Model description paper

# BoundaryLayerDynamics.jl v1.0: a modern codebase for atmospheric boundary-layer simulations

**Manuel F. Schmid**[1,2], **Marco G. Giometto**[2], **Gregory A. Lawrence**[1], and **Marc B. Parlange**[3,4,5]

[1]Department of Civil Engineering, University of British Columbia, Vancouver, BC, Canada
[2]Department of Civil Engineering & Engineering Mechanics, Columbia University, New York, NY, USA
[3]Mechanical, Industrial, and Systems Engineering, University of Rhode Island, Kingston, RI, USA
[4]Ocean Engineering, University of Rhode Island, Kingston, RI, USA
[5]Civil and Environmental Engineering, University of Rhode Island, Kingston, RI, USA

**Correspondence:** Manuel F. Schmid (mfs2173@columbia.edu) and Marco G. Giometto (mg3929@columbia.edu)

**Abstract.** We present BoundaryLayerDynamics.jl, a new code for turbulence-resolving simulations of atmospheric boundary-layer flows as well as canonical turbulent flows in channel geometries. The code performs direct numerical simulation as well as large-eddy simulation using a hybrid (pseudo)spectral and finite-difference approach with explicit time advancement. Written in Julia, the code strives to be flexible and adaptable without sacrificing performance, and extensive automated tests aim to ensure that the implementation is and remains correct. We show that the simulation results are in agreement with published results and that the performance is on par with an existing Fortran implementation of the same methods.

## 1 Introduction

Since Deardorff's early studies (Deardorff, 1969, 1970a, b), numerical simulations of the three-dimensional, unsteady flow field have become an integral part of microscale atmospheric boundary-layer (ABL) research. Direct numerical simulation (DNS) provides an extremely accurate tool to study fundamental properties of turbulent flows and their scaling from low to moderate Reynolds numbers (Moin and Mahesh, 1998). Large-eddy simulation (LES) provides a numerical model for a wide range of ABL flow phenomena at realistic Reynolds numbers while relying on modest, well-supported modeling assumptions (Meneveau and Katz, 2000; Stoll et al., 2020). Together, DNS and LES constitute the

backbone of the computational study of turbulent flow dynamics and have contributed many insights to our current understanding.

Many different implementations of these methods are in use for ABL research and continue to be actively developed. Projects such as PALM (Maronga et al., 2020), OpenFOAM (Chen et al., 2014), and the Weather Research and Forecasting (WRF) model (Skamarock et al., 2021) are developed in a community effort as open-source codes with broad applicability. In addition, many research groups have their own codes that have been developed and extended over decades and are passed person to person. Some studies also rely on commercial software such as Ansys Fluent, although not having access to implementation details is problematic for scientific reproducibility.

Codes differ along many dimensions, perhaps most importantly in the physical models that are implemented. The numerical methods used to compute a solution can also lead to important differences, particularly for LES, where the smallest resolved scales of motion are an integral part of the turbulence dynamics (Kravchenko and Moin, 1997). The performance characteristics of a code are also central as most turbulence-resolving simulations continue to be limited by their computational cost. Furthermore, there can be important differences in the effort and model-specific experience required for setting up simulations and making changes and additions to the source code while ensuring the correctness of the results. When developing a new code or selecting an ex-

isting model for a simulation, these different qualities have to be weighed against each other and trade-offs are inevitable.

The advent of the Julia programming language (Bezanson et al., 2017) represents a shift in the landscape of possible trade-offs between conflicting goals. Publicly launched in 2012 and stabilized with version 1.0 in 2018, Julia promises to combine the performance of Fortran, C, and C++ with the convenience of Python, MATLAB, and R. Automatic memory management, dynamic typing with type inference, multiple dispatch, and a built-in modern package manager facilitate rapid development of clear, concise code that keeps orthogonal functionality separate. At the same time, Julia's "just-ahead-of-time" compilation model allows code to run with no or minimal computational overhead.

In this paper, we present and discuss BoundaryLayerDynamics.jl (Schmid et al., 2023b), a new code for turbulence-resolving flow simulation optimized for ABL research. The code has been written to provide core functionality for DNS and LES of channel-flow configurations with a focus on making it easy to use, adapt, and extend the code without jeopardizing the correctness of the results. To achieve better trade-offs along these dimensions, the implementation relies on the Julia programming language, on automated testing, and on a modular design.

The suitability of the new code is of course not limited to simulations of ABL turbulence. In fact, none of the current functionality is specific to ABL applications. However, the choice of physical models and numerical methods is guided by the needs of such applications and future developments will similarly prioritize those use cases.

The code solves the incompressible Navier–Stokes equations relying on (pseudo)spectral and finite difference methods for discretization in the horizontal and vertical directions respectively, making use of the Message Passing Interface (MPI) for parallelization in the vertical direction and performing fully explicit time integration. The details of the numerical methods are described in Sect. 2. The implementation is validated via a number of automated tests as described in Sect. 3, where we also present a validation against DNS and LES results computed with different codes. The performance analysis presented in Sect. 4 shows that the computational cost is comparable to a Fortran implementation of the same numerical approach and that parallel performance scales favorably up to the maximum supported number of parallel processes.

BoundaryLayerDynamics.jl is open-source software and available under the MIT License through the official Julia package repository and on GitHub (https://github.com/efpl-columbia/BoundaryLayerDynamics.jl, last access: 22 November 2023), where the public repository of the package is currently hosted. The version described in this article is archived on Zenodo (Schmid et al., 2023b).

## 2 Governing equations and numerical methods

The choice of governing equations and numerical methods is guided by the goal of studying the turbulent flow dynamics in the atmospheric boundary layer. Other turbulent flows in engineering and in the natural environment are also considered insofar as their requirements do not conflict with those of atmospheric boundary-layer flows.

It is well-established that the Navier–Stokes equations are an extremely accurate physical model for flows with a Knudsen number $Kn \ll 1$ and that compressibility effects are minimal for flows with a Mach number $Ma \lesssim 1/3$ (Panton, 2013). Since these conditions are met for atmospheric boundary-layer flows, the incompressible Navier–Stokes equations,

$$\frac{\partial u_i}{\partial t} + u_j \left( \frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i} \right) = \frac{1}{Re} \frac{\partial^2 u_i}{\partial x_j \partial x_j} - \frac{\partial p}{\partial x_i} + f_i \qquad (1)$$

and

$$\frac{\partial u_i}{\partial x_i} = 0 \qquad (2)$$

are used as the mathematical model for this work, given here with the rotational form of the advection term (Orszag, 1971a), for which the total kinematic pressure $p = p^{\text{gauge}}/\rho + \frac{1}{2} u_i u_i$ includes both the static and dynamic pressure. Quantities are non-dimensionalized with a length scale $\mathcal{L}$ and a velocity $\mathcal{U}$, producing the Reynolds number $Re = \mathcal{U} \mathcal{L}/\nu$. The Cartesian coordinates $x_i$ denote the primary horizontal ($i = 1$, usually streamwise), secondary horizontal ($i = 2$, usually cross-stream), and vertical ($i = 3$) directions, while the corresponding velocity components are given as $u_i$. The term $f_i$ denotes the components of a body force, usually gravity or a constant pressure gradient, and the kinematic viscosity $\nu$ and fluid density $\rho$ are assumed scalar constants.

While this formulation can serve as a reasonable representation of a neutrally stratified atmospheric boundary layer, there are many important ABL processes that are not included. Coriolis forces, temperature, and humidity in particular are of central importance for many applications. The current code is meant to provide the core functionality necessary for ABL flow simulations and serve as a foundation for a more comprehensive set of physical and numerical models that can be added over time.

For flows with a moderate to high Reynolds number, current computational capabilities generally do not permit resolving the full range of scales of motion. In this case, the filtered Navier–Stokes equations,

$$\frac{\partial \widetilde{u}_i}{\partial t} + \widetilde{u}_j \left( \frac{\partial \widetilde{u}_i}{\partial x_j} - \frac{\partial \widetilde{u}_j}{\partial x_i} \right) + \frac{\partial \tau_{ij}^{\text{sgs}}}{\partial x_j}$$

$$= \frac{1}{Re} \frac{\partial^2 \widetilde{u}_i}{\partial x_j \partial x_j} - \frac{\partial \widetilde{p}}{\partial x_i} + \widetilde{f}_i \qquad (3)$$

and

$$\frac{\partial \widetilde{u}_i}{\partial x_i} = 0, \qquad (4)$$

are used as the computational model, where $\widetilde{u}_i$ represents the spatially filtered velocity field, i.e., $\widetilde{u}_i = \int G(\boldsymbol{r}, \boldsymbol{x}) u_i(\boldsymbol{x} - \boldsymbol{r}) \mathrm{d}\boldsymbol{r}$ with $G$ defining the filtering operation. The subgrid-scale stress tensor $\tau_{ij}^{\mathrm{sgs}} = \tau_{ij}^{\mathrm{R}} - \frac{1}{3}\tau_{kk}^{\mathrm{R}}\delta_{ij}$ represents the anisotropic component of the residual stress tensor $\tau_{ij}^{\mathrm{R}} = \widetilde{u_i u_j} - \widetilde{u}_i \widetilde{u}_j$ and has to be modeled as a function of the resolved velocity $\widetilde{u}_i$. The modified pressure $\widetilde{p} = \widetilde{p}^{\mathrm{gauge}}/\rho + \frac{1}{2}\widetilde{u}_i\widetilde{u}_i + \frac{1}{3}\tau_{ii}^{\mathrm{R}}$ now includes contributions from the filtered gauge pressure $\widetilde{p}^{\mathrm{gauge}}$, the resolved kinetic energy, and the unresolved kinetic energy. The forcing term $\widetilde{f}_i$ is simply the spatially filtered $f_i$. In the following, the same notation is used to represent both the unfiltered (DNS) and filtered (LES) equations to simplify the notation.

