ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Seminar for Applied Mathematics SAM

Efficient Computation of Large-Scale Statistical Solutions to Incompressible Fluid Flows

T. Rohner and S. Mishra

Research Report No. 2024-04

January 2024
Latest revision: April 2024

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

Funding ERC: 770880 COMANFLO

Efficient Computation of Large-Scale Statistical Solutions to Incompressible Fluid Flows

Tobias Rohner
ETH Zürich
Zürich, Switzerland
tobias.rohner@sam.math.ethz.ch

Siddhartha Mishra
ETH Zürich
Zürich, Switzerland
ETH AI Center
Zürich, Switzerland
siddhartha.mishra@sam.math.ethz.ch

ABSTRACT

This work presents the development, performance analysis and subsequent optimization of a GPU-based spectral hyperviscosity solver for turbulent flows described by the three dimensional incompressible Navier-Stokes equations. The method solves for the fluid velocity fields directly in Fourier space, eliminating the need to solve a large-scale linear system of equations in order to find the pressure field. Special focus is put on the communication intensive transpose operation required by the fast Fourier transform when using distributed memory parallelism. After multiple iterations of benchmarking and improving the code, the simulation achieves close to optimal performance on the Piz Daint supercomputer cluster, even outperforming the Cray MPI implementation on Piz Daint in its communication routines. This optimal performance enables the computation of large-scale statistical solutions of incompressible fluid flows in three space dimensions.

CCS CONCEPTS

• **Applied computing** → **Engineering**; • **Computing methodologies** → *Parallel computing methodologies*; **Simulation evaluation**.

KEYWORDS

Computational Fluid Dynamics, Direct Numerical Simulation, GPU accelerated simulation

ACM Reference Format:
Tobias Rohner and Siddhartha Mishra. 2023. Efficient Computation of Large-Scale Statistical Solutions to Incompressible Fluid Flows. In *Proceedings of The Platform for Advanced Scientific Computing (PASC) Conference (PASC'24).* ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

PASC'24, June 3-5 2024, Zürich, Switzerland

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The flow of an incompressible fluid is described by the Navier-Stokes equations,

$$\begin{aligned} \mathbf{u}_t + (\mathbf{u}\cdot\nabla)\,\mathbf{u} + \nabla p = \nu\Delta\mathbf{u}, \quad (x,t) \in D\times[0,T], \\ \operatorname{div}\mathbf{u} = 0, \\ \mathbf{u}(x,0) = \bar{\mathbf{u}}(x), \quad x\in D. \end{aligned} \tag{1.1}$$

Here, $\mathbf{u}(x,t) \in \mathbb{R}^d$ is the velocity of the fluid, measured at the spatial location $x \in D \subset \mathbb{R}^d$ and time $t \in [0,T]$ and $p \in \mathbb{R}_+$ denotes the fluid pressure. The kinematic viscosity is denoted by ν and it scales inversely vis-à-vis the *Reynolds number Re* i.e., $\nu \sim \frac{1}{Re}$. It is well-known that for most fluids of interest [13], particularly in the atmosphere and the ocean as well as in flows of practical engineering interest, the Reynolds number can be very high. Hence, one is interested in the regime $\nu \to 0$, which also corresponds to the zero-viscosity limit of the Navier-Stokes equations.

Formally, when we assume periodic boundary conditions by setting $D = \mathbb{T}^d$ to be the d-dimensional periodic Torus, letting $\nu \to 0$ in (1.1), one obtains the well-known Euler equations for an ideal, incompressible fluid,

$$\begin{aligned} \mathbf{u}_t + (\mathbf{u}\cdot\nabla)\,\mathbf{u} + \nabla p = 0, \quad (x,t) \in D\times[0,T], \\ \operatorname{div}\mathbf{u} = 0, \\ \mathbf{u}(x,0) = \bar{\mathbf{u}}(x), \quad x\in D. \end{aligned} \tag{1.2}$$

1.1 The Role of Turbulence

Turbulence [13] is loosely defined as the presence of energetic eddies that span a very large range of spatial and temporal scales, even if the initial data is only varying at a single scale. This spontaneous appearance of multiple scales is a manifestation of the nonlinearities in the momentum equation in (1.1) and implies a cascade of input energy into smaller and smaller scales.

Turbulence is the principal obstacle in the availability of global well-posedness results for the Euler and Navier-Stokes equations. It is also responsible for the possible lack of convergence of numerical methods as a large number of scales, corresponding to wave numbers of $\approx Re^{\frac{3}{4}}$ need to be resolved, making the computational cost of a direct numerical simulation (DNS) of the Navier-Stokes equations prohibitive. In particular, these facts automatically imply that one can only expect convergence with respect to grid size for the Navier-Stokes equations when grid sizes smaller than $\ll \nu^{\frac{3}{4}}$ are considered. For the Euler equations, as $\nu = 0$, this also implies that classical numerical methods will not converge to a weak solution. This fact is already computationally verified in two spatial

dimensions with rough initial data [15, 17]. Thus, it is unclear what exactly a numerical method actually computes when approximating the Euler and Navier-Stokes equations (for high Reynolds numbers) at a given resolution.

1.2 Statistical Solutions

The above discussion clearly brings out the fact that the current solution concepts of weak and strong solutions for the Navier-Stokes (and Euler) equations are inadequate. They may not be globally well-posed and numerical methods may not converge to them on mesh refinement. This also provides the rationale for the search of alternative, more suitable, solution paradigms for incompressible flow.

One such alternative is the concept of measure-valued solutions, first introduced in [4]. Herein, the solutions are sought for as *Young measures* or space-time parametrized probability measures. One can think of them as assigning a probability distribution (pdf) at each point in space-time. Measure-valued solutions exist globally [4] and can be realized as limits of popular numerical methods such as the spectral viscosity method [15]. However, they are not unique. This is due to the fact that there is no information on correlations across different points in space.

Adding information about all possible multi-point correlations in an attempt to recover uniqueness leads to the paradigm of *Statistical Solutions* [11, 8] and references therein. Statistical solutions are time-parametrized probability measures on the underlying function space of square-integrable velocity fields. Their time evolution can be written as an infinite system of nonlinear differential equations, each evolving a particular moment in terms of other moments.

Statistical solutions provide a language to express turbulence mathematically in terms of solutions of Euler and Navier-Stokes equations [11]. Moreover, they also provide a natural framework for quantifying the uncertainties that are inherent to fluid flows [8]. Recently, the equivalence of two different definitions of statistical solutions was shown in [10]. Moreover, it was also shown that under the assumptions analogous to (weaker than) those used by Kolmogorov to derive his famous K41 phenomenological theory of turbulence, one can prove that statistical solutions of the Navier-Stokes equations (1.1) converge to statistical solutions of the Euler equations (1.2) as $\nu \to 0$.

2 SIMULATION

The simulation developed in the scope of this work is written in C++ using CUDA to realize optimized compute kernels to run on GPGPUs. For performance considerations, the computational domain is restricted to the d-dimensional torus $\mathbb{T}^d$ discretized by a uniform rectangular grid with mesh width $\Delta = \frac{1}{N}$. Large simulations are distributed over multiple compute nodes along a single dimension of the grid, while communication of data between nodes is implemented using MPI.

2.1 Computing Statistical Solutions

Given that statistical solutions are probability measures on the infinite dimensional space of square-integrable velocity fields, the challenge of computing statistical solutions is formidable. However, in recent papers [9, 18, 17], a Monte Carlo ensemble averaging algorithm to compute statistical solutions for both compressible as well as incompressible flows was proposed. Let the initial uncertainty in (1.2) (for definiteness) be modeled in terms of a probability measure $\bar{\mu} \in \text{Prob}((L^2(\mathbb{T}^d)^d))$. The aim is to find a suitable approximation to the statistical solution μ_t, whose correlation marginals satisfy the corresponding moment equations. To approximate μ_t, [9] and [17] proposed the following algorithm:

Algorithm 1:

1. Given initial measure $\bar{\mu}$, find M Monte Carlo samples $\bar{\mathbf{u}}_i(\omega)$ such that $\bar{\mu} \approx \frac{1}{M}\sum_{i=1}^{M} \delta_{\bar{\mathbf{u}}_i(\omega)}$.
2. $\forall \omega$, evolve $\bar{\mathbf{u}}_i(\omega)$ with suitable numerical method, at mesh resolution Δ, to obtain $\mathbf{u}_i^{\Delta}(t)$.
3. Define approximate statistical solution by the *empirical measure*: $\mu_t^{\Delta,M} = \frac{1}{M}\sum_{i=1}^{M} \delta_{\mathbf{u}_i^{\Delta}(t)}$.

