *IBM Journal of Research and Development*, 11(2):215–234, 1967.

[18] B. Shraiman and E.D. Siggia. Scalar turbulence.

[19] Parviz Moin and Krishnan Mahesh. Direct numerical simulation: A tool in turbulence research. *Annual Review of Fluid Mechanics*, 30(1):539–578, 1998.

[20] Z. Warhaft. Passive scalars in turbulent flows. *Annual Review of Fluid Mechanics*, 32(1):203–240, 2000.

[21] Antonio Celani, Massimo Cencini, Andrea Mazzino, and Massimo Vergassola. Active versus passive scalar turbulence. *Physical review letters*, 89(23):234502/4–234502, 2002.

[22] G. K. Batchelor. Small-scale variation of convected quantities like temperature in turbulent fluid part 1. general discussion and the case of small conductivity. *Journal of Fluid Mechanics*, 5(1):113–133, 1959.

[23] Scott A. Socolofsky and Gerhard H. Jirka. Environmental fluid mechanicspart i: Mass transfer and diffusion. 2002.

[24] Verma, Siddhartha, and G. Blanquart. On filtering in the viscous-convective subrange for turbulent mixing of high schmidt number passive scalars. *Physics of Fluids*, 25(5):55–104, 2013.

[25] Christopher E. Brennnen. Fundamentals of multiphase flow. 2005.

[26] D. J. Clements. *Food emulsions: principles, practices, and techniques.* CRC press, 2015.

[27] Sunil Kokal. Crude oil emulsions: A state-of-the-art review. 2005.

[28] Marco Crialesi-Esposito, Marco Edoardo Rosti, Sergio Chibbaro, and Luca Brandt. Modulation of homogeneous and isotropic turbulence in emulsions. *Journal of Fluid Mechanics*, 2022.

[29] Ii, Satoshi, Sugiyama, Kazuyasu, Takeuchi, Shintaro, Takagi, Shu, and Xiao Feng Matsumoto, Yoichiro. An interface capturing method with a continuous function: The thinc method with multi-dimensional reconstruction. *Journal of Computational Physics*, pages 2328–2358, 2012.

[30] B.R. Munson, A.P. Rothmayer, and T.H. Okiishi. *Fundamentals of Fluid Mechanics, 7th Edition*. Wiley, 2012.

[31] Pedro Costa. A fft-based finite-difference solver for massively-parallel direct numerical simulations of turbulent flows. *Computers and mathematics with applications*, pages 1853–1862, 2018.

[32] Radyadour Kh. Zeytounian. Joseph boussinesq and his approximation: a contemporary view. *Comptes Rendus Mécanique*, 331:575–586, 2003.

[33] E. Harier, G. Wanner, and S.P. Nørsett. *Solving Ordinary Differential Equations I: Nonstiff problems*. Springer Berlin, 1993.

[34] Patrick Tabeling. *Introduction to Microfluidics*.

[35] S Laizet and N. Li. 2decompfft - a highly scalable 2d decomposition library and fft interface. 2010.

[36] Uriel Frisch. *Turbulence: The Legacy of A.N. Kolmogorov*. 11 1995.