The past decades have seen several efforts to develop a suitable model for $\tau_{ij}^{\mathrm{sgs}}$ (Smagorinsky, 1963; Schumann, 1975; Bardina et al., 1980; Germano et al., 1991; Meneveau et al., 1996; Porté-Agel et al., 2000; Bou-Zeid et al., 2005). The current implementation includes the static Smagorinsky (1963) subgrid-scale model:

$$\tau_{ij}^{\mathrm{sgs}} = -2l_{\mathrm{S}}^2 \mathcal{S} S_{ij}, \tag{5}$$

where $S_{ij} = 1/2\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$ is the resolved strain rate, $\mathcal{S} = \sqrt{2S_{ij}S_{ij}}$ is the characteristic or total strain rate, and $l_{\mathrm{S}}$ is the Smagorinsky length scale, taken to be the product of the filter width $\Delta$ and a constant Smagorinsky coefficient $C_{\mathrm{S}}$.

Specifying boundary conditions for turbulent flows remains a challenging research problem, since chaotic velocity fluctuations have to be prescribed in a physically accurate manner. To avoid these difficulties, turbulence-resolving simulations are often run with periodic boundary conditions. While this requires that the problem is formulated such that it can be approximated with a periodic flow field, unphysical border regions are avoided and accurate results can be obtained in the whole domain as long as the domain is large enough to accommodate all relevant scales of motion. Furthermore, the flow field can then be expressed in terms of periodic basis functions. For a horizontally periodic domain of size $L_1 \times L_2$, the velocity field can be written as

$$u_i(x_1, x_2, x_3) = \sum_{-\infty < \kappa_1 < \infty} \sum_{-\infty < \kappa_2 < \infty} \hat{u}_i^{\kappa_1 \kappa_2}(x_3)\, e^{i\kappa_1 2\pi x_1/L_1}\, e^{i\kappa_2 2\pi x_2/L_2}. \tag{6}$$

With similar expressions for the pressure $p$ and the forcing $f_i$, we can rewrite the governing equations for a single mode with wavenumber $\kappa_1$ in $x_1$ direction and $\kappa_2$ in $x_2$ direction,

$$\frac{\partial}{\partial t}\hat{u}_i^{\kappa_1\kappa_2}(x_3) + \sum_{-\infty < \kappa_1' < \infty} \sum_{-\infty < \kappa_2' < \infty} \hat{u}_j^{(\kappa_1-\kappa_1')(\kappa_2-\kappa_2')}(x_3)$$
$$\left(\hat{D}_j^{\kappa_1'\kappa_2'}\hat{u}_i^{\kappa_1'\kappa_2'}(x_3) - \hat{D}_i^{\kappa_1'\kappa_2'}\hat{u}_j^{\kappa_1'\kappa_2'}(x_3)\right)$$
$$+ \hat{D}_j^{\kappa_1\kappa_2}\hat{\tau}_{ij}^{\mathrm{sgs}\,\kappa_1\kappa_2}(x_3)$$
$$= \frac{1}{Re}\left(\hat{D}_j^{\kappa_1\kappa_2}\right)^2 \hat{u}_i^{\kappa_1\kappa_2}(x_3)$$
$$- \hat{D}_i^{\kappa_1\kappa_2}\hat{p}^{\kappa_1\kappa_2}(x_3) + \hat{f}_i^{\kappa_1\kappa_2}(x_3), \tag{7}$$

where $\hat{D}_1^{\kappa_1\kappa_2} = \frac{2\pi i \kappa_1}{L_1}$, $\hat{D}_2^{\kappa_1\kappa_2} = \frac{2\pi i \kappa_2}{L_2}$, and $\hat{D}_3^{\kappa_1\kappa_2} = \frac{\partial}{\partial x_3}$ are the differential operators, with $i$ denoting the imaginary unit. For direct numerical simulations the subgrid-scale term is omitted. The continuity equation becomes

$$\hat{D}_i^{\kappa_1\kappa_2}\hat{u}_i^{\kappa_1\kappa_2}(x_3) = 0. \tag{8}$$

In the vertical direction, turbulence is not homogeneous for ABL flows and periodic boundary conditions are not applicable. For a channel geometry, boundary conditions have to be specified for $u_i(x_3 = 0)$ and $u_i(x_3 = L_3)$ with the constraint that

$$\int_0^{L_2}\int_0^{L_1} u_3(x_3 = 0) - u_3(x_3 = L_3)\,\mathrm{d}x_1\mathrm{d}x_2 = 0, \tag{9}$$

which can be obtained from integrating the continuity equation over the whole domain. A number of engineering flows such as smooth-wall open- and closed-channel flows can be modeled with Dirichlet and Neumann boundary conditions. The complex boundaries of atmospheric flows can require significant modeling effort, and simulations generally have to partially resolve surfaces (e.g., immersed boundary method, terrain-following coordinates) or represent their effect with a wall model for $\tau_{i3}^{\mathrm{sgs}}$, usually formulated for the discretized equations (Piomelli and Balaras, 2002). The current implementation includes an algebraic equilibrium rough-wall model defined similar to Mason and Callen (1986) with

$$\tau_{i3}^{\mathrm{sgs}}(x_3 = 0) = \frac{-\kappa^2 \sqrt{u_1(x_3^{\mathrm{ref}})^2 + u_2(x_3^{\mathrm{ref}})^2}}{\log(x_3^{\mathrm{ref}}/z_0)^2} u_i(x_3^{\mathrm{ref}}) \tag{10}$$

for $i = 1, 2$ and $\tau_{33}^{\mathrm{sgs}}(x_3 = 0) = 0$, where $z_0$ is the roughness length, $\kappa \approx 0.4$ is the von Kármán constant, and $x_3^{\mathrm{ref}} > z_0$ is a reference height at which the (resolved) velocity is obtained, usually chosen as the first grid point. To improve the near-wall behavior of the subgrid-scale model, the Smagorinsky length scale is adjusted to $l_{\mathrm{S}}^{-n} = (C_{\mathrm{S}}\Delta)^{-n} + (\kappa x_3)^{-n}$ as proposed by Mason and Thomson (1992), with $n = 2$ as the default value.

For the numerical solution of Eqs. (7) and (8), we limit ourselves to $N_1 \times N_2$ wavenumbers at $N_3$ vertical grid points. The wavenumbers are selected symmetrically around $\kappa_i = 0$, i.e., $|\kappa_1| \leq (N_1 - 1)/2$ and $|\kappa_2| \leq (N_2 - 1)/2$. This results in an odd number of wavenumbers in each direction and avoids the need for a special treatment of Nyquist frequencies. Since $\hat{\phi}^{-\kappa_1-\kappa_2} = \hat{\phi}^{\kappa_1\kappa_2*}$ for any real-valued $\phi$, we only need to explicitly solve for half the modes and can obtain the others through complex conjugation. In vertical direction, equidistant grid points are selected from the interval $[0, 1]$, which is then mapped to the domain with a function $x_3 : [0, 1] \rightarrow [0, L_3]$, $\zeta \longmapsto x_3(\zeta)$. This function can be used for grid stretching in the vertical direction; the choice of $x_3 : \zeta \longmapsto L_3\zeta$ defines a uniform grid. A staggered arrangement of grid points with $\zeta_C$ at the center of the $N_3$ segments

and $\zeta_I$ at the $N_3 - 1$ interfaces between them, i.e.,

$$\zeta_C \in \left\{ \frac{1/2}{N_3}, \frac{3/2}{N_3}, \ldots, \frac{N_3 - 1/2}{N_3} \right\} \tag{11}$$

for $u_1, u_2, p, f_1, f_2, \tau_{ii}^{\mathrm{sgs}}, \tau_{12}^{\mathrm{sgs}}$, along with

$$\zeta_I \in \left\{ \frac{1}{N_3}, \frac{2}{N_3}, \ldots, \frac{N_3 - 1}{N_3} \right\} \tag{12}$$

for $u_3, f_3, \tau_{13}^{\mathrm{sgs}}, \tau_{23}^{\mathrm{sgs}}$, avoids the need to specify boundary conditions for the pressure field, prevents odd–even decoupling, and results in a smaller effective grid spacing (Ferziger et al., 2020). When running large-eddy simulations, the discretization implicitly defines the spatial filter $G$ and the filter width $\Delta$ is taken to be $\Delta = \sqrt[3]{\Delta_1 \Delta_2 \Delta_3}$ (Scotti et al., 1993) with $\Delta_1 = L_1/N_1$, $\Delta_2 = L_2/N_2$, and $\Delta_3 = 1/N_3 \mathrm{d}x_3/\mathrm{d}\zeta$.