Note the possibility of two independent parallelization strategies for accelerating the simulation. One approach computes multiple samples $\mathbf{u}_i^{\Delta}(t)$ in parallel, while the other approach parallelizes the evaluation of a single sample. Our simulation supports both strategies, as well as using them in combination. Due to the embarrassingly parallel nature of Monte Carlo sampling, we will only discuss parallelization of the evaluation of a single sample here.

2.2 Spectral Hyper-Viscosity Method

Note that in Algorithm 1 the number of Monte Carlo samples M should ideally scale with $M \propto N^2 = \frac{1}{\Delta^2}$. It is therefore of utmost importance that the computation of a single sample is implemented as efficiently as possible. Fourier spectral methods are an obvious choice of numerical method given our toroidal domain. In Fourier space, enforcing the divergence-free constraint of (1.2) reduces to a projection of each Fourier mode onto divergence-free vector fields. This circumvents the expensive computation of the pressure field reducing the computational complexity of the solver from $O(N^{2d+1})$ to $O(N^{d+1}\log N)$ where d is the spatial dimension.

We consider the following spatial discretization of the incompressible Navier-Stokes equations [17]

$$\begin{cases} \partial_t \mathbf{u}^{\Delta} + \mathcal{P}_N(\mathbf{u}^{\Delta}\cdot\nabla\mathbf{u}^{\Delta}) + \nabla p^{\Delta} & = \varepsilon_N |\nabla|^{2s}(Q_N * \mathbf{u}^{\Delta}) \\ \nabla\cdot\mathbf{u}^{\Delta} & = 0 \\ \mathbf{u}^{\Delta}|_{t=0} & = \mathcal{P}_N\mathbf{u}_0 \end{cases} \tag{2.1}$$

where $\mathcal{P}_N$ is the spatial Fourier projection operator mapping a function $f(x,t)$ to its first N Fourier modes: $\mathcal{P}_N = \sum_{|k|_\infty \le N} \hat{f}_k(t) e^{ik\cdot x}$. We additionally introduce the hyperviscosity parameter $s \ge 1$. When simulating the vanishing viscosity limit of the Navier-Stokes equations, this parameter can be used to fine-tune the dampening of the Fourier modes allowing for a large part of the spectrum to be free of dissipation. The viscosity term we use for the stabilization of the solver consists of a possibly resolution dependent viscosity ε_N and a Fourier multiplier Q_N controlling the strength at which different Fourier modes are dampened. This allows us to remove the dampening for the low frequencies, while applying some diffusion to the problematic higher ones. The Fourier multiplier Q_N is of the

form

$$Q_N(\mathbf{x}) = \sum_{\mathbf{k}\in\mathbb{Z}^d, |\mathbf{k}|\leq N} \hat{Q}_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{x}}. \tag{2.2}$$

In order to have convergence, the Fourier coefficients of Q_N need to fulfill [24, 23, 17]

$$\hat{Q}_{\mathbf{k}} = 0 \text{ for } |\mathbf{k}| \leq m_N, 1 - \left(\frac{m_N}{|\mathbf{k}|}\right)^{\frac{2s-1}{\theta}} \leq \hat{Q}_{\mathbf{k}} \leq 1 \tag{2.3}$$

where we have introduced an additional parameter $\theta > 0$. The quantities m_N and ε_N are required to scale as

$$m_N \sim N^{\theta}, \varepsilon_N \sim \frac{1}{N^{2s-1}}, 0 < \theta < \frac{2s-1}{2s}. \tag{2.4}$$

The authors of [16] show convergence of the above numerical method for a large class of initial conditions in the case of the two-dimensional incompressible Euler equations. For three dimensions no such result is available, but experimental evidence performed by the authors suggest that the results persist for three dimensions as well.

2.3 Implementation

Applying the Fourier transform to (2.1) and using the divergence-free constraint to replace the pressure by its exact solution yields

$$\partial_t \hat{\mathbf{u}}^{\Delta}_{\mathbf{k}} = \left(1 - \frac{\mathbf{k}\mathbf{k}^T}{|\mathbf{k}|^2}\right) \cdot \hat{\mathbf{b}}_{\mathbf{k}} \tag{2.5}$$

where $\hat{\mathbf{b}}_{\mathbf{k}} = -i\mathbf{k}^T \cdot \mathcal{F}\left[\mathbf{u}^{\Delta} \otimes \mathbf{u}^{\Delta}\right]_{\mathbf{k}}$. Note that the equation enforces the divergence-free constraint by orthogonally projecting the $\hat{\mathbf{b}}_{\mathbf{k}}$ onto vector fields fulfilling $i\mathbf{k}^T \cdot \hat{\mathbf{b}}_{\mathbf{k}} = 0$. This represents a pointwise operation in Fourier space resulting in the superior computational complexity of this spectral method over other classical methods.

The nonlinear term $\mathcal{F}\left[\mathbf{u}^{\Delta} \otimes \mathbf{u}^{\Delta}\right]$ is computed in physical space and uses the well known 2/3-dealiasing rule [21] to ensure stability of the solution. The algorithm to compute the time derivative $\partial_t \hat{\mathbf{u}}_{\mathbf{k}}$ can thus be described by the following steps, each represented by a compute kernel in the code itself:

Algorithm 2:

1. Pad $\hat{\mathbf{u}}^{\Delta}$ with zeroes to size $\frac{3}{2}N$
2. Compute $\mathbf{u}^{\Delta} = \mathcal{F}^{-1}\left[\hat{\mathbf{u}}^{\Delta}\right]$
3. Compute $\mathbf{B} = \mathbf{u}^{\Delta} \otimes \mathbf{u}^{\Delta}$
4. Compute $\hat{\mathbf{B}} = \mathcal{F}\left[\mathbf{B}\right]$
5. Dealias by removing the upper third of frequencies in $\hat{\mathbf{B}}$
6. Use $\hat{\mathbf{B}}$ to compute $\partial_t \hat{\mathbf{u}}^{\Delta}_{\mathbf{k}}$

Note that all kernels in Algorithm 2 have very low and (almost) constant operational intensity in the mesh resolution N. This suggests that reducing the number of memory accesses is crucial for obtaining a highly performant code. We do this with a combination of algorithmic improvements, kernel fusion, and blocking as detailed later in this paper.

Given the $O(N^d)$ memory requirements to capture the current state of the simulated flow field $\mathbf{u}^{\Delta}$, the need to distribute computations across multiple compute nodes arises quite early when striving for higher resolution simulations. (A single GPU on Piz Daint is able to contain a simulation of size 256^3). Note that the only nonlocal kernels present in Algorithm 2 are the forward and backward Fourier transforms, implying that the memory layout should be optimized towards their most efficient computation. Computing FFTs over distributed data, if desired to be efficient, is a nontrivial task [6]. We desire to minimize the amount of inter-node communication required by the algorithm, as we expect all the compute kernels to be memory bound. Hence, the optimal split of the data over compute nodes is given by a *slab decomposition* requiring only two transpose operations per FFT. Given all kernels, with minimal modifications to the code, can work with transposed data, the algorithm only needs a single transpose operation per FFT.