[37] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

A Interpolation scheme algorithm

```
subroutine interpolation(nx,ny,nz,Mx,Lx,Ly,Lz,u,v,w,Us,Vs,Ws,nh_d,nh_ds)
  !
  implicit none
  !
  integer, intent(in ) :: nx,ny,nz,Mx,nh_d,nh_ds
  real(rp), intent(in ) :: Lx,Ly,Lz
  real(rp), dimension(0:2) :: L
  integer, dimension(0:2) :: N
  real(rp), dimension(0:2) :: dli
  real(rp), dimension(1-nh_d:ny+nh_d,1-nh_d:nx+nh_d,1-nh_d:nz+nh_d):: u,v,w
  real(rp), dimension(1-nh_d:ny*Mx+nh_d,
                      1-nh_d:nx*Mx+nh_d,
                      1-nh_d:nz*Mx+nh_d) :: Us,Vs,Ws
  real(rp) :: dX,dY,dZ,dXs,dYs,dZs,divtot,divmax
  double precision, dimension(0:nz*Mx+2*nh_d) :: dzfs
  !For calculating A and As
  real(rp), dimension(0:Mx-1,0:2) :: R_i
  real(rp), dimension(0:2,0:2) :: A = 0, A_inv = 0
  real(rp), dimension(0:Mx-1,0:2) :: As
  real(rp), dimension(0:2,0:0) :: U_i,V_i,W_i
  !Numerical parameters
  integer :: xl,yl,zl,i,j,k,NI,NJ,NK,NIs,NJs,NKs,
            pp,pm,ip,jp,kp,p_i,ik,i_s,j_s,k_s
  !Miscellaneous
  real(rp), dimension(0:Mx-2,0:nx+1) :: Vt_j, Wt_j
  real(rp), dimension(0:Mx-2,0:ny+1) :: Ut_j
  real(rp) :: Xi,Xm,a11,a12,a13,a21,a22,a23,a31,a32,a33
  !
  L = (/Lx,Ly,Lz/)
  N = (/nx,ny,nz/)
  dli = N(:)/L
  dX = L(0)/N(0)
  dY = L(1)/N(1)
  dZ = L(2)/N(2)

  dXs = dX/Mx
  dYs = dY/Mx
  dZs = dZ/Mx
  dzfs =  dZs
  NI = N(0)+1
  NJ = N(1)+1
  NK = N(2)+1
  NIs = size(Vs(0,0:,0))-1
  NJs = size(Vs(0:,0,0))-1
  NKs = size(Vs(0,0,0:))-1
  !
```

```
47      !Step 1 - Solving the R matrix
48      do i=1,1
49          A(:,0) = 1
50          As(:,0) = 1
51          do j=-1,1
52              Xi = (i+j+1)*dX
53              Xm = (i+j)*dX
54              A(j+1,1) = 0.5*(Xi + Xm)
55              A(j+1,2) = 0.33333333*(Xi**2 + Xi*Xm + Xm**2)
56          end do
57          do j=0,Mx-1
58              ik = i*Mx+j
59              Xi = (ik+1)*dXs
60              Xm = (ik)*dXs
61              As(j,1) = 0.5*(Xi + Xm)
62              As(j,2) = 0.33333333*(Xi**2 + Xi*Xm + Xm**2)
63          end do
64
65          !Calculating the inversion of the 3x3 A matrix
66          a11 = (A(1,1)*A(2,2) - A(1,2)*A(2,1))/(A(0,0)*A(1,1)*A(2,2)
67                - A(0,0)*A(1,2)*A(2,1) - A(0,1)*A(1,0)*A(2,2) + A(0,1)*A(1,2)*A(2,0)
68                + A(0,2)*A(1,0)*A(2,1) - A(0,2)*A(1,1)*A(2,0))
69          a12 = ...
70          !
71          A_inv = reshape( [ a11, a21, a31, &
72                             a12, a22, a32, &
73                             a13, a23, a33], [3,3])
74          R_i = matmul(As,A_inv)
75      end do
76      !Step 2 - Finding cell-surface velocities
77      do k=1,NK-1
78          zl = k*Mx
79          do j=1,NJ-1
80              yl = j*Mx
81              do i=1,NI-1
82                  xl = i*Mx
83                  V_i = reshape([ v(j,i-1,k) , v(j,i,k) , v(j,i+1,k)],[3,1])
84                  Vs(yl,1+(i-1)*Mx:i*Mx,zl) = reshape(matmul(R_i,V_i), [Mx])
85                  U_i = reshape([ u(j-1,i,k) , u(j,i,k) , u(j+1,i,k)],[3,1])
86                  Us(1+(j-1)*Mx:j*Mx,xl,zl) = reshape(matmul(R_i,U_i), [Mx])
87                  W_i = reshape([ w(j,i-1,k) , w(j,i,k) , w(j,i+1,k)],[3,1])