The horizontal derivatives $\hat{D}_1^{\kappa_1 \kappa_2}$ and $\hat{D}_2^{\kappa_1 \kappa_2}$ can be computed exactly. For the vertical derivative $\hat{D}_3^{\kappa_1 \kappa_2}$, we use central second-order finite differences on the staggered $\zeta$ nodes (Moin and Verzicco, 2016) as well as the analytical derivative of $x_3(\zeta)$, i.e.,

$$\begin{aligned} \hat{D}_3^{\kappa_1 \kappa_2} \hat{\phi}^{\kappa_1 \kappa_2} \Big|_\zeta &= \frac{\mathrm{d}\zeta}{\mathrm{d}x_3} \Big|_\zeta \frac{\partial \hat{\phi}^{\kappa_1 \kappa_2}}{\partial \zeta} \Big|_\zeta \\ &= \frac{\mathrm{d}\zeta}{\mathrm{d}x_3} \Big|_\zeta \frac{\hat{\phi}^{\kappa_1 \kappa_2}(\zeta + \delta\zeta/2) - \hat{\phi}^{\kappa_1 \kappa_2}(\zeta - \delta\zeta/2)}{\delta\zeta} \\ &\quad + \mathcal{O}\left(\delta\zeta^2\right) \end{aligned} \tag{13}$$

for any field $\phi$, where $\delta\zeta \equiv 1/N_3$ is the grid spacing in the $\zeta$ coordinate. Vertical derivatives are therefore evaluated at the opposite set of grid points to the ones where $\phi$ is defined, as typical for staggered grids. At the boundary, one-sided second-order stencils are employed. This approximation of the vertical derivatives results in a truncation error of order $\mathcal{O}\left(\delta\zeta^2\right)$.

The non-linear advection term of Eq. (7) requires further approximations. First, some of the terms (e.g., $i = 1, j = 3$) are evaluated at the opposite set of vertical grid points than where they are required and have to be interpolated. With the simple interpolation $\hat{\phi}(\zeta) = 1/2\left(\hat{\phi}(\zeta - \delta\zeta/2) + \hat{\phi}(\zeta + \delta\zeta/2)\right) + \mathcal{O}\left(\delta\zeta^2\right)$, the truncation error generally increases but remains of the order $\mathcal{O}\left(\delta\zeta^2\right)$. Furthermore, the double sum can only be computed over the resolved range of wavenumbers, i.e., $|\kappa_1'| \leq (N_1 - 1)/2$ and $|\kappa_2'| \leq (N_2 - 1)/2$, producing another truncation error that decreases exponentially with the number of resolved wavenumbers. The same applies to the non-linear expressions involved in the evaluation of $\tau_{ij}^{\mathrm{sgs}}$. This discretization of the advection term in rotational form conserves kinetic energy in the absence of time-integration errors (Mansour et al., 1979) as long as the grid is uniform.

To simplify the computation of non-linear terms and avoid evaluating expensive convolutions, those terms are computed on $N_1^{\mathrm{PD}} \times N_2^{\mathrm{PD}}$ equidistant grid points in the physical domain, relying on the fast Fourier transform (FFT) algorithm for forward and backward transforms (Orszag, 1969, 1971b). In principle, $N_1^{\mathrm{PD}}$ and $N_2^{\mathrm{PD}}$ are parameters that can be chosen independently of $N_1$ and $N_2$, but the choice of $N_i^{\mathrm{PD}} \geq 1 + 3\kappa_i^{\mathrm{max}}$ avoids introducing aliasing errors for a simple product of two variables such as the resolved advection term (Patterson and Orszag, 1971). In this case, the physical-domain evaluation is equivalent to a true spectral Galerkin method computing the convolution of Eq. (7) over all resolved wavenumbers. Contributions from wavenumbers $|\kappa_i| > (N_i - 1)/2$ are discarded upon return to the Fourier domain, and vertical derivatives can be computed before or after the horizontal Fourier transforms as the two operations commute.

For the more complex non-linear expressions introduced by the SGS model, full dealiasing is generally not feasible and physical-domain evaluations incur aliasing errors in addition to the truncation errors. This approach, dubbed the pseudospectral method by Orszag (1971b), can achieve similar accuracy to a Galerkin method (Orszag, 1972). While it is common to set $N_i^{\mathrm{PD}} = N_i$ for pseudospectral approximations and only discard the Nyquist wavenumber, $N_i^{\mathrm{PD}}$ can in principle be chosen freely for more control over aliasing errors. Furthermore, it can be beneficial to choose different values of $N_i^{\mathrm{PD}}$ for each non-linear term, since computing the resolved advection requires only 9 transforms and is known to be sensitive to aliasing errors (Kravchenko and Moin, 1997; Margairaz et al., 2018), while the evaluation of the SGS model requires 15 transforms.

Combining the velocity components into a single vector $\hat{\boldsymbol{u}}$ of length $N_1 \times N_2 \times (3N_3 - 1)$, the spatially discretized momentum equation can be written as

$$\frac{\mathrm{d}\hat{\boldsymbol{u}}}{\mathrm{d}t} = \mathrm{Adv}(\hat{\boldsymbol{u}}) + \frac{1}{Re} \boldsymbol{\Delta} \hat{\boldsymbol{u}} + \frac{1}{Re} \hat{\boldsymbol{b}}_\Delta - \mathbf{G}\hat{\boldsymbol{p}} + \hat{\boldsymbol{f}} . \tag{14}$$

Here, $\mathrm{Adv}(\cdot)$ is the non-linear advection operator that includes both resolved and subgrid-scale contributions. $\boldsymbol{\Delta}$ is the linear operator that computes the Laplacian of each velocity component, with $\hat{\boldsymbol{b}}_\Delta$ denoting the contributions from vertical boundary conditions. $\mathbf{G}$ is the linear operator that computes the gradient of the pressure, discretized as a vector $\hat{\boldsymbol{p}}$ of size $N_1 \times N_2 \times N_3$. The forcing term $\hat{\boldsymbol{f}}$, a vector of the same length as $\hat{\boldsymbol{u}}$, can be constant in time and space (e.g., pressure-driven channel flow), vary in time only (e.g., constant-mass-flux channel flow), vary in space only (e.g., baroclinic flow), or even vary in time and space as a function of $\hat{\boldsymbol{u}}$, which can be used to model vegetation drag or to represent complex geometry with an immersed boundary method. The discrete continuity equation,

$$\mathbf{D}\hat{\boldsymbol{u}} + \hat{\boldsymbol{b}}_\mathrm{D} = \boldsymbol{0} , \tag{15}$$

contains the linear divergence operator $\mathbf{D}$ with the contributions $\hat{\boldsymbol{b}}_\mathrm{D}$ from the vertical boundary conditions of $u_3$.

This hybrid approach, relying on spectral approximations in horizontal direction (pseudospectral for the evaluation of

$\tau_{ij}^{\mathrm{sgs}}$) and second-order-accurate finite differences in vertical direction, has long been employed for computational studies of turbulent flows in channel geometries (Moin and Kim, 1982; Moeng, 1984; Albertson and Parlange, 1999a, b). It combines the fast convergence and low dissipation of spectral methods (Giacomini and Giometto, 2021) with the ease of parallelization and simple handling of boundary conditions of finite differences. Conversely, handling complex domains and non-periodic boundaries can be problematic, though still possible (Chester et al., 2007; Schmid, 2015; Li et al., 2016).

Following Perot (1993), we obtain the expressions for time integration through a block LU decomposition of the fully discretized equations. This results in expressions in the style of the fractional step method (Chorin, 1968; Temam, 1969), but it avoids the need for boundary conditions for the intermediate velocity and the pressure on a staggered grid and can easily be adapted when new terms are included or different numerical methods are employed.

Adams–Bashforth methods solving ordinary differential equations of the form $\mathrm{d}\boldsymbol{u}/\mathrm{d}t = f(\boldsymbol{u})$, $\boldsymbol{u}(t_0) = \boldsymbol{u}_0$ can be written as $\boldsymbol{u}^{(n+1)} = \boldsymbol{u}^{(n)} + \Delta t \sum_{i=0}^{s-1} \beta_i f(\boldsymbol{u}^{(n-i)})$, where $\beta_i$ are the coefficients of the method, $s$ is the order of accuracy, and superscripts denote the time step (Hairer et al., 1993). With $\beta_0 = 1$ this corresponds to the forward Euler method ($s = 1$), while $\beta_0 = 3/2$, $\beta_1 = -1/2$ gives second-order accuracy ($s = 2$). Applied to the momentum Eq. (14), this can be written as

$$\hat{\boldsymbol{u}}^{(n+1)} = \hat{\boldsymbol{u}}^{(n)} + \Delta t \sum_{i=0}^{s-1} \beta_i \left( \mathrm{F}(\hat{\boldsymbol{u}}^{(n-i)}) - \mathbf{G}\hat{\boldsymbol{p}}^{(n-i)} \right), \qquad (16)$$

where terms are grouped with the definition $\mathrm{F}(\hat{\boldsymbol{u}}) \equiv \mathrm{Adv}(\hat{\boldsymbol{u}}) + \frac{1}{Re}\boldsymbol{\Delta}\hat{\boldsymbol{u}} + \frac{1}{Re}\hat{\boldsymbol{b}}_\Delta + \hat{\boldsymbol{f}}$ to simplify the notation. Together with the continuity Eq. (15), the fully discretized equations become