The typical MPI setup of a simulation looks as follows: The Monte Carlo samples are distributed over M independent subcommunicators, each computing a single sample at a time parallelized over N compute nodes. This strategy enables optimal utilization of compute resources. By taking N to be the minimal amount of compute nodes needed to fit a simulation of a single sample we maximize the degree of parallelization over Monte Carlo samples. As each sample can be computed independently, this provides us with a near optimal scaling of our simulation on massively parallel hardware.

Although our simulator supports arbitrary time-stepping schemes, we generally use the third-order strong stability preserving Runge-Kutta described by

$$\begin{aligned} u^{(1)} &= u(t) + \Delta t \partial_t u(t) \\ u^{(2)} &= \frac{3}{4}u(t) + \frac{1}{4}u^{(1)} + \frac{1}{4}\Delta t \partial_t u^{(1)} \\ u(t+\Delta t) &= \frac{1}{3}u(t) + \frac{2}{3}u^{(2)} + \frac{2}{3}\Delta t \partial_t u^{(2)}. \end{aligned} \tag{2.6}$$

Contrary to other SSP Runge-Kutta schemes, the third-order one is energy-diminishing in the context of linear hyperbolic systems[25]. This means that as long as the CFL condition is respected, the simulation is much more likely to stay stable even with nonlinear PDEs such as the incompressible Euler and Navier-Stokes equations.

2.4 In-Situ Processing

Our simulator's flexible IO system allows us to perform most post-processing of the data in-situ saving a massive amount of IO bandwidth and disk space. Flexibility is achieved by abstracting away the process of storing data to disk by introducing the concept of a `Writer`. Each writer provides the simulation with a list of time points at which it processes the current simulation state and stores the result to disk. Furthermore, a simulation can contain an arbitrary number of different writers each one storing data at different time intervals. Currently supported operations by writers include:

- Writing the whole flow field to disk
- Computing the energy spectrum
- Computing the enstrophy spectrum
- Computing various structure functions
- Perform visualization using ParaView and Catalyst [1]

All of these writers are parallelized over the MPI ranks of the simulation and require minimal additional memory.

3 OPTIMIZATION STRATEGIES

The third-order Runge Kutta scheme employed for time-stepping requires three evaluations of the time derivative $\partial_t \mathbf{u}^\Delta$. The focus of this section will therefore be on the optimization of the kernels used in Algorithm 2. Furthermore, all the benchmarking will be performed on the GPU partition of the Piz Daint compute cluster [3]. Each node contains an Intel Xeon E5-2690 v3 CPU paired with a single NVIDIA Tesla P100 GPU. Device memory is limited to 16GB, while the host provides 64GB of memory.

3.1 Fourier Transform

As the Fourier transforms in Algorithm 2 dominate its computational complexity, they can be expected to contribute to a significant portion of the total computational time. It is therefore crucial to optimize them meticulously. To this end, we choose to use the highly optimized and readily available FFT implementations FFTW [12] on the host and cuFFT [20] on the device. Both implementations excel when the size of the Fourier transform only contains small prime factors. For general sizes, slower algorithms must be used. This enables for some quite strong optimization by noticing that the 2/3-law for dealiasing can be relaxed to allow larger padding sizes. As the FFT is only applied to padded data, the size of this data can be chosen arbitrarily as long as it is larger than $\frac{3}{2}N$. By default, we round the padded data's size up to the next value of the form $2^a 3^b 5^c 7^d$ enabling both FFTW and cuFFT to make use of their optimized algorithms. Nonetheless, due to hardware and implementation details, increasing the padding size even more might still result in an overall speedup of the FFT computation. On any new system, we therefore once run a benchmark of only the Fourier transform storing its results and extracting the optimal padding size from that generated data. If no suitable size is found in the benchmark results, we choose the next larger N with prime factors at most 7 instead. This strategy guarantees the optimality of the FFT on each system the simulation is deployed on.

Having an optimal FFT implementation on a single node is not yet sufficient for problems where the domain is distributed over multiple compute nodes. Although FFTW provides an MPI implementation for exactly this case, cuFFT can only distribute over multiple GPUs given they are managed by the same process. We are therefore forced to perform the Fourier transform in at least three separate steps. First, we can apply it batchwise along the two axis where each rank owns all the data. After, we transpose the data to align the formerly distributed axis with one of the rank-local ones. This then enables us to apply another batched Fourier transform along this remaining axis. As the node interconnect bandwidth can be assumed to be significantly lower than the GPU memory bandwidth, special care should be taken when optimizing the required transpose operation.

3.2 Padding

In a naive implementation, padding the data with zeros would require an expensive `MPI_Alltoallw` call to redistribute data between ranks. However, by merging the padding with the Fourier transform kernel, we can omit this call completely. Note that if the Fourier transform is applied along a certain axis, it is sufficient for the data to be only padded along the same axis. This way, all the communication can be condensed into the single `MPI_Alltoall` call required by the Fourier transform itself. An illustration of the resulting padded Fourier transform kernel is shown in Figure 1 where the algorithm is decomposed into the following five fundamental steps:

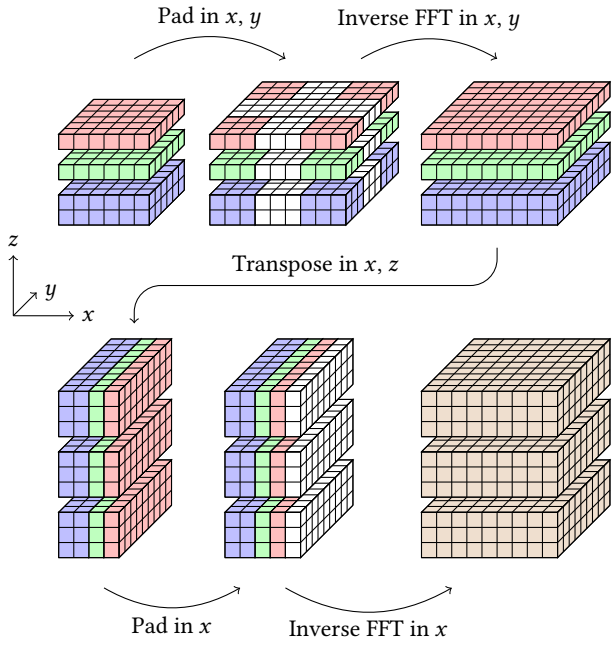


Figure 1: Illustration of the combined padding and inverse Fourier transform kernels for a domain size $N = 6$, padded size $N_{pad} = 9$, and 3 MPI ranks. Each cell of the domain is drawn as a small cube, while the subdomain currently located on a single MPI rank is illustrated as a block of these small cubes. The subdomains located on each rank are spaced apart from each other to reinforce the notion of them not residing in the same shared memory space. The initial data in Fourier space (top left) is complex valued and has shape $N \times N \times (\lfloor N/2 \rfloor + 1)$, while the output of the kernel (bottom right) is real valued and has shape $N_{pad} \times N_{pad} \times N_{pad}$.

Algorithm 3:

1. Pad the data with zeros in the rank-local x- and y-axes.
2. Apply a batched complex-to-complex inverse Fourier transform along the x- and y-axes.
3. Transpose by swapping the x- and z-axes
4. Pad with zeros in the new x-axis (formerly the z-axis)
5. Apply a batched complex-to-real inverse Fourier transform along the new x-axis.

Note that in this fused padding and Fourier transform kernel, the amount of data to be transposed is reduced by a factor of approximately 1.5, as it is only partially padded at that stage. Compared to the unfused kernels, this padded Fourier transform thus reduces the communicated data by almost a factor three. Furthermore, the forward Fourier transform needed in the simulation simultaneously

removes the padding from the data. This is achieved by executing Algorithm 3 in reverse.

