Supercomputing Frontiers and Innovations

2021, Vol. 8, No. 4

Scope

- Future generation supercomputer architectures
- Exascale computing
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Novel approaches to computing targeted to solve intractable problems
- Convergence of high performance computing, machine learning and big data technologies
- Distributed operating systems and virtualization for highly scalable computing
- Management, administration, and monitoring of supercomputer systems
- Mass storage systems, protocols, and allocation
- Power consumption minimization for supercomputing systems
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Scientific visualization in supercomputing environments
- Education in high performance computing and computational science

Editorial Board

Contents

DOI: 10.14529/jsfi210401

Technology for Supercomputer Simulation of Turbulent Flows in the Good New Days of Exascale Computing

Andrey V. Gorobets[1], *Alexey P. Duben*[1]

A technology for scale-resolving simulations of turbulent flows in the problems of aerodynamics and aeroacoustics is presented. It is based on the higher accuracy numerical schemes on unstructured mixed-element meshes and latest non-zonal hybrid approaches combining Reynolds-averaged Navier – Stokes (RANS) and Large eddy simulation (LES) methods for turbulence modeling. It targets a wide range of high performance computing (HPC) systems, from a compute server or small cluster to an exascale supercomputer. The advantages of the key components of the technology are summarized. These key components are a hybrid RANS-LES turbulence modeling method, a numerical scheme for discretization in space, a parallel algorithm, and a portable software implementation for modern hybrid systems with extra massive parallelism. Examples of our simulations are given and parallel performance on various HPC systems is presented.

Keywords: computational fluid dynamics, turbulent flows, scale-resolving simulation, hybrid RANS-LES approach, CPU+GPU, MPI+OpenMP+OpenCL.

Introduction

Hyrbid RANS-LES methods are widely recognized as the most efficient ones in terms of cost/accuracy ratio in many computational aerodynamics and aeroacoustics applications [6, 11]. Such methods combine Reynolds-averaged Navier – Stokes (RANS) and Large Eddy Simulation (LES) turbulence models. However, scale-resolving simulation of complex configurations such as an entire aircraft is still too computationally expensive for widespread use in practice. The growing computing power and the emergence of exascale supercomputers are expanding the applicability of such resource-intensive applications. The evolution of high-accuracy schemes and turbulence models is aimed at reducing requirements for mesh resolution, which leads to a significant reduction in computational costs. The development of parallel algorithms and heterogeneous software implementations ensures efficient use of modern hybrid supercomputers. The present work is devoted to the successful choice of these main components of the simulation technology: hybrid turbulence modeling approaches, high-accuracy numerical schemes, parallel algorithms and portable software implementation for hybrid supercomputers. In the following sections, a combination of these key components is proposed and the advantages of the selected state-of-the-art methods are outlined.

1. High-accuracy Schemes

To describe a turbulent flow, the Navier – Stokes equations for a viscous compressible gas are discretized in space using unstructured mixed-element meshes. The following is required from numerical schemes: high accuracy to provide sufficiently accurate solutions on much coarser meshes than are needed for low order schemes, low computational cost, low memory requirements to fit in scarce GPU memory, applicability to flows with discontinuities to simulate supersonic flows, implicit time integration to overcome the time step constraints, which make it by far unacceptably small due to the mesh step size in boundary layers. For spatial discretization, we use vertex-centered edge-based reconstruction schemes (EBR) for smooth flows [1] (subsonic)

[1]Keldysh Institute of Applied Mathematics, RAS Moscow, Russian Federation

and flows with discontinuities [2] (supersonic). Quasi one-dimensional reconstruction of variables with simple interpolation constructs significantly increases accuracy while keeping computational cost almost equal to a basic compact low-order scheme. The EBR schemes have "superpowers" on translationally-invariant meshes (such as structured Cartesian mesh zones), which allow reaching the fifth order of accuracy. On arbitrary unstructured meshes, they can compete in terms of accuracy with higher-order schemes, which are far more expensive. In the case of implicit time integration, another advantage of EBR schemes consists in using a simplified Jacobian with the same sparse matrix portrait as for a compact scheme with only direct nodal adjacency by edges. This ability to discard additional nodal couplings of rather wide interpolation constructs dramatically reduces memory consumption, which is critical on GPUs. In terms of accuracy, the effect of using EBR schemes on the numerical solution is shown in Fig. 1. It is important to note that the difference in computing cost with a low order scheme is within 15%. As a drawback, such vertex-centered schemes require careful use in terms of mesh quality, especially in the transition between structured and unstructured mesh areas. Certain stability or monotonicity problems still remain to be solved.