$$\hat{\boldsymbol{u}}^{(n+1)} + \mathbf{G}\hat{\boldsymbol{\varphi}}^{(n+1)} = \hat{\boldsymbol{u}}^{(n)} + \Delta t \sum_{i=0}^{s-1} \beta_i \mathrm{F}\left(\hat{\boldsymbol{u}}^{(n-i)}\right) \qquad (17)$$

and

$$\mathbf{D}\hat{\boldsymbol{u}}^{(n+1)} = -\hat{\boldsymbol{b}}_{\mathrm{D}} \qquad (18)$$

if we group the pressure contributions with $\hat{\boldsymbol{\varphi}}^{(n+1)} \equiv \Delta t \sum_{i=0}^{s-1} \beta_i \hat{\boldsymbol{p}}^{(n-i)}$. Similarly, explicit $s$-stage Runge–Kutta methods can be written as $\boldsymbol{u}^{(n,i)} = \sum_{k=0}^{i-1} \left( \alpha_{ik}\boldsymbol{u}^{(n,k)} + \Delta t \beta_{ik} f(\boldsymbol{u}^{(n,k)}) \right)$ for $i = 1, \ldots, s$, with $\boldsymbol{u}^{(n,0)} = \boldsymbol{u}^{(n)}$ and $\boldsymbol{u}^{(n+1)} = \boldsymbol{u}^{(n,s)}$ (Gottlieb et al., 2009). This is referred to as the Shu–Osher form (Shu and Osher, 1988), where the coefficients $\alpha_{ik}$ and $\beta_{ik}$ are not uniquely determined by the Butcher tableau of the method and can be chosen to minimize storage requirements. In this case, the fully discretized equations can be written as

$$\hat{\boldsymbol{u}}^{(n,i)} + \mathbf{G}\hat{\boldsymbol{\varphi}}^{(n,i)} = \sum_{k=0}^{i-1} \left( \alpha_{ik}\hat{\boldsymbol{u}}^{(n,k)} + \Delta t \beta_{ik} \mathrm{F}\left(\hat{\boldsymbol{u}}^{(n,k)}, t^{(n,k)}\right) \right) \qquad (19)$$

and

$$\mathbf{D}\hat{\boldsymbol{u}}^{(n,i)} = -\hat{\boldsymbol{b}}_{\mathrm{D}} \qquad (20)$$

with the definition $\hat{\boldsymbol{\varphi}}^{(n,i)} \equiv \Delta t \sum_{k=0}^{i-1} \beta_{ik} \, \hat{\boldsymbol{p}}^{(n,k)}$.

Each step or stage requires the solution of a system of equations in the form $\hat{\boldsymbol{u}} + \mathbf{G}\hat{\boldsymbol{\varphi}} = \hat{\boldsymbol{a}}$ and $\mathbf{D}\hat{\boldsymbol{u}} = \hat{\boldsymbol{b}}$. This can be solved with the LU decomposition:

$$\begin{aligned}
\begin{pmatrix} \mathbf{I} & \mathbf{G} \\ \mathbf{D} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{u}} \\ \hat{\boldsymbol{\varphi}} \end{pmatrix} &= \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{D} & -\mathbf{D}\mathbf{G} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{G} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{u}} \\ \hat{\boldsymbol{\varphi}} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{D} & -\mathbf{D}\mathbf{G} \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{u}}^\star \\ \hat{\boldsymbol{\varphi}} \end{pmatrix} \\
&= \begin{pmatrix} \hat{\boldsymbol{a}} \\ \hat{\boldsymbol{b}} \end{pmatrix},
\end{aligned} \qquad (21)$$

where $\hat{\boldsymbol{u}}^\star \equiv \hat{\boldsymbol{u}} + \mathbf{G}\hat{\boldsymbol{\varphi}}$ has been introduced. The steps to compute the solution are therefore

$$\begin{aligned}
\hat{\boldsymbol{u}}^\star &= \hat{\boldsymbol{a}}, \\
\mathbf{D}\mathbf{G}\hat{\boldsymbol{\varphi}} &= \mathbf{D}\hat{\boldsymbol{u}}^\star - \hat{\boldsymbol{b}}, \text{ and} \\
\hat{\boldsymbol{u}} &= \hat{\boldsymbol{u}}^\star - \mathbf{G}\hat{\boldsymbol{\varphi}},
\end{aligned} \qquad (22)$$

where the second step requires solving a linear system. Since the operator ($\mathbf{D}\mathbf{G}$) has no coupling between different wavenumbers, Eq. (22) can be decomposed into $N_1 \times N_2$ tridiagonal systems of size $N_3$. For $\kappa_1 = \kappa_2 = 0$ the system is singular due to the fact that the governing equations only include the gradient of the pressure and do not place any restrictions on the absolute magnitude of the pressure variable. This system therefore has to be solved iteratively or the equations have to be regularized, e.g., by specifying an arbitrary value for one element of $\hat{\boldsymbol{\varphi}}$. The current implementation relies on the Thomas algorithm (Quarteroni et al., 2007) to solve the tridiagonal systems and includes the forward Euler and second-order Adams–Bashforth methods for time integration as well as the strong stability preserving Runge–Kutta methods SSPRK (2,2) and SSPRK (3,3) (Gottlieb et al., 2009). Adding other explicit methods is straightforward, provided they can be formulated in a similar fashion.

The simulation code is written in the Julia programming language, relying on the Julia bindings to the FFTW library (Frigo and Johnson, 2005) for fast Fourier transforms. For parallelization, the domain is vertically split into up to $N_3$ blocks that are computed by separate processes exchanging information through the Message Passing Interface (MPI).

## 3 Model validation

The validation efforts presented in this section aim to confirm that the numerical methods are implemented faithfully and that these methods produce physically relevant results. To maintain this confidence as the code is inevitably modified, a
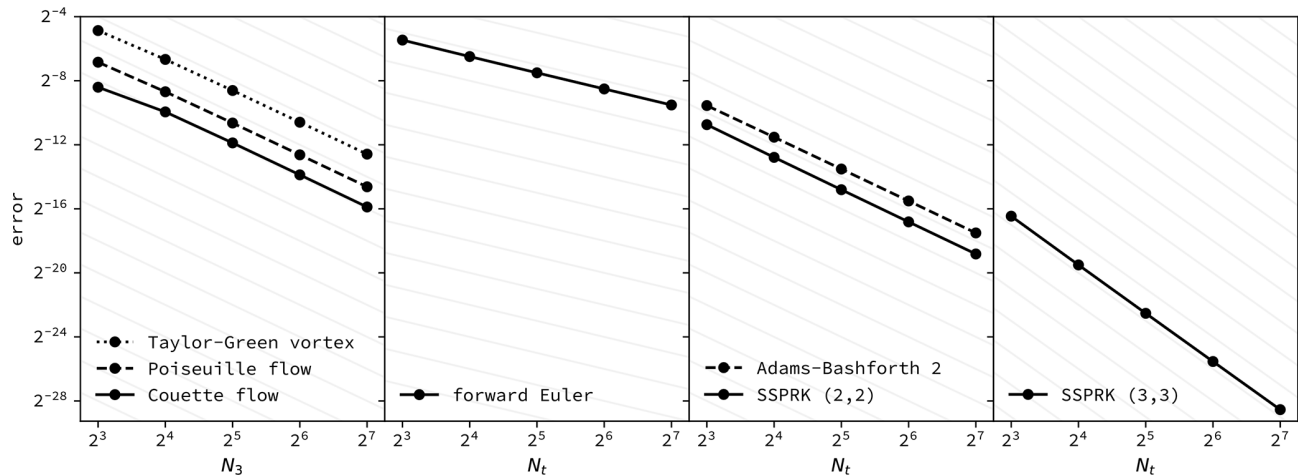
**Figure 1.** Error convergence for transient two-dimensional laminar flows. The left panel shows second-order convergence as the vertical grid resolution is refined for flows set up along the vertical direction and a randomly chosen horizontal direction. The other panels show first-, second-, and third-order convergence as the time resolution is refined for a Taylor–Green vortex set up along the two horizontal directions, in which case the spatial discretization is exact and the order of convergence of the time integration methods is measured. Grid lines show the formal order of convergence for each case.

focus is placed on automated tests that can be rerun after every change. A set of automated unit tests verifies the expected order of accuracy when computing individual terms of the discretized equations for prescribed velocity fields and when applying the time-integration algorithms to ordinary differential equations. A set of automated integration tests verifies that the solution to canonical transient two-dimensional laminar flows can be simulated with the expected order of accuracy. Finally, fully turbulent flow solutions are computed and compared to published results produced with different codes. These tests are not automated since they require significant computational resources and have no analytical solution to compare against so there is some degree of judgment required to evaluate the quality of the solution.

The automated tests of individual terms make use of the fact that the implemented numerical methods are exact for certain velocity fields. The diffusion term and the pressure solver are exact for a function that is the product of truncated Fourier series along horizontal dimensions and a quadratic polynomial in vertical direction. The advection term is only exact for a linear function in vertical direction due to linear interpolations, although the term is still second-order accurate. By constructing such a function with randomized parameters, each term can be computed numerically as well as analytically, and matching values give a high degree of confidence in the correctness of the implementation. Furthermore, we can verify the order of convergence when computing the terms at different grid resolutions for a velocity field that cannot be handled exactly by the implemented methods. The time integration methods are verified in a similar way by solving ordinary differential equations that have analyti-

cal solutions with different time steps. We also verify that the tridiagonal solver is exact for random inputs.