3.3 Transpose

The transpose operation is the kernel with the most potential for optimization as it contains slow MPI communication between compute nodes and a strided memory access pattern resulting in cache unfriendly code and diminishing of the memory bandwidth. During the distributed transpose operation, the data is redistributed among the MPI ranks according to step 3 depicted in Figure 1. Each rank splits its data along the x-axis where the sizes of these blocks are given by their respective shapes in the transposed data. The slices are then transposed locally and sent to their corresponding ranks. This redistribution of the data requires copying the data from the GPU to the host, sending it to the other MPI ranks with an `MPI_Alltoall` call, and finally copying the received data back onto the GPU while at some point also performing the local transpose. The path a single packet of data takes in the implementation of the transpose is visualized in Figure 2. The local transpose operation is performed on the GPU benefiting from its higher bandwidth compared to the CPU. The data is then copied down into page locked memory enabling asynchronous transfers and doubling the bandwidth. From there it is sent to the receiver node on which it takes the same path in reverse. All of these operations are independent between the blocks sent to different nodes. It is therefore possible to completely parallelize them by overlapping most of the local transposes and moving data from and to the GPU with the slow inter node communication. However, note that communication can only start if the first block of memory was already transposed and copied to host memory. Hence, the first block is not able to be overlapped with the communication making it necessary to also optimize the locally performed operations. The local transpose is a prime candidate for optimization, as a naive implementation has very bad cache locality. This can be solved with the well known technique called blocking [5] where the transpose of an array is evaluated in small blocklets each fully fitting into cache. This way, no data that will later still be used is ever displaced from the cache allowing the algorithm to fully exploit the memory bandwidth.

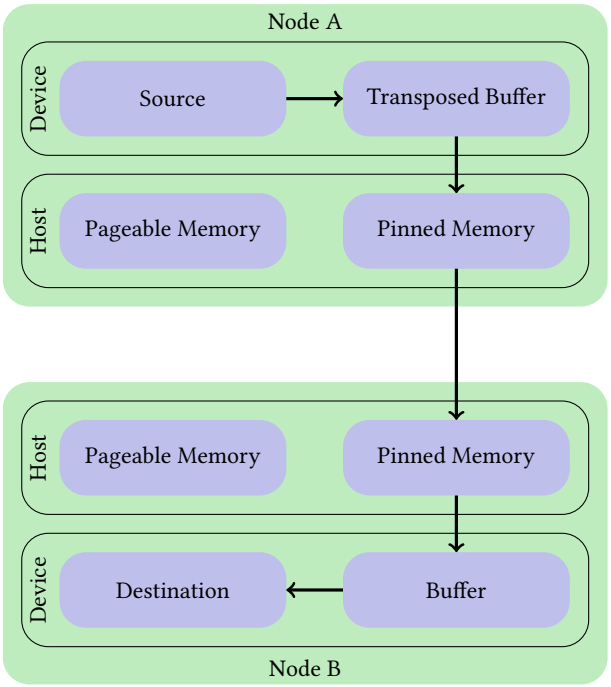


Figure 2: Unidirectional data path between two nodes for the distributed transpose operation on GPUs. We first preprocess the data by transposing it locally on the GPU. This data is then copied down into pinned memory on the host bypassing pageable memory doubling the bandwidth for device to host transfers. This is followed by sending the data asynchronously over MPI to the receiving node. There, the data is received directly into another pinned memory block and consequently copied onto the GPU. Finally, the data is then distributed into the destination array containing the transposed data.

3.4 Memory Optimizations

Computing the Fourier transform over data shared by multiple MPI ranks is a lot more expensive than computing a rank local Fourier transform due to the transpose operation required. We therefore want to fit large domains onto a single compute node. The main restriction for this is the amount of memory needed by the simulation. Reducing the memory requirements for a single kernel is difficult if not impossible. However, all the kernels are called sequentially and none of them require storing a significant amount of information across iterations of the simulation. Because most kernels are very cheap frequent allocation and deallocation of memory would result in a considerable hit on performance. We therefore implemented the reduction of memory requirements by introducing the concept of shared `Workspaces`. A workspace is nothing but a contiguous buffer of memory that is provided to a kernel as storage for in- or outputs or temporary results. Because these workspaces are managed by the simulation itself and not by each individual kernel, they can be shared across kernel boundaries enabling a significant portion of the buffers to overlap. This resulted in 12.5% memory savings for the single node simulation and in over 50% savings for the MPI implementation.

4 VALIDATION

For the validation of the code, over 150 tests have been implemented. First, the 2D solver is validated by simulating some analytically known solutions to the incompressible Euler equations and checking their convergence. Examples thereof are the 2D Taylor-Green Vortex [26], or the vortex patch [19]. The 3D solver is then checked by initializing the flow field with the previously checked 2D initial conditions while keeping the extra dimension constant. Both the 2D and the 3D versions are additionally verified visually by computing non-stationary solutions which were simulated/observed in previous work such as the discontinuous shear layer in [17], the double shear layer from [2], or the 3D Taylor-Green vortex [26]. This is especially important for the 3D incompressible Euler code, as there are no non-trivial stationary solutions in that case. Finally,

Table 1: Measured Bandwidths on the Piz Daint Cluster used to compute the lower bounds on the runtime of the compute kernels.

Type	Speed
β^H	21.24GB/s
β^D	488.64GB/s
β^{H-D}	19.24GB/s
β^{MPI}	9GB/s

we compute observables and the structure functions of some statistical solutions and compare them with structure functions from previous experiments done with different solvers as in [17, 22].

5 PERFORMANCE ASSESSMENT

Denote the host memory bandwidth by β^H, the device memory bandwidth by β^D, the bandwidth of the link between host and device by β^{D-H}, and the MPI communication bandwidth by β^{MPI}. Their values as measured on Piz Daint can be found in Table 1. Henceforth, we will assume that the FFTW and cuFFT implementations are optimal and not include them in the performance assessment of the simulation. The execution speed of all other kernels including the transpose operation will be compared against their theoretical optimum obtained by carefully analyzing their required data movements. We denote the resolution of the simulation domain by N, the padded size by N_{pad}, and the number of MPI processes by p. Additionally, s will denote the size of the underlying scalar datatype of our simulation in bytes. This will either be $s = 4$ for floats or $s = 8$ for doubles.

5.1 Padding and Unpadding

Padding is performed in two steps as depicted in Figure 1. As both of these padding operations only need to copy data to another array without performing any computations on them, the kernel is certainly memory bound. Hence, we only need to take the memory bandwidth of the system into account in order to obtain a lower bound for the runtime. Both steps must read each element of the input array exactly once and write to each element of the output exactly once as well. Counting the total number of (complex valued) elements in the arrays, we obtain $3N^2\frac{\lfloor N/2\rfloor+1}{p}$, $3N_{pad}^2\frac{\lfloor N/2\rfloor+1}{p}$, $3(\lfloor N/2\rfloor+1)N_{pad}\frac{N_{pad}}{p}$, and $3(\lfloor N_{pad}/2\rfloor+1)N_{pad}\frac{N_{pad}}{p}$. Collecting all the reads and writes on these elements and combining them with the memory bandwidth of our system, we obtain the following lower bound for the execution time of the padding kernel:

$$t^{pad}(N,p) \approx \frac{213}{16}\frac{N^3}{p}\frac{2s}{\beta} \tag{5.1}$$

where we have used that $\lfloor N/2\rfloor + 1 \approx N/2$ and $N_{pad} \approx \frac{3}{2}N$ for large N. β takes the value of either the host memory bandwidth β^H or the device memory bandwidth β^D depending on where the kernel is executed.

Although the unpadding operation is very similar, there are still a few key differences compared to the padding operation. Firstly, as unpadding is performed on the upper triangular matrix $B = u \otimes u$, we have 6 instead of 3 components to unpad. Secondly, the unpadding does not require reading the high frequency modes, as they are discarded anyway. Considering these changes, the lower bound for the execution time of the unpadding operation is given by

$$t^{unpad}(N,p) \approx \frac{78}{4}\frac{N^3}{p}\frac{2s}{\beta} \tag{5.2}$$

where we have used the same simplifications as above.

Table 2 shows the timings and achieved performances of our kernels. Note that the padding kernel achieves around 50% of the optimal performance we computed. This is most probably due to expensive index computations and the branching operation to decide whether to copy or write zero to the destination array. This hypothesis is also supported by the fact that the unpadding operation achieves almost 80% of the optimal performance, as it can reuse a single index computation for 6 instead of 3 components of the array. Additionally, it does not require branching leading to its improved performance results.