88                  Ws(yl,1+(i-1)*Mx:i*Mx,zl) = reshape(matmul(R_i,W_i), [Mx])
89              end do
90          end do
91          !Setting BC
92          !
93          Us(:,1-nh_d:0,zl)          = Us(:,NIs-2*nh_d+1:NIs-nh_d,zl)
94          Us(:,NIs-nh_d+1:NIs,zl) = Us(:,1:nh_d,zl)
95          Us(1-nh_d:0,:,zl)          = Us(NJs-2*nh_d+1:NJs-nh_d,:,zl)
```

```
96          Us(NJs-nh_d+1:NJs,:,zl) = Us(1:nh_d,:,zl)
97          !
98          Vs(:,1-nh_d:0,zl)       = Vs(:,NIs-2*nh_d+1:NIs-nh_d,zl)
99          Vs(:,NIs-nh_d+1:NIs,zl) = Vs(:,1:nh_d,zl)
100         Vs(1-nh_d:0,:,zl)       = Vs(NJs-2*nh_d+1:NJs-nh_d,:,zl)
101         Vs(NJs-nh_d+1:NJs,:,zl) = Vs(1:nh_d,:,zl)
102         !
103         Ws(:,1-nh_d:0,zl)       = Ws(:,NIs-2*nh_d+1:NIs-nh_d,zl)
104         Ws(:,NIs-nh_d+1:NIs,zl) = Ws(:,1:nh_d,zl)
105         Ws(1-nh_d:0,:,zl)       = Ws(NJs-2*nh_d+1:NJs-nh_d,:,zl)
106         Ws(NJs-nh_d+1:NJs,:,zl) = Ws(1:nh_d,:,zl)
107         !
108     end do
109     !
110     Vs(:,:,1-nh_d:0) = Vs(:,:,NKs-2*nh_d+1:NKs-nh_d)
111     Vs(:,:,NKs-nh_d+1:NKs) = Vs(:,:,1:nh_d)
112     Us(:,:,1-nh_d:0) = Us(:,:,NKs-2*nh_d+1:NKs-nh_d)
113     Us(:,:,NKs-nh_d+1:NKs) = Us(:,:,1:nh_d)
114     Ws(:,:,1-nh_d:0) = Ws(:,:,NKs-2*nh_d+1:NKs-nh_d)
115     Ws(:,:,NKs-nh_d+1:NKs) = Ws(:,:,1:nh_d)
116     !
117     !Step 3 - Finding intermediate velocities for V
118     do k=1,NK-1
119         zl = k*Mx
120         do j=1,NJ-1
121             do i=1,NI-1
122                 if(modulo(Mx,2)==0) then
123                     pp = int(Mx/2)
124                     pm = pp
125                     ip = Mx*i+pp-2
126                     jp = Mx*j+pp-2
127                 else
128                     pp = int((Mx-1)/2)+1
129                     pm = int((Mx-1)/2)
130                     ip = Mx*i+pm-1
131                     jp = Mx*j+pm-1
132                 end if
133                 do p_i=0,Mx-2
134                     if(p_i==0) then
135                         Vt_j(Mx-2,i) = v(j,i,k) - (Us(jp+pm,ip-pp,zl)
136                                                  - Us(jp+pm,ip+pm,zl))*dYs/dX
137                     else
138                         Vt_j(Mx-2-p_i,i) = Vt_j(Mx-1-p_i,i)
139                                          - (Us(jp+pm-p_i,ip-pp,zl)
140                                          - Us(jp+pm-p_i,ip+pm,zl))*dYs/dX
141                     end if
142                 end do
143             end do
144             Vt_j(:,0) = Vt_j(:,size(Vt_j(0,:))-2)
```

```
145             Vt_j(:,size(Vt_j(0,:))-1) = Vt_j(:,1)
146             !Step 4 - Finding remaining velocities for V
147             do i=1,NI-1
148                 if(modulo(Mx,2)==0) then
149                     pp = int(Mx/2)+1
150                     pm = int(Mx/2)-1
151                     ip = Mx*i+pm-1
152                     jp = Mx*j+pm-1
153                 else
154                     pp = int((Mx-1)/2)+1
155                     pm = int((Mx-1)/2)
156                     ip = Mx*i+pm-1
157                     jp = Mx*j+pm-1
158                 end if
159                 do p_i=0,Mx-2