To test the full solver including time integration, the automated tests include a number of laminar flow problems, currently the transient Poiseuille and Couette flows as well as decaying Taylor–Green vortices. Numerical solutions computed at different resolutions are then compared to the analytical solution to ensure that the order of convergence corresponds to formal order of the numerical methods, as shown in Fig. 1. Dimensional parameters such as domain sizes and velocity scales are again chosen randomly since parameters that are zero or unity can mask errors in the solution. For Poiseuille and Couette flows, this includes the horizontal direction of the flow. The two-dimensional Taylor–Green vortices are oriented both in horizontal and vertical planes. For the former, the spatial discretization is exact so the test case verifies the order of convergence of the time integration method.

The above test cases have been verified in single-process (serial) mode as well as in multi-process (parallel) mode. Multi-process tests are run both with a vertical resolution greater than and equal to the number of processes since those configurations sometimes rely on different code paths. Since the tests are automated and run within minutes on consumer hardware, they can be rerun whenever changes are made to the code to ensure that any future version of the code still satisfies all the tested properties.

Turbulence-resolving flow simulations require substantially more computation and are therefore not included in the automated tests that are meant to be run routinely during code development. The validation cases presented below are chosen such that they represent scientifically relevant flow sys-

tems while keeping the computational cost moderate. Each case can be simulated in about 2 h using 32 MPI processes on a single compute node of the Intel Skylake generation.

The results of direct numerical simulations are not supposed to depend on the exact method used for modeling the flow, at least for lower-order flow statistics. For relatively low Reynolds numbers, simulations have been run with many different codes and with a wide range of parameters such as domain sizes, aspect ratios, and grid resolutions, so the expected simulation results are well-established and have been validated against wind-tunnel measurements (Kim et al., 1987; del Álamo and Jiménez, 2003; Lee and Moser, 2015).

In Fig. 2, we show a comparison of a closed-channel flow at $Re_\tau \approx 180$ with data published by Lee and Moser (2015). The friction Reynolds number $Re_\tau = u_\tau \delta / \nu$ is based on the half-channel height $\delta$ and the friction velocity $u_\tau^2 = \nu \frac{\partial u_1}{\partial x_3}\big|_{x_3=0}$ here. The simulation is run with a bulk Reynolds number of $Re_b = U_b \delta / \nu = 20\,000/7$, where the vertically averaged bulk velocity $U_b$ is held constant by a dynamically adjusted pressure forcing. The solution is computed in a domain of size $4\pi\delta$ in streamwise and $2\pi\delta$ in crossstream direction. The velocity field is discretized with $255 \times 191$ Fourier modes at 96 vertical grid points that are spaced according to a sinusoidal grid transform $x_3(\zeta) = \delta + \delta \sin((2\zeta - 1)\eta\pi/2)/\sin(\eta\pi/2)$ with $\eta = 0.97$. The mean statistics computed over $\sim 17.5$ large-eddy turnover times $T_\tau = \delta/u_\tau$ after a spinup time of $\sim 3.5\,T_\tau$ closely match the results from Lee and Moser (2015).

For large-eddy simulation, validation is not as straightforward since results remain relatively sensitive to differences in the modeling approach and in the grid resolution. To validate the new implementation, we limit ourselves to a comparison with a pre-existing Fortran implementation of the same physical and numerical models (Giometto et al., 2017) and refer to previous publications for validation studies and discussions of limitations of the modeling approach (Porté-Agel et al., 2000; Yue et al., 2007, 2008; Giometto et al., 2016).

In Fig. 3 we show the results of this comparison for an open-channel flow at $Re_\tau = 10^8$ driven by a constant body force $f_1$. The friction Reynolds number $Re_\tau = u_\tau h/\nu$ is based on the channel height $h$ and the friction velocity $u_\tau^2 = h f_1$ here. The lower surface is characterized by a roughness length $z_0$ that results in a non-dimensional channel height of $h/z_0 = 10^4$. The solution is computed in a domain of size $2\pi h$ in streamwise and $(4/3)\pi h$ in cross-stream direction. The velocity field is discretized with $63 \times 63$ Fourier modes at 64 equidistant vertical grid points. The mean statistics computed over 200 large-eddy turnover times $T_\tau = h/u_\tau$ after a spinup time of $50\,T_\tau$ closely match for the two separate implementations.

Combined, these validation efforts provide ample evidence that the implementation matches the mathematical formulation of the methods and that those methods are capable of accurately simulating flow physics, within the limitations of the physical models. A comprehensive set of easily repeatable validation tests serves both to verify the current implementation and to ensure that future developments do not jeopardize correctness. This should not only facilitate adding new functionality but also help make changes to existing functionality and avoid getting locked into design decisions that might prove suboptimal for future developments.

## 4 Performance and scaling

Defined in a broad way, performance can be understood as the time required to obtain a solution at the required quality given the available computational resources. We can examine how the time changes as a function of the required quality and the available resources (relative performance) or how fast different methods arrive at a solution for fixed quality and resources (absolute performance). However, it is difficult to measure the overall quality of a turbulent flow simulation in a quantitative way since the system is chaotic and analytic solutions are not available. Furthermore, there is a great diversity of computational resources that vary along important dimensions such as floating point operations per second (FLOPS), memory bandwidth and latency, network bandwidth and latency, and many more. We therefore narrow the scope of the performance analysis to the question of the time required to obtain a solution given the specified simulation parameters and the number of compute nodes, as measured on a fairly typical high-performance computing (HPC) system.

For the implemented explicit time integration schemes, the computational cost of a single evaluation of the right-hand side of $d\boldsymbol{u}/dt = f(\boldsymbol{u})$ fully characterizes the overall cost of a simulation, which is a simple function of the number of steps and the evaluations per step (i.e., stages of a Runge–Kutta method). The computational cost of a single evaluation of $f(\boldsymbol{u})$ depends primarily on the number of Fourier modes and vertical grid points and whether a subgrid-scale term is modeled (LES) or not (DNS). The impact of other parameters such as the type of pressure forcing (constant flux vs. constant force), boundary conditions, and grid transformations is imperceptible.

To assess relative performance, Fig. 4 shows the time required to compute the advection, diffusion, and pressure terms for different numbers of compute nodes and vertical grid points. The figure displays both strong scaling, where the number of processes is varied for a problem of fixed size, and weak scaling, where the problem size is varied in proportion to the computational resources. These results shows that the advection term contributes most to the overall cost, while the pressure term exhibits the most problematic scaling behavior.

Computing the advection term is a global operation in horizontal direction but only involves neighboring nodes in vertical direction. The bulk of the computational work consists
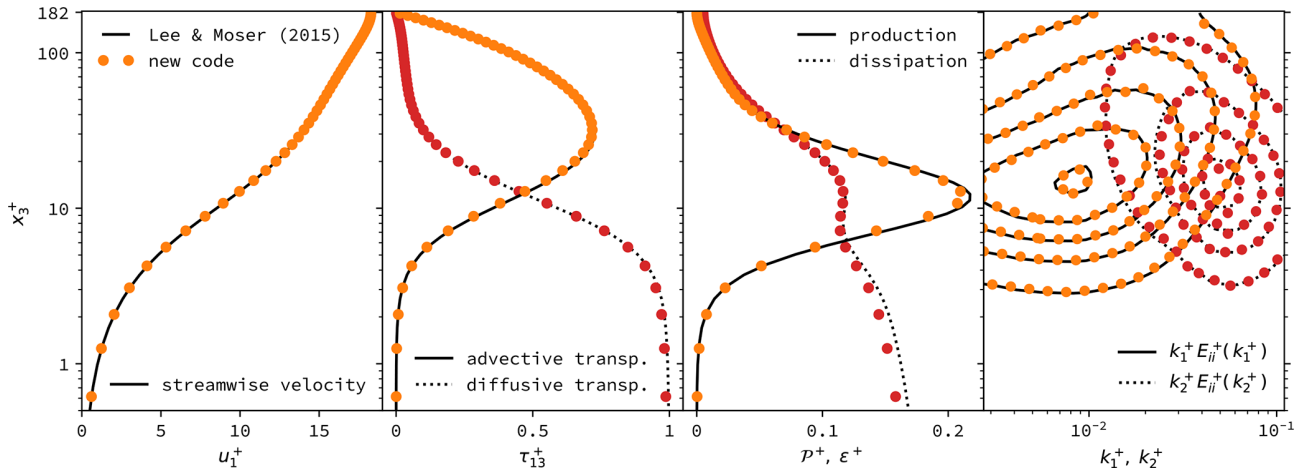
**Figure 2.** Direct numerical simulation (DNS) of a turbulent channel flow at $Re_\tau \approx 180$, validated against data published by Lee and Moser (2015). Mean profiles are shown for the streamwise velocity $u_1^+$, the advective transport $u_1^+ u_3^+$, the diffusive transport $\partial u_1^+ / \partial x_3^+$, as well as the production $\mathcal{P}^+$ and (pseudo)dissipation $\varepsilon^+$ of turbulent kinetic energy. The last panel shows contours of the premultiplied turbulent kinetic energy spectra $E_{ii}^+$ along the streamwise ($k_1^+ = 2\pi\kappa_1/L_1^+$) and cross-stream ($k_2^+ = 2\pi\kappa_2/L_2^+$) direction. The superscript $^+$ marks values in inner units, i.e., non-dimensionalized with the friction velocity $u_\tau$ and the kinematic viscosity $\nu$.
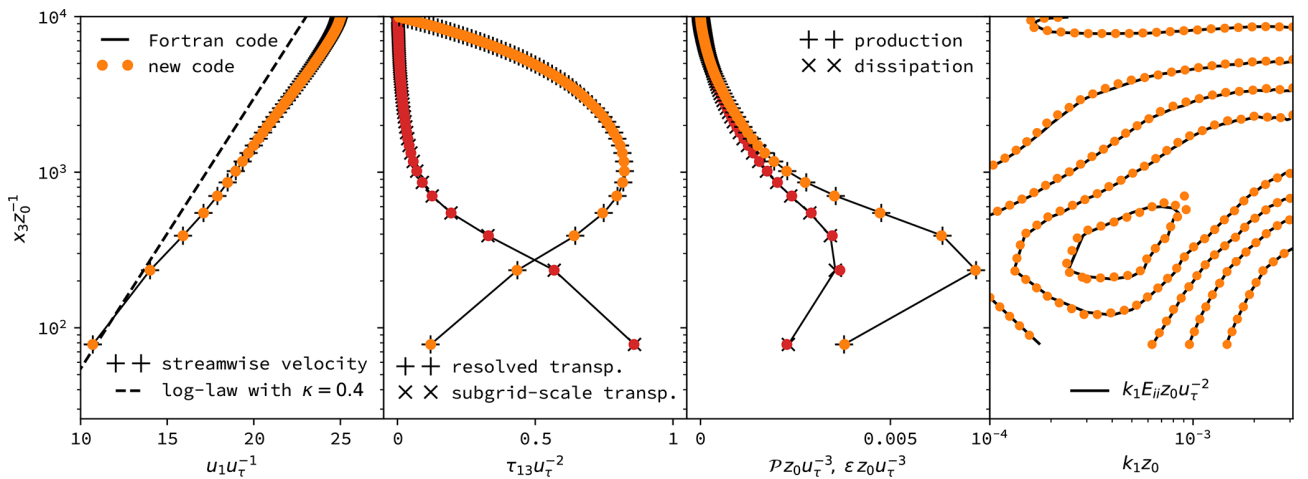