5.2 Transpose

In order to find the optimal execution speed to the transpose kernel, we need to know the critical path of the algorithm limiting the speed of the solver. As the pre-, and post-processing, and communication are independent, they can be overlapped in order to maximize parallelization. The algorithm must start by locally transposing a single block of data containing $3\frac{N_{pad}}{p}N_{pad}\frac{\lfloor N/2\rfloor+1}{p}$ complex valued elements on the GPU. This is followed by copying the block of data down into host memory, while simultaneously starting the local transpose of the second block. After the whole first block is located in host memory, the MPI communication can begin and the last block of data will be communicated after each MPI rank has sent its $3N_{pad}N_{pad}\frac{\lfloor N/2\rfloor+1}{p}$ local elements. The algorithm then finishes by copying this last block of data from the host onto the GPU and finally inserting that data in the destination buffer. Adding up all of these contributions, the lower bound for the execution time of the transpose operation of a single velocity component is given by

$$t^T(N,p) \approx \frac{9}{2}\frac{N^3}{p^2}\frac{2s}{\beta^D} + \frac{9}{4}\frac{N^3}{p^2}\frac{2s}{\beta^{H-D}} + \frac{9}{8}\frac{N^3}{p}\frac{2s}{\beta^{MPI}}. \tag{5.3}$$

To find the times for transposing u or B, this estimate can simply be multiplied by their respective number of components.

The benchmarking results in Table 2 show that the kernels achieve around 50% of the previously computed optimal performance estimate. This changes, however, if β^{MPI} is changed from the point-to-point message bandwidth to the bandwidth of an all-to-all communication reducing it to approximately $\beta^{MPI} = 4$GB/s. This has a large effect on the value of t^{opt} increasing it by almost a factor of two. Consequently also the efficiency is increased by the same amount. The resulting timings and efficiencies are listed in Table 2 in parentheses. Note that the transpose of B achieves over 100% efficiency. This can be contributed to the new upper bound of the MPI bandwidth obtained through a benchmark of `MPI_Alltoall`. Our custom implementation beats the native MPI on Piz Daint for messages larger than 1MB, achieving a transfer speed of approximately 5GB/s instead of the 4GB/s of the native MPI in the case covered by the benchmark problem considered.

Table 2: Runtimes of the compute kernels for a single precision floating point simulation of resolution $N = 512$ computed on the GPUs of 8 MPI ranks on the Piz Daint cluster. We measure the runtime t of a single call to the kernel, compare it to the lower bound t^{opt} previously computed, and also give the contribution to the total runtime of the simulation for each kernel. The execution time t, and the efficiency t^{opt}/t for the transpose is given twice. The value in parentheses is obtained by taking the MPI_Alltoall bandwidth instead of the point-to-point bandwidth on Piz Daint for the computation of the lower bound t^{opt}.

Kernel	t	t^{opt}	t^{opt}/t	Total Runtime %
Padding	6.982ms	3.657ms	52.4%	1.7%
Unpadding	6.783ms	5.356ms	79%	3%
Transpose u	120.765ms	56.681ms (119.596ms)	46.9% (99%)	29.8%
Transpose B	200.895ms	113.362ms (239.191ms)	56.4% (119.1%)	49.6%

5.3 Parallel Scaling

We measure strong and weak scaling of parallelization in both a single sample and over multiple Monte Carlo samples. The results can be found in Figures 3, and 4. Note that for statistical solutions, we are particularly interested in computing many Monte Carlo samples, even at the expense of some drop in spatial resolution. For a single sample, we therefore usually use close to the fewest number of ranks such that the sample still fits in memory and the main parallelization is done over Monte Carlo samples. This strategy is in agreement with the scaling of the solver, as the strong scaling seems to be slightly better in the number of samples per MPI rank than in the number of MPI ranks per sample. Also, the weak scaling efficiency of both the parallelization of a single sample and parallelization over Monte Carlo samples are almost optimal. The weak scaling efficiency over a single sample (Figure 3, right) even grows to be larger than 100% when using only slightly more ranks than necessary for the problem. Same can be observed for the weak scaling over Monte Carlo samples. After an early peak in efficiency, it levels off at approximately 97%. This small decrease in efficiency can most probably be attributed to more parallel accesses to the file system.

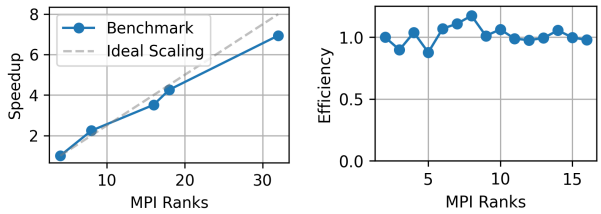


Figure 3: Strong (left) and weak (right) scaling of our simulation when parallelizing over a single sample. Strong scaling is computed at a domain size of $N = 512$ and $p = 4$ MPI ranks in the base case. Scaling can be observed to be almost optimal. To measure the weak scaling, we start with $N = 256$ parallelized over $p = 2$ MPI ranks. Then, to keep the amount of work per rank constant according to the computational complexity $O(N^3 \log_2(N))$ of the solver, we add more benchmarks for p up to 16 ranks while adjusting the domain size N accordingly. We observe the parallel efficiency to be optimal with some configurations even reaching efficiencies over 100%. This can be attributed to the more efficient overlap of computation with communication due to the smaller size and higher amount of data blocks to be communicated.

5.4 Baselines

In order to provide a benchmark demonstrating the speed of the whole parallelized FFT implementation, we present a comparison to the FFT used in the HACC (Hardware/Hybrid Accelerated Cosmology Code) introduced in [14]. The comparison of our implementation versus the SWFFT used in the HACC can be found in Table 3. The benchmark problems are chosen to be representative of the conditions encountered during typical use of our simulation. The domain sizes $N = 768^3$ and $N = 1536^3$ reflect the most common resolutions used in the context of statistical solutions computed with our solver and, due to padding according to the 2/3-dealiasing rule, correspond to an effective resolution of $N = 512^3$ and $N = 1024^3$. Simulations with resolutions lower than that fit onto a single GPU on Piz Daint and therefore use cuFFT directly, while resolutions larger than the $N = 1024^3$ are computationally too expensive to draw the necessary number of Monte Carlo samples from. As SWFFT is highly configurable, we run a grid search over all configurations selecting the fastest one for each individual problem. Nonetheless, our implementation consistently outperforms the SWFFT implementation by approximately 20%. This is mainly enabled by the fact that for statistical solutions we are able to trade

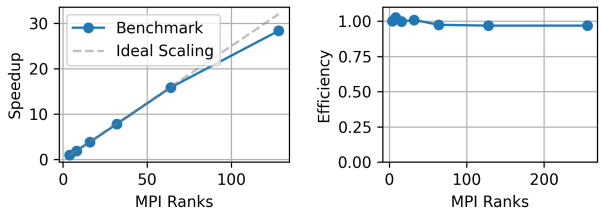


Figure 4: Strong (left) and weak (right) scaling of our simulation when parallelizing over multiple Monte Carlo samples at a fixed resolution $N = 512$ with 4 MPI ranks per sample. Strong scaling considers $M = 128$ samples, while for the weak scaling, we use $M = 128$ samples computed with $p = 256$ MPI ranks and scale down both M and p down accordingly to keep the work per rank constant. As expected from Monte Carlo sampling, both the strong and the weak scaling are close to optimal, with the only inefficiencies probably coming from parallel accesses to the file system.

Table 3: Runtimes of the GPU-based forward and backward Fourier transform performed with SWFFT and our implementation averaged over 8 runs. Domain sizes of $N = 768^3$ and $N = 1536^3$ were tested as these are the most common use cases in our simulation. Furthermore, each domain size was parallelized over $p = 8, 16, 32, 64$ MPI ranks. For SWFFT we ran a grid search to find the fastest configuration for each problem instance separately. Nonetheless, our implementation consistently outperformed SWFFT by around 20%.