160                     Vs(jp-pm+p_i,ip-pm:ip+pp-1,zl) = matmul(R_i,Vt_j(p_i,i-1:i+1))
161                 end do
162             end do
163         end do
164         Vs(:,1-nh_d:0,zl)          = Vs(:,NIs-2*nh_d+1:NIs-nh_d,zl)
165         Vs(:,NIs-nh_d+1:NIs,zl) = Vs(:,1:nh_d,zl)
166         Vs(1-nh_d:0,:,zl)          = Vs(NJs-2*nh_d+1:NJs-nh_d,:,zl)
167         Vs(NJs-nh_d+1:NJs,:,zl) = Vs(1:nh_d,:,zl)
168         !
169     end do
170     !
171     do k=1,NK-1
172        zl = k*Mx
173        do i=1,NI-1
174            do j=1,NJ-1
175                if(modulo(Mx,2)==0) then
176                    pp = int(Mx/2)
177                    pm = pp
178                    ip = Mx*i+pp-2
179                    jp = Mx*j+pp-2
180                else
181                    pp = int((Mx-1)/2)+1
182                    pm = int((Mx-1)/2)
183                    ip = Mx*i+pm-1
184                    jp = Mx*j+pm-1
185                end if
186                do p_i=0,Mx-2
187                    if(p_i==0) then
188                        Ut_j(Mx-2,j) = u(j,i,k) - (Vs(jp-pp,ip+pm,zl)
189                                                 - Vs(jp+pm,ip+pm,zl))*dXs/dY
190                    else
191                        Ut_j(Mx-2-p_i,j) = Ut_j(Mx-1-p_i,j)
192                                         - (Vs(jp-pp,ip+pm-p_i,zl)
193                                         - Vs(jp+pm,ip+pm-p_i,zl))*dXs/dY
```

```
194                 end if
195             end do
196         end do
197         Ut_j(:,0) = Ut_j(:,size(Ut_j(0,:))-2)
198         Ut_j(:,size(Ut_j(0,:))-1) = Ut_j(:,1)
199         !Step 4 - Finding remaining velocities for U
200         do j=1,NJ-1
201             if(modulo(Mx,2)==0) then
202                 pp = int(Mx/2)+1
203                 pm = int(Mx/2)-1
204                 ip = Mx*i+pm-1
205                 jp = Mx*j+pm-1
206             else
207                 pp = int((Mx-1)/2)+1
208                 pm = int((Mx-1)/2)
209                 ip = Mx*i+pm-1
210                 jp = Mx*j+pm-1
211             end if
212             do p_i=0,Mx-2
213                 Us(jp-pm:jp+pp-1,ip-pm+p_i,zl) = matmul(R_i,Ut_j(p_i,j-1:j+1))
214             end do
215         end do
216     end do
217     Us(:,1-nh_d:0,zl)          = Us(:,NIs-2*nh_d+1:NIs-nh_d,zl)
218     Us(:,NIs-nh_d+1:NIs,zl) = Us(:,1:nh_d,zl)
219     Us(1-nh_d:0,:,zl)          = Us(NJs-2*nh_d+1:NJs-nh_d,:,zl)
220     Us(NJs-nh_d+1:NJs,:,zl) = Us(1:nh_d,:,zl)
221 end do
222 !
223 Vs(:,:,1-nh_d:0)          = Vs(:,:,NKs-2*nh_d+1:NKs-nh_d)
224 Vs(:,:,NKs-nh_d+1:NKs) = Vs(:,:,1:nh_d)
225 Us(:,:,1-nh_d:0)          = Us(:,:,NKs-2*nh_d+1:NKs-nh_d)
226 Us(:,:,NKs-nh_d+1:NKs) = Us(:,:,1:nh_d)
227 !
228 !Step 5 - Remaining sub-cell velocities for V and U
229 do k=1,NK-1
230     zl = k*Mx
231     do j=1,NJ-1
232         do i=1,NI-1
233             if(modulo(Mx,2)==0) then
234                 pp = int(Mx/2)
235                 pm = int(Mx/2)
236                 ip = Mx*i+pm-1
237                 jp = Mx*j+pm-1
238                 kp = Mx*k+pm-1
239             else
240                 pp = int(Mx/2)+1
241                 pm = int(Mx/2)
242                 ip = Mx*i+pm-1
```