**Figure 3.** Large-eddy simulation (LES) of a turbulent channel flow at $Re_\tau = 10^8$ with an aerodynamically rough wall and a channel height of $h/z_0 = 10^4$, validated against a tried and tested Fortran code with the same numerical approach (Giometto et al., 2017). All values are non-dimensionalized with the friction velocity $u_\tau$ and the roughness length $z_0$. Mean profiles are shown for the streamwise velocity $u_1$, the resolved transport $u_1 u_3$, the subgrid-scale transport $\tau_{13}^{\text{sgs}}$, as well as the production $\mathcal{P}$ and (pseudo-)dissipation $\varepsilon$ of resolved turbulent kinetic energy. The last panel shows contours of the resolved turbulent kinetic energy spectra $E_{ii}$ along the streamwise direction, premultiplied with the wavenumber $k_1 = 2\pi\kappa_1/L_1$.

of computing discrete Fourier transforms, which are local to each MPI process and scale as $\mathcal{O}(N \log N)$, where $N$ is the number of modes. It appears that this cost dominates over the cost of communication, resulting in near-perfect strong and weak scaling. When computing subgrid-scale stresses with a static Smagorinsky model, additional transforms are required and the cost increases to almost twice as much without affecting the scaling behavior.

Computing the pressure term has no data dependency between horizontal modes but is a sequential, global process

in vertical direction (Thomas algorithm). Horizontal modes can be processed in batches to stagger the sequential passes up and down the domain, where the size of those batches is a tuning parameter that represents a trade-off between maximizing parallelism and minimizing per-batch overhead. The resulting performance shows imperfect weak and strong scaling. Scaling appears to improve when there are more vertical grid points per process, increasing the work-to-communication ratio. While the overall cost appears to remain at most about a quarter of the cost of the advection term,
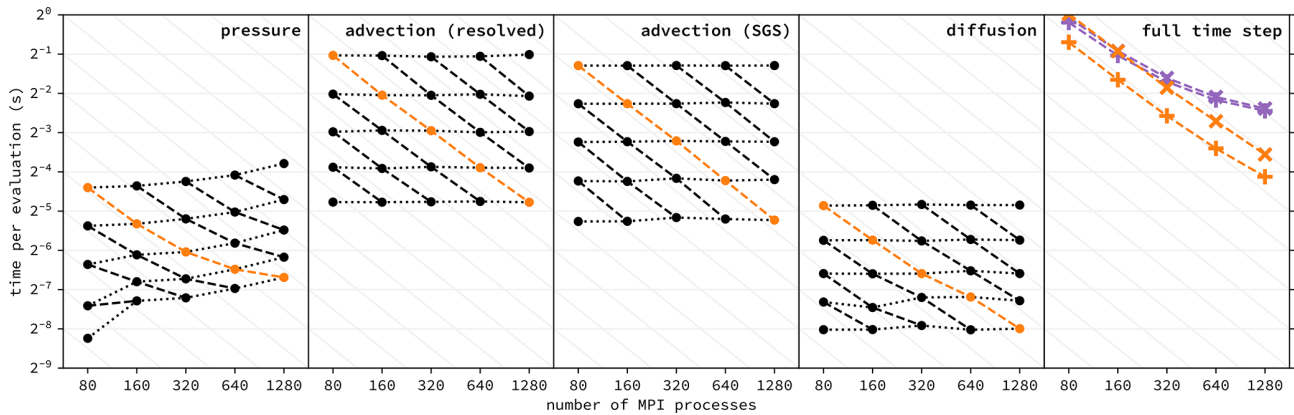
**Figure 4.** Performance and scaling of individual terms and full time step, as measured on Intel Xeon Ice Lake nodes of the Stampede2 system at the Texas Advanced Computing Center. Individual terms are computed at a resolution of $256 \times 256 \times \lambda N_\mathrm{p}$, where $\lambda \in \{1, 2, 4, 8, 16\}$ is the number of vertical grid points per MPI process and $N_\mathrm{p}$ is the total number of MPI processes. Dotted lines indicate weak scaling, dashed lines indicate strong scaling, and the grid lines correspond to perfect scaling. The last panel shows the overall performance for the resolution $256 \times 256 \times 1280$ (highlighted in orange on other panels) in comparison to a pre-existing Fortran implementation of the same numerical approach (Giometto et al., 2017). DNS performance is shown with the symbol $+$, LES performance with the symbol $\times$.

it is possible that the two costs are even closer for some combinations of hardware configurations and simulation parameters, in which case it could be worth optimizing the batch size parameter of the pressure solver.

Computing the diffusion term only involves neighboring vertical grid points and has no global data dependencies. This results in near-perfect weak scaling. Strong scaling is not quite perfect, which is explained by the fact that there is very little work to do for each grid point so the work-to-communication ratio is low. This has no discernible effect on the overall scaling behavior however, as computing the diffusion term is always at least an order of magnitude less work than computing the advection term.

To assess absolute performance, Fig. 4 includes a comparison with a Fortran code that implements the same numerical methods (Giometto et al., 2017). While such a comparison does not answer the question of whether either code is making optimal use of the computational resources, it does respond to the practical question of whether there are any performance trade-offs when substituting the new code for a codebase that has been actively used for turbulence research for over 2 decades. The comparison shows that the overall performance of both implementations is of a similar order of magnitude, with the new Julia implementation showing somewhat better scaling and significantly faster DNS performance.

It appears that the new Julia code has avoided introducing excessive overhead without much effort devoted to performance optimization. That it even surpasses the performance of the Fortran implementation is likely explained by two factors. First, the new code is formulated with the Fourier domain representation at its center, which makes it easier to avoid unnecessary Fourier transforms than in the physical-space formulation of the Fortran implementation. Second, the new code makes different trade-offs between work and communication which appear to be more suitable for modern hardware. Some of these insights will flow back to the Fortran implementation, reducing the performance discrepancy between the two codes.

Overall, the performance characteristics of the new code are as expected. The computational cost is dominated by the Fourier transforms necessary to compute the non-linear term, while the pressure solver shows the least favorable scaling properties, and the overall performance is comparable to a Fortran implementation of an equivalent numerical scheme. The analysis shows that for the implemented numerical scheme and current HPC hardware, performance is optimized by reducing the number and size of Fourier transforms and choosing an efficient implementation of the fast Fourier transform algorithm. Other details matter less as long as the computational cost can be kept significantly below the cost of the non-linear term.

Future improvements are likely to focus on parallelism along horizontal coordinate directions, either through multithreaded CPU code or through GPGPU computation, allowing the code to scale to larger systems. There is also room for optimization in how communication is handled, which might become important if the work-to-communication is decreased through additional parallelism. However, performance optimizations always have to be weighed against their impact on code simplicity and ease of adaptation. Since the computational cost of flow simulations is a strongly non-linear function of the grid resolution, large performance differences are required for a practical difference in the scientific problems that can be tackled.