N	p	SWFFT	Ours	Speedup
768	8	0.9754s	0.7945s	22.8%
768	16	0.4994s	0.4048s	23.4%
768	32	0.257s	0.2177s	18.1%
768	64	0.1382s	0.1301s	6.2%
1536	8	crashed	6.5391s	—
1536	16	4.057s	3.3237s	22.1%
1536	32	2.06s	1.7186s	19.9%
1536	64	1.0646s	0.87s	22.4%

resolution for the number of Monte Carlo samples drawn. The FFT used in our solver can exploit this by optimizing specifically for these (in comparison to HACC) low resolutions. In particular, we are able to use a slab instead of a pencil decomposition resulting in a significant reduction in communication overhead. Furthermore, smaller optimizations like overlapping communication with computation are able to be specifically tuned for the resolutions encountered when computing statistical solutions.

6 APPLICATION TO TURBULENT FLOWS

Recall that the concept of statistical solutions was originally introduced to restore convergence of the simulation results under mesh refinement. In this section we provide two examples to demonstrate this property. To this end, we use our solver to approximate statistical solutions to given initial conditions and demonstrate convergence of key statistical quantities such as the mean and the variance, as well as convergence in the Wasserstein distance of the empirical measures to a high-resolution reference solution.

6.1 Taylor-Green Vortex

The famous Taylor-Green Vortex in 3D [26] initializes the flow field with

$$\begin{aligned} u_0(x,y,z) &= A\cos(2\pi x)\sin(2\pi y)\sin(2\pi z) \\ v_0(x,y,z) &= B\sin(2\pi x)\cos(2\pi y)\sin(2\pi z) \\ w_0(x,y,z) &= C\sin(2\pi x)\sin(2\pi y)\cos(2\pi z). \end{aligned} \tag{6.1}$$

In order for the flow to be divergence free, we need to satisfy the constraint $A + B + C = 0$. To do so, we choose $A = 1$, $B = -1$, and $C = 0$. We add a small perturbation to the initial conditions which is given by the first-order harmonics with random amplitudes.We define $8d$ i.i.d uniformly distributed random variables $\delta_{d,i,j,k} \sim \mathcal{U}_{[-0.025,0.025]}$. We then define the perturbation $\varepsilon_d(x,y,z)$ on the

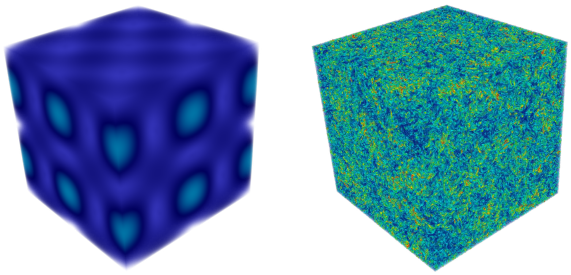


Figure 5: Vorticity magnitude of the perturbed Taylor-Green vortex at time $t = 0$ (left) and time $t = 5$ (right) at a resolution of $N = 512$.

d-th velocity component as

$$\begin{aligned} \varepsilon_d(x,y,z) &= \frac{1}{8}\sum_{(i,j,k)\in\{0,1\}^3} \delta_{d,i,j,k}\alpha_i(4\pi x)\alpha_j(4\pi y)\alpha_k(4\pi z), \\ \text{where } \alpha_i(x) &= \begin{cases} \sin(x) & \text{if } i = 0, \\ \cos(x) & \text{if } i = 1 \end{cases}. \end{aligned} \tag{6.2}$$

Finally, we arrive at the initial measure by perturbing the initial conditions u_0, v_0, and w_0 with ε_d

$$\begin{aligned} u_0(x,y,z) &= \cos(2\pi x)\sin(2\pi y)\sin(2\pi z) + \varepsilon_0(x,y,z) \\ v_0(x,y,z) &= -\sin(2\pi x)\cos(2\pi y)\sin(2\pi z) + \varepsilon_1(x,y,z) \\ w_0(x,y,z) &= \varepsilon_2(x,y,z). \end{aligned} \tag{6.3}$$

For the simulations we choose the hyperviscosity parameter $s = 1.5$ as this increases the range of the intermittent scales in the simulation. A realization of a simulation with the given random initial conditions can be found in Figure 5. Upon zooming in, an intricate web of vortex filaments can be seen at time $t = 5$.

According to the famous K41 theory [13] we expect to observe an anomalous energy dissipation even in the limit $\nu \to 0$. The flexible IO design of our simulator enables the periodic computation of the kinetic energy of the system at short time intervals. By consequently doing finite differences in time, we are able to compute the evolution of the energy dissipation rate of the system (Figure 6). The obtained rates agree well with results in common literature [7] providing additional verification of the solver.

To demonstrate the need for statistical instead of classical solutions, we plot the convergence under mesh refinement of a single sample, the mean, the variance, and the Wasserstein distance in Figure 7. As expected, we observe no sample-wise convergence. Contrary to this, the statistical quantities considered do converge at a reasonable rate. In particular does the convergence in Wasserstein distance of the one-point correlation marginals imply the convergence of all pointwise statistical moments.

6.2 Cylindrical Shear Flow

The Cylindrical Shear Flow is heavily inspired by the Flat Vortex Sheet experiment in [17] and is introduced as a 3D equivalent to

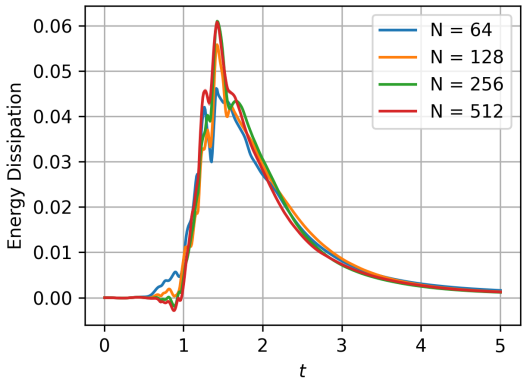


Figure 6: Evolution of energy dissipation $-\frac{\mathrm{d}E}{\mathrm{d}t}$ in the Taylor-Green vortex experiment. Visually, there seems to be a good agreement with energy dissipation rates commonly found in literature [7].

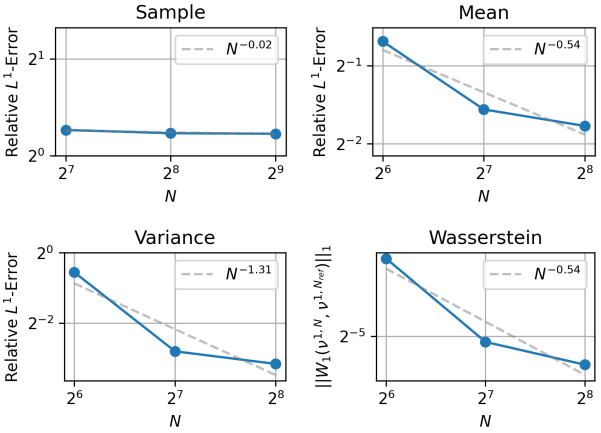


Figure 7: Convergence rates under mesh refinement of a single sample (top left), the mean (top right), the variance (bottom left), and the 1-point Wasserstein distance (bottom right) for the Taylor-Green Vortex. As expected, no sample wise convergence can be observed which can be explained by the ever smaller vortices being resolved and influencing the larger scales of the flow. Nevertheless, the pointwise mean and variance both seem to converge with a reasonable rate. This is confirmed by the convergence in the Wasserstein distance of the one-point correlation marginals of the statistical solution.