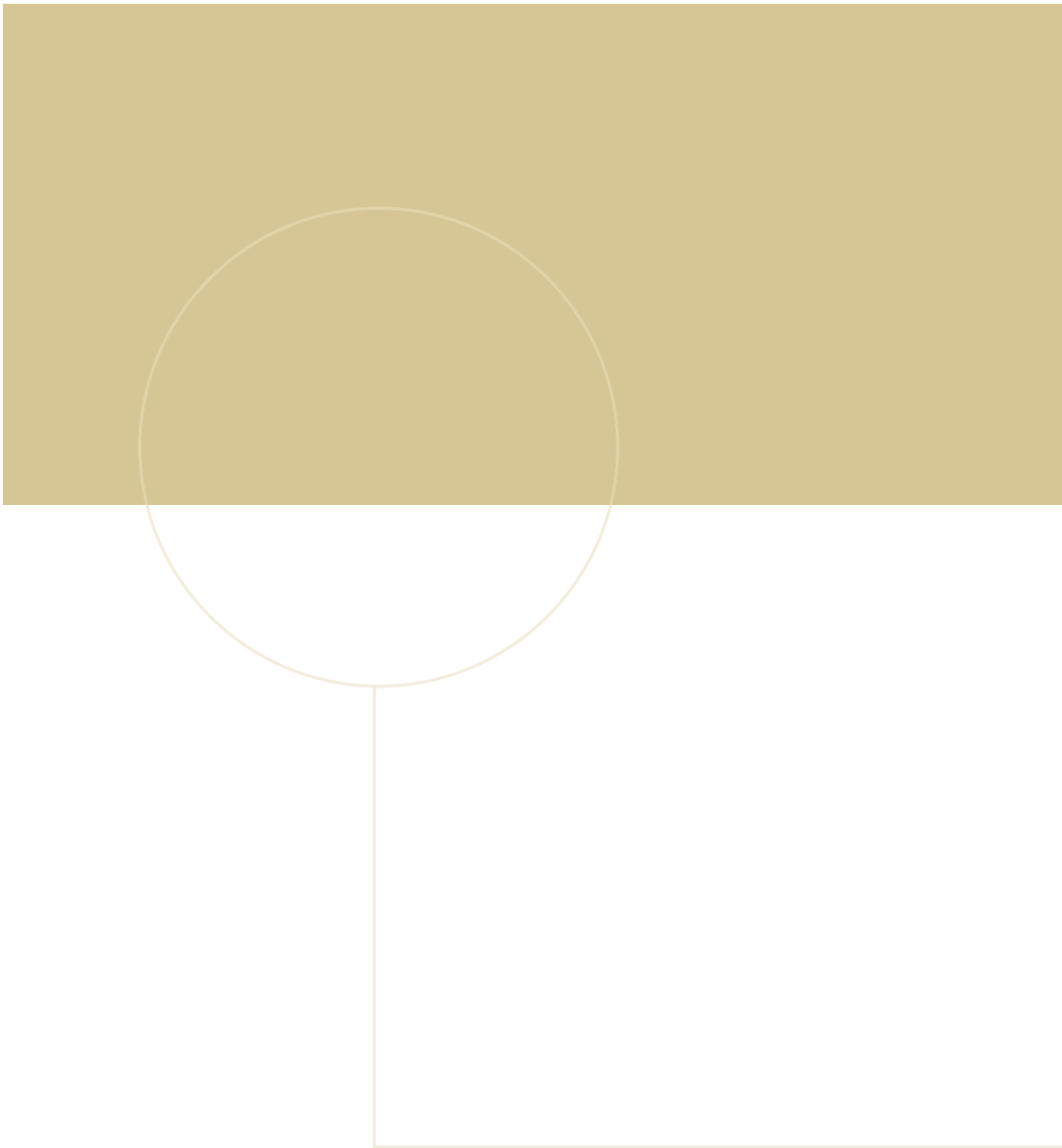
```
                    jp = Mx*j+pm-1
                    kp = Mx*k+pm-1
                end if
                do j_s=0,Mx-1
                    do i_s=0,Mx-1
                        V_i = reshape([ Vs(jp-pm+j_s,ip-pm+i_s,zl-Mx)
                                        Vs(jp-pm+j_s,ip-pm+i_s,zl)
                                        Vs(jp-pm+j_s,ip-pm+i_s,zl+Mx)],[3,1])
                        Vs(jp-pm+j_s,ip-pm+i_s,kp-pm:kp+pp-1) = reshape(
                                                                matmul(R_i,V_i),
                                                                [Mx])
                        U_i = reshape([ Us(jp-pm+j_s,ip-pm+i_s,zl-Mx)
                                        Us(jp-pm+j_s,ip-pm+i_s,zl)
                                        Us(jp-pm+j_s,ip-pm+i_s,zl+Mx)],[3,1])
                        Us(jp-pm+j_s,ip-pm+i_s,kp-pm:kp+pp-1) = reshape(
                                                                matmul(R_i,U_i),
                                                                [Mx])
                    end do
                end do
            end do
        end do
    end do
    !BC
    Vs(:,1-nh_d:0,:)          = Vs(:,NIs-2*nh_d+1:NIs-nh_d,:)
    Vs(:,NIs-nh_d+1:NIs,:)    = Vs(:,1:nh_d,:)
    Vs(1-nh_d:0,:,:)          = Vs(NJs-2*nh_d+1:NJs-nh_d,:,:)
    Vs(NJs-nh_d+1:NJs,:,:)    = Vs(1:nh_d,:,:)
    Vs(:,:,1-nh_d:0)          = Vs(:,:,NKs-2*nh_d+1:NKs-nh_d)
    Vs(:,:,NKs-nh_d+1:NKs)    = Vs(:,:,1:nh_d)
    !
    Us(:,1-nh_d:0,:)          = Us(:,NIs-2*nh_d+1:NIs-nh_d,:)
    Us(:,NIs-nh_d+1:NIs,:)    = Us(:,1:nh_d,:)
    Us(1-nh_d:0,:,:)          = Us(NJs-2*nh_d+1:NJs-nh_d,:,:)
    Us(NJs-nh_d+1:NJs,:,:)    = Us(1:nh_d,:,:)
    Us(:,:,1-nh_d:0)          = Us(:,:,NKs-2*nh_d+1:NKs-nh_d)
    Us(:,:,NKs-nh_d+1:NKs)    = Us(:,:,1:nh_d)

    !Step 6 - Intermediate velocities for W