**On the suitability of Julia for high-performance computation**

Conceptually, there are two main differences in the performance characteristics of Julia compared to languages like Fortran, C, and C++. The first difference concerns the handling of types. For a statically typed language like Fortran, the type of every variable is specified and therefore available to the compiler, which can use the information to generate efficient machine code. In Julia, machine code is generated when a function is called for the first time, at which point the types of the function arguments are known. For subsequent function calls with the same argument types, the compiled code is reused while a new copy of the function is compiled for calls with different argument types. If the types of all variables inside a function can be derived from the type of function arguments, the compiler has access to the same information as for a statically typed language and can in principle generate the same or equivalent machine code. The second difference concerns the handling of memory allocations. For a language with (semi-)manual memory management like Fortran, memory is allocated and freed either manually with explicit commands or following deterministic rules. In Julia, memory is automatically allocated whenever required for the operation that is performed and the program is periodically interrupted to examine which of the memory is still in use and which memory can be freed (garbage collection). Vectorized Julia code written in a naive way often allocates large amounts of memory for intermediate results, but writing Julia code that operates in place and avoids such allocations is relatively easy with experience. Therefore, there is no a priori reason that code written in Julia should be significantly slower (or faster) than code written in Fortran, C, or C++, although Julia does make it much easier to accidentally include expensive operations that result in poor performance.

Julia was chosen as implementation language for this code with the goal of improving the ease of development, hoping that the negative effect on performance could be kept minimal. However, the experience so far has shown that the net impact on performance might even be positive. Performance optimizations can be seen as a continuum from low level (e.g., vectorized CPU instructions, optimal use of CPU cache) to high level (e.g., choice of algorithms, speed–accuracy trade-offs). At the lower end of this range, Julia relies on the LLVM compiler framework and a number of pre-existing libraries, making use of countless hours of optimization work, but Julia also makes it rather easy to write code it cannot optimize very well. Maintaining close-to-optimal performance therefore requires regular measurements and occasional fixes. High-level optimizations on the other hand require understanding the performance characteristics of different approaches and choosing the right one, often by measuring the performance of different implementations. This type of optimization work benefits from the Julia language features and the ease of integrating packages from a growing ecosystem. The impact of language choice on performance depends not so much on what optimizations are theoretically possible but which ones are simple enough that they are done in practice. It is therefore possible that the choice of Julia will be a benefit rather than a drawback for the performance of this code over its lifetime.

## 5   Conclusions

Turbulence-resolving ABL flow simulations are subject to a number of competing requirements that have to be considered when developing simulation code. Availability of physical and numerical models, performance and scalability, ease of use and ease of modification, safeguards against implementation and usage errors, as well as license terms may vary considerably, and trade-offs are often inevitable. The Julia programming language is promising more favorable trade-offs by offering the ergonomics of a modern high-level language without sacrificing performance.

In this paper, we have introduced a new code for turbulence-resolving flow simulations, designed for the requirements of atmospheric boundary-layer research and written in Julia. The performance is shown to be in line with a Fortran implementation of the same modeling approach. In fact, it even appears that easier experimentation with algorithmic approaches and implementation trade-offs might have a stronger impact on performance than the remaining computational overhead compared to highly optimized Fortran compilers.

The code also places a focus on automated testing and minimizing the chances for errors both during development and usage. This is particularly important in exploratory research, where the expected behavior of a new model or flow system is not known a priori, and it is difficult to discern between inconspicuous errors and novel results.

The code provides the core functionality for both direct numerical simulation and large-eddy simulation in channel-flow geometries. In the future, we expect to expand the scope by adding functionality such as more advanced subgrid-scale models, support for temperature, humidity and transport of passive scalars, and partially resolved complex terrain.

# References

Albertson, J. D. and Parlange, M. B.: Surface length scales and shear stress: implications for land–atmosphere interaction over complex terrain, Water Resour. Res., 35, 2121–2132, https://doi.org/10.1029/1999wr900094, 1999a.

Albertson, J. D. and Parlange, M. B.: Natural integration of scalar fluxes from complex terrain, Adv. Water Resour., 23, 239–252, https://doi.org/10.1016/s0309-1708(99)00011-1, 1999b.

Bardina, J., Ferziger, J. H., and Reynolds, W. C.: Improved subgrid-scale models for large-eddy simulation, in: 13th Fluid and PlasmaDynamics Conference, American Institute of Aeronautics and Astronautics, https://doi.org/10.2514/6.1980-1357, 1980.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B.: Julia: A fresh approach to numerical computing, SIAM Review, 59, 65–98, https://doi.org/10.1137/141000671, 2017.

Bou-Zeid, E., Meneveau, C., and Parlange, M. B.: A Scale-Dependent Lagrangian Dynamic Model for Large Eddy Simulation of Complex Turbulent Flows, Phys. Fluids, 17, 025105, https://doi.org/10.1063/1.1839152, 2005.

Chen, G., Xiong, Q., Morris, P. J., Paterson, E. G., Sergeev, A., and Wang, Y.-C.: OpenFOAM for computational fluid dynamics, Notices of the American Mathematical Society, 61, 354, https://doi.org/10.1090/noti1095, 2014.

Chester, S., Meneveau, C., and Parlange, M. B.: Modeling turbulent flow over fractal trees with renormalized numerical simulation, J. Comput. Phys., 225, 427–448, https://doi.org/10.1016/j.jcp.2006.12.009, 2007.

Chorin, A. J.: Numerical solution of the Navier–Stokes equations, Math. Comput., 22, 745–762, https://doi.org/10.1090/s0025-5718-1968-0242392-2, 1968.

Deardorff, J. W.: Similarity Principles for Numerical Integrations of Neutral Barotropic Planetary Boundary Layers and Channel Flows, J. Atmos. Sci., 26, 763–767, https://doi.org/10.1175/1520-0469(1969)026<0763:SPFNIO>2.0.CO;2, 1969.

Deardorff, J. W.: A Three-dimensional Numerical Investigation of the Idealized Planetary Boundary Layer, Geophys. Fluid Dynam., 1, 377–410, https://doi.org/10.1080/03091927009365780, 1970a.

Deardorff, J. W.: A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers, J. Fluid Mech., 41, 453–480, https://doi.org/10.1017/S0022112070000691, 1970b.

del Álamo, J. C. and Jiménez, J.: Spectra of the very large anisotropic scales in turbulent channels, Phys. Fluids, 15, L41, https://doi.org/10.1063/1.1570830, 2003.

Ferziger, J. H., Perić, M., and Street, R. L.: Computational methods for fluid dynamics, Springer Nature Switzerland AG, 4th edn., https://doi.org/10.1007/978-3-319-99693-6, 2020.

Frigo, M. and Johnson, S. G.: The design and implementation of FFTW3, Proc. IEEE, 93, 216–231, https://doi.org/10.1109/jproc.2004.840301, 2005.

Germano, M., Piomelli, U., Moin, P., and Cabot, W. H.: A dynamic subgrid-scale eddy viscosity model, Phys. Fluids A, 3, 1760–1765, https://doi.org/10.1063/1.857955, 1991.

Giacomini, B. and Giometto, M. G.: On the suitability of second-order accurate finite-volume solvers for the simulation of atmospheric boundary layer flow, Geosci. Model Dev., 14, 1409–1426, https://doi.org/10.5194/gmd-14-1409-2021, 2021.

Giometto, M. G., Christen, A., Meneveau, C., Fang, J., Krafczyk, M., and Parlange, M. B.: Spatial Characteristics of Roughness Sublayer Mean Flow and Turbulence Over a Realistic Urban Surface, Bound.-Lay. Meteorol., 160, 425–452, https://doi.org/10.1007/s10546-016-0157-6, 2016.

Giometto, M. G., Christen, A., Egli, P. E., Schmid, M. F., Tooke, R. T., Coops, N. C., and Parlange, M. B.: Effects of Trees on Mean Wind, Turbulence and Momentum Exchange Within and Above a Real Urban Environment, Adv. Water Resour., 106, 154–168, https://doi.org/10.1016/j.advwatres.2017.06.018, 2017.

Gottlieb, S., Ketcheson, D. I., and Shu, C.-W.: High order strong stability preserving time discretizations, J. Sci. Comput., 38, 251–289, https://doi.org/10.1007/s10915-008-9239-z, 2009.

Hairer, E., Nørsett, S. P., and Wanner, G.: Solving Ordinary Differential Equations I, Springer Series in Computational Mathematics, Springer-Verlag Berlin, Heidelberg, 2nd edn., https://doi.org/10.1007/978-3-540-78862-1, 1993.

Kim, J., Moin, P., and Moser, R. D.: Turbulence Statistics in Fully Developed Channel Flow At Low Reynolds Number, J. Fluid Mech., 177, 133, https://doi.org/10.1017/s0022112087000892, 1987.

Kravchenko, A. G. and Moin, P.: On the effect of numerical errors in large eddy simulations of turbulent flows, J. Comput. Phys., 131, 310–322, https://doi.org/10.1006/jcph.1996.5597, 1997.

Lee, M. and Moser, R. D.: Direct Numerical Simulation of Turbulent Channel Flow Up To $Re_\tau \approx 5200$, J. Fluid Mech., 774, 395–415, https://doi.org/10.1017/jfm.2015.268, 2015.

Li, Q., Bou-Zeid, E., and Anderson, W.: The Impact and Treatment of the Gibbs Phenomenon in Immersed Boundary Method Simulations of Momentum and Scalar Transport, J. Comput. Phys., 310, 237–251, https://doi.org/10.1016/j.jcp.2016.01.013, 2016.