the latter. The initial conditions are given by

$$\begin{aligned} u_0(x,y,z) &= \tanh\left(2\pi\frac{r-0.25}{\rho}\right) \\ v_0(x,y,z) &= 0 \\ w_0(x,y,z) &= 0 \end{aligned} \tag{6.4}$$

where $r^2 = (y-0.5+\sigma_\delta^y(x))^2+(z-0.5+\sigma_\delta^z(x))^2$ and ρ is the smoothness parameter. We define the perturbations $\sigma_\delta^y(x)$ and $\sigma_\delta^z(x)$ in the following way: Let α_k^y and α_k^z be i.i.d uniformly distributed on $[0,1]$ and let β_k^y and β_k^z be i.i.d uniformly distributed on $[0,2\pi]$. Then $\sigma_\delta^y(x)$ and σ_δ^z are given by

$$\begin{aligned} \sigma_\delta^y(x) &= \delta\sum_{k=1}^{p}\alpha_k^y\sin(2\pi kx-\beta_k^y) \\ \sigma_\delta^z(x) &= \delta\sum_{k=1}^{p}\alpha_k^z\sin(2\pi kx-\beta_k^z) \end{aligned} \tag{6.5}$$

where we have chosen $p = 10$ and $\delta = 0.025$. These initial conditions are well defined in the limit $\rho \to 0$ where the interface between the flow directions becomes discontinuous and are then equal to

$$\begin{aligned} u_0(x,y,z) &= \begin{cases} -1 & \text{for } r \le 0.25 \\ 1 & \text{otherwise} \end{cases} \\ v_0(x,y,z) &= 0 \\ w_0(x,y,z) &= 0 \end{aligned} \tag{6.6}$$

where r is defined as above.

The solutions to this experiment contain a multitude of different flow regimes as the turbulence starts to develop at the interface $r = \sqrt{y^2+z^2} \approx 0.25$ and slowly propagates outward.

The simulations again use hyperviscosity parameter $s = 1.5$ in order to extend the intermittent range. A single realization of the initial and final conditions for $\rho = 0$ is shown in Figure 8 where we plot the vorticity magnitude of the flow field. As expected, turbulence spreads from the initial shear layer outward giving a beautiful view into its delicate structures. Likewise, we visualize the mean (Figure 9) and the variance (Figure 10) for different mesh resolutions. These low moments of the statistical solution contain only relatively low frequencies resulting in their smeared out appearance. This also suggests that in theory only few Fourier modes are needed to accurately approximate the mean and variance of statistical solutions and hints at the possibility of quite accurate simulations thereof with e.g. sophisticated turbulence modeling.

7 CONCLUSIONS

We have successfully implemented and optimized a spectral hyperviscosity solver for the incompressible Navier-Stokes equations and their vanishing viscosity limit. The solver's efficiency enables computation of both two- and three-dimensional large-scale statistical solutions to fluid flows which in turn can be used to deepen our knowledge about turbulence and serve as an invaluable source of training data for scientific Machine Learning models. Benchmarks of the code show close to optimal performance on the Piz Daint supercomputer, while also having excellent strong and weak scaling properties.

Optimality of the computational kernels was assessed by comparing their runtimes against their analytically computed lower bounds. These bounds assume zero cost for arithmetic operations and use memory bandwidths measured on the Piz Daint cluster. Furthermore, all additional simplifications were chosen such that they introduce a bias toward lower runtimes. Nonetheless, the MPI

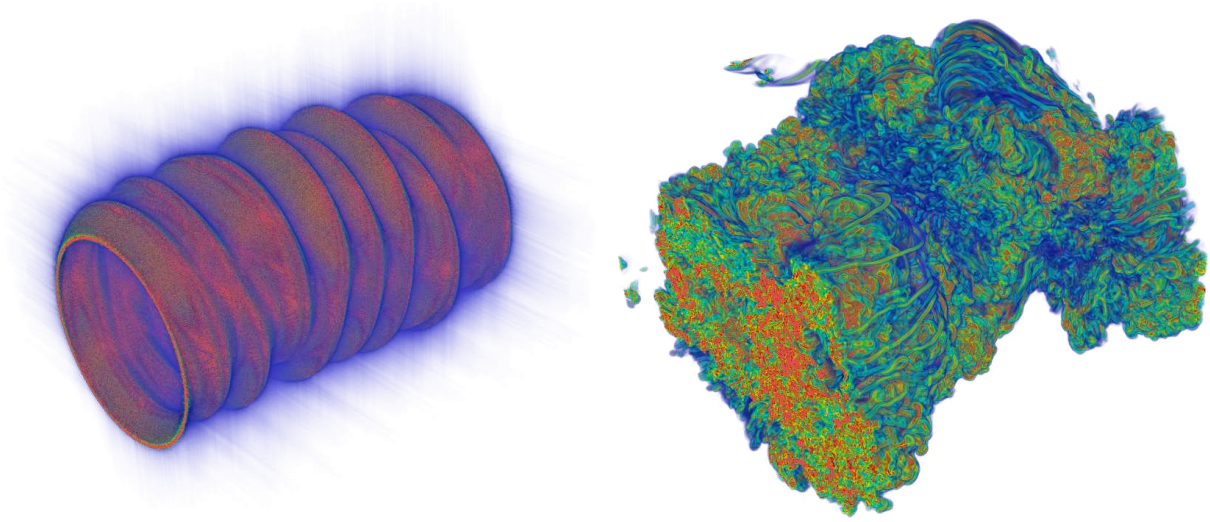
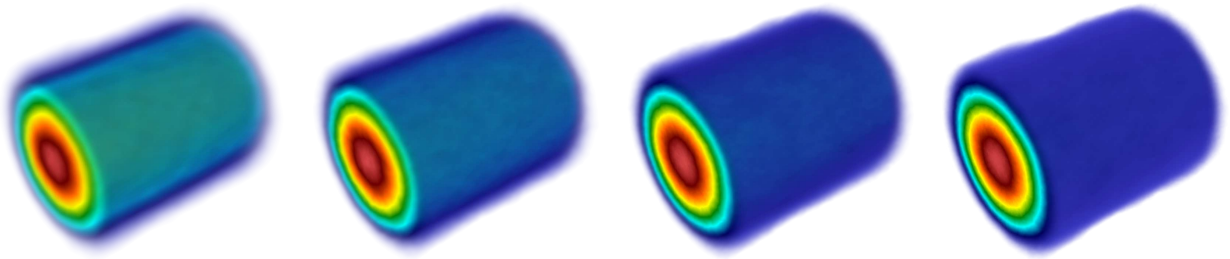
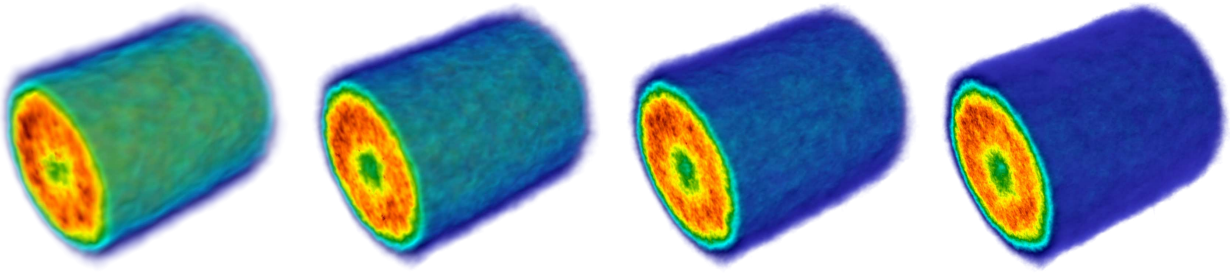


Figure 8: Vorticity magnitude of the Cylindrical Shear Flow at time $t = 0$ (left) and time $t = 1$ (right) at a resolution of $N = 512$. This experiment provides an especially good view of the vortex stretching in turbulent flows, as vortex tubes are clearly visible on the boundary between turbulent and laminar flow regimes.



(a) $N = 64$ **(b)** $N = 128$ **(c)** $N = 256$ **(d)** $N = 512$

Figure 9: Mean x-component of the flow field for the Cylindrical Shear Flow at time $t = 1$ at different resolutions.