    do j=1,NJ-1
        yl = j*Mx
        do k=1,NK-1
            do i=1,NI-1
                if(modulo(Mx,2)==0) then
                    pp = int(Mx/2)
                    pm = pp
                    ip = Mx*i+pp-2
                    kp = Mx*k+pp-2
                else
                    pp = int((Mx-1)/2)+1
```

```
                    pm = int((Mx-1)/2)
                    ip = Mx*i+pm-1
                    kp = Mx*k+pm-1
                end if
                do p_i=0,Mx-2
                    if(p_i==0) then
                        Wt_j(Mx-2,i) = w(j,i,k) - (Us(yl,ip-pp,kp+pm)
                                                 - Us(yl,ip+pm,kp+pm))*dZs/dX
                    else
                        Wt_j(Mx-2-p_i,i) = Wt_j(Mx-1-p_i,i)
                                         - (Us(yl,ip-pp,kp+pm-p_i)
                                         - Us(yl,ip+pm,kp+pm-p_i))*dZs/dX
                    end if
                end do
            end do
            Wt_j(:,0) = Wt_j(:,size(Wt_j(0,:))-2)
            Wt_j(:,size(Wt_j(0,:))-1) = Wt_j(:,1)
            !
            do i=1,NI-1
                if(modulo(Mx,2)==0) then
                    pp = int(Mx/2)+1
                    pm = int(Mx/2)-1
                    ip = Mx*i+pm-1
                    kp = Mx*k+pm-1
                else
                    pp = int((Mx-1)/2)+1
                    pm = int((Mx-1)/2)
                    ip = Mx*i+pm-1
                    kp = Mx*k+pm-1
                end if
                do p_i=0,Mx-2
                    Ws(yl,ip-pm:ip+pp-1,kp-pm+p_i) = matmul(R_i,Wt_j(p_i,i-1:i+1))
                end do
            end do
        end do
        Ws(yl,1-nh_d:0,:)          = Ws(yl,NIs-2*nh_d+1:NIs-nh_d,:)
        Ws(yl,NIs-nh_d+1:NIs,:)    = Ws(yl,1:nh_d,:)
        Ws(yl,:,1-nh_d:0)          = Ws(yl,:,NKs-2*nh_d+1:NKs-nh_d)
        Ws(yl,:,NKs-nh_d+1:NKs)    = Ws(yl,:,1:nh_d)