Mansour, N. N., Moin, P., Reynolds, W. C., and Ferziger, J. H.: Improved Methods for Large Eddy Simulations of Turbulence, in: Turbulent Shear Flows I, edited by: Durst, F., Launder, B. E., Schmidt, F. W., and Whitelaw, J. H., 386–401, Springer-Verlag, Berlin Heidelberg, 1979.

Margairaz, F., Giometto, M. G., Parlange, M. B., and Calaf, M.: Comparison of dealiasing schemes in large-eddy simulation of neutrally stratified atmospheric flows, Geosci. Model Dev., 11, 4069–4084, https://doi.org/10.5194/gmd-11-4069-2018, 2018.

Maronga, B., Banzhaf, S., Burmeister, C., Esch, T., Forkel, R., Fröhlich, D., Fuka, V., Gehrke, K. F., Geletič, J., Giersch, S., Gronemeier, T., Groß, G., Heldens, W., Hellsten, A., Hoffmann, F., Inagaki, A., Kadasch, E., Kanani-Sühring, F., Ketelsen, K., Khan, B. A., Knigge, C., Knoop, H., Krč, P., Kurppa, M., Maamari, H., Matzarakis, A., Mauder, M., Pallasch, M., Pavlik, D., Pfafferott, J., Resler, J., Rissmann, S., Russo, E., Salim, M., Schrempf, M., Schwenkel, J., Seckmeyer, G., Schubert, S., Sühring, M., von Tils, R., Vollmer, L., Ward, S., Witha, B., Wurps, H., Zeidler, J., and Raasch, S.: Overview of the PALM model system 6.0, Geosci. Model Dev., 13, 1335–1372, https://doi.org/10.5194/gmd-13-1335-2020, 2020.

Mason, P. J. and Callen, N. S.: On the magnitude of the subgrid-scale eddy coefficient in large-eddy simulations of turbulent channel flow, J. Fluid Mech., 162, 439–462, https://doi.org/10.1017/s0022112086002112, 1986.

Mason, P. J. and Thomson, D. J.: Stochastic backscatter in large-eddy simulations of boundary layers, J. Fluid Mech., 242, 51–78, https://doi.org/10.1017/S0022112092002271, 1992.

Meneveau, C. and Katz, J.: Scale-Invariance and Turbulence Models for Large-Eddy Simulation, Annu. Rev. Fluid Mech., 32, 1–32, https://doi.org/10.1146/annurev.fluid.32.1.1, 2000.

Meneveau, C., Lund, T. S., and Cabot, W. H.: A Lagrangian dynamic subgrid-scale model of turbulence, J. Fluid Mech., 319, 353, https://doi.org/10.1017/s0022112096007379, 1996.

Moeng, C.-H.: A large-eddy-simulation model for the study of planetary boundary-layer turbulence, J. Atmos. Sci., 41, 2052–2062, https://doi.org/10.1175/1520-0469(1984)041<2052:alesmf>2.0.co;2, 1984.

Moin, P. and Kim, J.: Numerical investigation of turbulent channel flow, J. Fluid Mech., 118, 341–377, https://doi.org/10.1017/s0022112082001116, 1982.

Moin, P. and Mahesh, K.: Direct Numerical Simulation: A Tool in Turbulence Research, Annu. Rev. Fluid Mech., 30, 539–578, https://doi.org/10.1146/annurev.fluid.30.1.539, 1998.

Moin, P. and Verzicco, R.: On the suitability of second-order accurate discretizations for turbulent flow simulations, Eur. J. Mec. B/Fluids, 55, 242–245, https://doi.org/10.1016/j.euromechflu.2015.10.006, 2016.

Orszag, S. A.: Numerical methods for the simulation of turbulence, Phys. Fluids, 12, II–250, https://doi.org/10.1063/1.1692445, 1969.

Orszag, S. A.: Numerical simulation of incompressible flows within simple boundaries: accuracy, J. Fluid Mech., 49, 75–112, https://doi.org/10.1017/s0022112071001940, 1971a.

Orszag, S. A.: Numerical simulation of incompressible flows within simple boundaries. I. Galerkin (spectral) representations, Stud. Appl. Math., 50, 293–327, https://doi.org/10.1002/sapm1971504293, 1971b.

Orszag, S. A.: Comparison of pseudospectral and spectral approximation, Stud. Appl. Math., 51, 253–259, https://doi.org/10.1002/sapm1972513253, 1972.

Panton, R. L.: Incompressible Flow, John Wiley & Sons, Inc., Hoboken, New Jersey, https://doi.org/10.1002/9781118713075, 2013.

Patterson, Jr., G. S. and Orszag, S. A.: Spectral calculations of isotropic turbulence: efficient removal of aliasing interactions, Phys. Fluids, 14, 2538, https://doi.org/10.1063/1.1693365, 1971.

Perot, J. B.: An analysis of the fractional step method, J. Comput. Phys., 108, 51–58, https://doi.org/10.1006/jcph.1993.1162, 1993.

Piomelli, U. and Balaras, E.: Wall-layer models for large-eddy simulations, Annu. Rev. Fluid Mech., 34, 349–374, https://doi.org/10.1146/annurev.fluid.34.082901.144919, 2002.

Porté-Agel, F., Meneveau, C., and Parlange, M. B.: A Scale-Dependent Dynamic Model for Large-Eddy Simulation: Application To a Neutral Atmospheric Boundary Layer, J. Fluid Mech., 415, 261–284, https://doi.org/10.1017/s0022112000008776, 2000.

Quarteroni, A., Sacco, R., and Saleri, F.: Numerical mathematics, Texts in Applied Mathematics, Springer New York, https://doi.org/10.1007/b98885, 2007.

Schmid, M. F.: Resolution of the Gibbs Phenomenon for Navier–Stokes Simulations, Master's thesis, Ecole Polytechnique Fédérale de Lausanne, 2015.

Schmid, M. F., Giometto, M. G., Lawrence, G. A., and Parlange, M. B.: Data & code for "BoundaryLayerDynamics.jl v1.0: a modern codebase for atmospheric boundary-layer simulations", Zenodo [data set], https://doi.org/10.5281/zenodo.7954004, 2023a.

Schmid, M. F., Giometto, M. G., Lawrence, G. A., and Parlange, M. B.: BoundaryLayerDynamics.jl: v1.0.0, Zenodo [code], https://doi.org/10.5281/zenodo.10105035, 2023b.

Schumann, U.: Subgrid scale model for finite difference simulations of turbulent flows in plane channels and annuli, J. Comput. Phys., 18, 376–404, https://doi.org/10.1016/0021-9991(75)90093-5, 1975.

Scotti, A., Meneveau, C., and Lilly, D. K.: Generalized Smagorinsky model for anisotropic grids, Phys. Fluids A, 5, 2306–2308, https://doi.org/10.1063/1.858537, 1993.

Shu, C.-W. and Osher, S.: Efficient implementation of essentially non-oscillatory shock-capturing schemes, J. Comput. Phys., 77, 439–471, https://doi.org/10.1016/0021-9991(88)90177-5, 1988.

Skamarock, W. C., Klemp, J. B., Dudhia, J., Gill, D. O., Liu, Z., Berner, J., Wang, W., Powers, J. G., Duda, M. G., Barker, D. M., and Huang, X.-Y.: A description of the Advanced Research WRF Version 4, Tech. rep., National Center for Atmospheric Research, https://doi.org/10.5065/1dfh-6p97, 2021.

Smagorinsky, J.: General Circulation Experiments With The Primitive Equations, Mon. Weather Rev., 91, 99–164, https://doi.org/10.1175/1520-0493(1963)091<0099:gcewtp>2.3.co;2, 1963.

Stoll, R., Gibbs, J. A., Salesky, S. T., Anderson, W., and Calaf, M.: Large-eddy simulation of the atmospheric boundary layer, Bound.-Lay. Meteorol., 177, 541–581, https://doi.org/10.1007/s10546-020-00556-3, 2020.

Temam, R.: Sur l'approximation de la solution des équations de Navier–Stokes par la méthode des pas fractionnaires (II), Arch. Ration. Mech. An., 33, 377–385, https://doi.org/10.1007/bf00247696, 1969.

Towns, J., Cockerill, T., Dahan, M., Foster, I., Gaither, K., Grimshaw, A., Hazlewood, V., Lathrop, S., Lifka, D., Peterson, G. D., Roskies, R., Scott, J. R., and Wilkins-Diehr, N.: XSEDE: accelerating scientific discovery, Comput. Sci. Eng., 16, 62–74, https://doi.org/10.1109/mcse.2014.80, 2014.

Yue, W., Parlange, M. B., Meneveau, C., Zhu, W., Van Hout, R., and Katz, J.: Large-eddy simulation of plant canopy flows using plant-scale representation, Bound.-Lay. Meteorol., 124, 183–203, https://doi.org/10.1007/s10546-007-9173-x, 2007.

Yue, W., Meneveau, C., Parlange, M. B., Zhu, W., Kang, H. S., and Katz, J.: Turbulent kinetic energy budgets in a model canopy: comparisons between LES and wind-tunnel experiments, Environ. Fluid Mech., 8, 73–95, https://doi.org/10.1007/s10652-007-9049-0, 2008.