(a) $N = 64$ **(b)** $N = 128$ **(c)** $N = 256$ **(d)** $N = 512$

Figure 10: Variance of x-component of the flow field for the Cylindrical Shear Flow at time $t = 1$ at different resolutions.

transpose responsible for half of the total runtime of the simulation still gets over 50% efficiency in these highly conservative estimates. Even more notably, it beats the speed of the native MPI implementation on Piz Daint by almost 20%.

We present the results of two simulations performed using our code. The well-known Taylor-Green vortex [26] demonstrates the need for statistical solutions excellently by developing turbulence throughout the domain early in the simulation and consequently not showing any sample wise convergence under mesh refinement. Statistical quantities on the other hand converge with a good rate making them useful in further research on turbulence. Additionally, the experiment clearly demonstrates Kolmogorov's hypothesized energy dissipation in the vanishing viscosity limit of the Navier-Stokes equations. As a second experiment, the Cylindrical Shear Flow demonstrates the statistical solutions ability to capture properties of flow fields with highly varying turbulence regimes accurately. Particular focus is put on the demonstration of the approximation capabilities on the pointwise mean and variance as they belong to the most important quantities used to conceptualize probability distributions.

The work introduced here paves the way for considerably more research into the behavior of turbulent fluid flows, especially their statistical properties. Combined with the convergence of statistical solutions under mesh refinement, this provides a promising path towards more accurate solutions to turbulent fluids than possible with current turbulence modeling strategies.

ACKNOWLEDGMENTS

This work was supported by a grant from the Swiss National Supercomputing Centre (CSCS) under project ID 1217.

REFERENCES

[1] Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. 2015. Paraview catalyst: enabling in situ data analysis and visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (ISAV2015). Association for Computing Machinery, Austin, TX, USA, 25–29. ISBN: 9781450340038. DOI: 10.1145/2828612.2828624.

[2] David L. Brown. 1995. Performance of under-resolved two-dimensional incompressible flow simulations. *Journal of Computational Physics*, 122, 1, 165–183. DOI: https://doi.org/10.1006/jcph.1995.1205.

[3] CSCS. 2023. Piz daint. Retrieved October 04, 2023 from https://www.cscs.ch/computers/piz-daint/.

[4] Ronald J. DiPerna and Andrew J. Majda. 1987. Oscillations and concentrations in weak solutions of the incompressible fluid equations. *Communications in Mathematical Physics*, 108, 4, 667–689. DOI: 10.1007/BF01214424.

[5] Ulrich Drepper. 2007. What every programmer should know about memory. Retrieved February 20, 2024 from https://www.akkadia.org/drepper/cpumemory.pdf.

[6] Truong Vinh Truong Duy and Taisuke Ozaki. 2014. A decomposition method with minimum communication amount for parallelization of multi-dimensional ffts. *Computer Physics Communications*, 185, 1, 153–164. DOI: https://doi.org/10.1016/j.cpc.2013.08.028.

[7] Niklas Fehn, Martin Kronbichler, Peter Munch, and Wolfgang A. Wall. 2022. Numerical evidence of anomalous energy dissipation in incompressible euler flows: towards grid-converged results for the inviscid taylor–green problem. *Journal of Fluid Mechanics*, 932, A40. DOI: 10.1017/jfm.2021.1003.

[8] Ulrik S. Fjordholm, Samuel Lanthaler, and Siddhartha Mishra. 2017. Statistical solutions of hyperbolic conservation laws: foundations. *Archive for Rational Mechanics and Analysis*, 226, 2, (Nov. 2017), 809–849. DOI: 10.1007/s00205-017-1145-9.

[9] Ulrik S. Fjordholm, Kjetil O. Lye, Siddhartha Mishra, and Franziska Weber. 2020. Statistical solutions of hyperbolic systems of conservation law: Numerical approximation. *Mathematical Models and Methods in Applied Sciences*, 30, 3, 539–609. DOI: 10.1142/S0218202520500141.

[10] Ulrik S. Fjordholm, Siddhartha Mishra, and Franziska Weber. 2022. On the vanishing viscosity limit of statistical solutions of the incompressible navier-stokes equations. arXiv: 2110.04674. DOI: 10.48550/arXiv.2110.04674.

[11] Ciprian Foias, Oscar Manley, Ricardo Rosa, and Roger Temam. 2001. *Navier-Stokes Equations and Turbulence. Encyclopedia of Mathematics and its Applications*. Cambridge University Press. DOI: 10.1017/CBO9780511546754.

[12] Matteo Frigo and Steven G. Johnson. 2005. The design and implementation of fftw3. *Proceedings of the IEEE*, 93, 2, 216–231. DOI: 10.1109/JPROC.2004.840301.

[13] Uriel Frisch. 1995. *Turbulence: The Legacy of A.N. Kolmogorov*. Cambridge University Press. ISBN: 9780521451031.

[14] Salman Habib et al. 2016. Hacc: simulating sky surveys on state-of-the-art supercomputing architectures. *New Astronomy*, 42, 49–65. DOI: https://doi.org/10.1016/j.newast.2015.06.003.

[15] Samuel Lanthaler and Siddhartha Mishra. 2015. Computation of measure-valued solutions for the incompressible Euler equations. *Mathematical Models and Methods in Applied Sciences*, 25, 11, 2043–2088. DOI: 10.1142/S0218202515500529.

[16] Samuel Lanthaler and Siddhartha Mishra. 2020. On the convergence of the spectral viscosity method for the two-dimensional incompressible euler equations with rough initial data. *Foundations of Computational Mathematics*, 20, 5, (Oct. 2020), 1309–1362. DOI: 10.1007/s10208-019-09440-0.

[17] Samuel Lanthaler, Siddhartha Mishra, and Carlos Parés-Pulido. 2021. Statistical solutions of the incompressible euler equations. *Mathematical Models and Methods in Applied Sciences*, 31, 02, (Feb. 2021), 223–292. DOI: 10.1142/s0218202521500068.

[18] Kjetil O. Lye. 2020. *Computation of statistical solutions of hyperbolic systems of conservation laws*. Ph.D. Dissertation.

[19] Siddhartha Mishra, Carlos Parés-Pulido, and Kyle G. Pressel. 2020. Arbitrarily high-order (weighted) essentially non-oscillatory finite difference schemes for anelastic flows on staggered meshes. (2020). arXiv: 1905.13665. DOI: 10.48550/arXiv.1905.13665.

[20] NVIDIA. 2023. Cufft library user's guide. NVIDIA. Retrieved October 04, 2023 from https://docs.nvidia.com/cuda/pdf/CUFFT_Library.pdf.

[21] Steven A. Orszag. 1971. On the elimination of aliasing in finite-difference schemes by filtering high-wavenumber components. *Journal of Atmospheric Sciences*, 28, 6, 1074–1074. DOI: 10.1175/1520-0469(1971)028<1074:OTEOAI>2.0.CO;2.

[22] Carlos Pares-Pulido. 2021. *Statistical solutions of the incompressible Euler equations with finite volume methods*. Ph.D. Dissertation.

[23] Eitan Tadmor. 2004. Burgers' Equation with Vanishing Hyper-Viscosity. *Communications in Mathematical Sciences*, 2, 2, 317–324. DOI: 10.4310/CMS.2004.v2.n2.a9.

[24] Eitan Tadmor. 1989. Convergence of spectral methods for nonlinear conservation laws. *SIAM Journal on Numerical Analysis*, 26, 1, 30–44. DOI: 10.1137/0726003.

[25] 2002. *From semidiscrete to fully discrete: stability of runge-kutta schemes by the energy method. ii*. D.J. Estep and S. Tavener. *Collected Lectures on the Preservation of Stability Under Discretization*. Chap. 3.

[26] Geoffrey I. Taylor and Albert E. Green. 1937. Mechanism of the production of small eddies from large ones. *Proceedings of the Royal Society of London. Series A - Mathematical and Physical Sciences*, 158, 895, 499–521. DOI: 10.1098/rspa.1937.0036.

Received XXX; revised XXX; accepted XXX