    end do
    Ws(1-nh_d:0,:,:)           = Ws(NJs-2*nh_d+1:NJs-nh_d,:,:)
    Ws(NJs-nh_d+1:NJs,:,:)     = Ws(1:nh_d,:,:)

    !Step 7 - Remaining sub-cells for W
    do k=1,NK-1
        do j=1,NJ-1
            yl = j*Mx
            do i=1,NI-1
                if(modulo(Mx,2)==0) then
```

```
341                     pp = int(Mx/2)
342                     pm = int(Mx/2)
343                     ip = Mx*i+pm-1
344                     jp = Mx*j+pm-1
345                     kp = Mx*k+pm-1
346                 else
347                     pp = int((Mx-1)/2)+1
348                     pm = int((Mx-1)/2)
349                     ip = Mx*i+pm-1
350                     jp = Mx*j+pm-1
351                     kp = Mx*k+pm-1
352                 end if
353                 do i_s=0,Mx-1
354                     do k_s=0,Mx-1
355                         W_i = reshape([ Ws(yl-Mx,ip-pm+i_s,kp-pm+k_s),
356                                         Ws(yl,   ip-pm+i_s,kp-pm+k_s),
357                                         Ws(yl+Mx,ip-pm+i_s,kp-pm+k_s)],[3,1])
358                         Ws(jp-pm:jp+pp-1,ip-pm+i_s,kp-pm+k_s) = reshape(
359                                                                 matmul(R_i,W_i),
360                                                                 [Mx])
361                     end do
362                 end do
363             end do
364         end do
365     end do
366     Ws(:,1-nh_d:0,:)          = Ws(:,NIs-2*nh_d+1:NIs-nh_d,:)
367     Ws(:,NIs-nh_d+1:NIs,:)    = Ws(:,1:nh_d,:)
368     Ws(1-nh_d:0,:,:)          = Ws(NJs-2*nh_d+1:NJs-nh_d,:,:)
369     Ws(NJs-nh_d+1:NJs,:,:)    = Ws(1:nh_d,:,:)
370     Ws(:,:,1-nh_d:0)          = Ws(:,:,NKs-2*nh_d+1:NKs-nh_d)
371     Ws(:,:,NKs-nh_d+1:NKs)    = Ws(:,:,1:nh_d)
372     !
373     return
374   end subroutine interpolation
```



NTNU

Kunnskap for en bedre verden