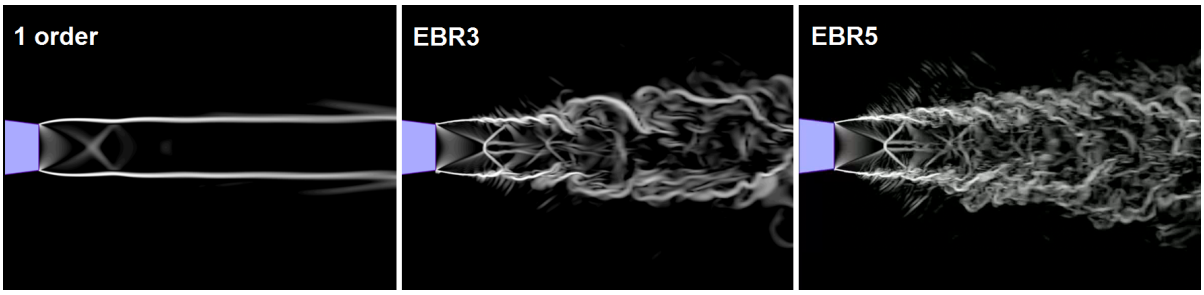


Figure 1. The effect of using EBR schemes compared to a basic first order scheme (4.5 million nodes mesh, round underexpanded jet, density gradient in the mid-span section)

2. Turbulence Modeling

For turbulence modeling, non-zonal hybrid approaches are used, which combine LES and RANS. The former needs fine enough spatial and temporal resolution to capture accurately relevant turbulent structures. The later requires much lower computational costs due to the possibility of using coarser and anisotropic meshes, since only average flow gradients need to be resolved. It is more widely, used but in many applications it can be inaccurate, especially in flows with strong separation, or inapplicable if unsteady characteristics such as noise are needed. Combining RANS in the near-wall regions and LES elsewhere allows taking advantages of both methods: RANS significantly reduces resolution requirements in wall-tangential directions in boundary layers, while LES efficiently reproduce unsteady flow features.

The following is required from modern turbulence modeling approaches: adaptive switching between different modes depending on the mesh resolution and flow features; fast transition from RANS to LES in shear layers; a minimum level of empiricism and user involvement; simplicity of parallel implementation. In accordance with the above requirements, we suggest using the following combination of methods. For faster RANS-LES transition (see Fig. 2), we use the most recent formulations of the detached eddy simulation (DES) approach, the Delayed DES (DDES) [14] and Improved DDES (IDDES) [10], but with alternative LES models and subgrid length scales.

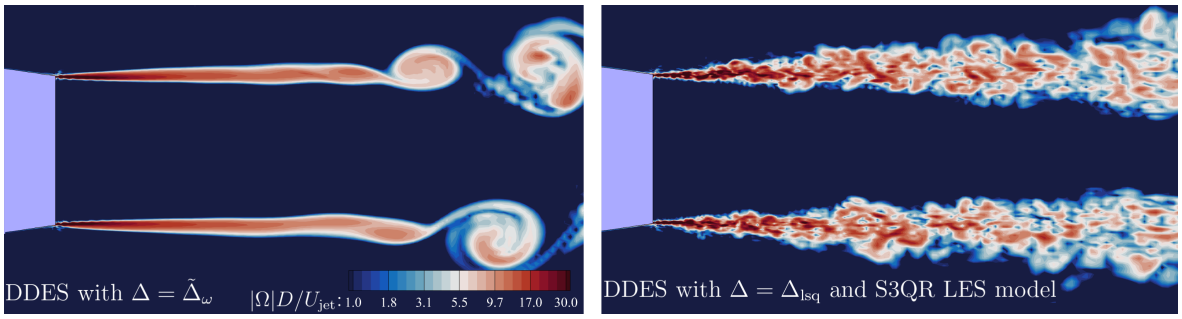


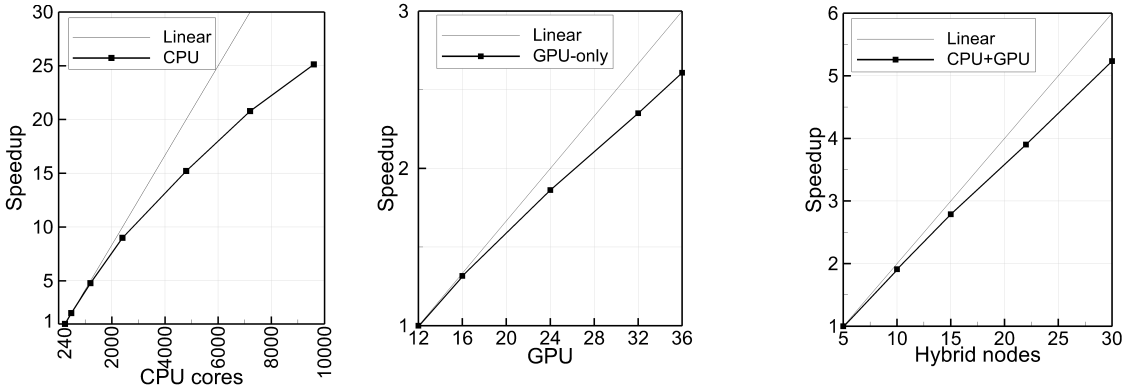
Figure 2. Faster transition to turbulence in shear layers with the Δ_{lsq} scale and S3QR LES model (9 million nodes mesh, round subsonic jet, vorticity magnitude in the mid-span section)

The latest empiricism-free adaptive subgrid length scale Δ_{lsq} [13, 16] is used because it works just as well as the Δ_{SLA} scale used in [10, 14], but it is much easier to implement for unstructured meshes, simply on the basis of a gradient operator with minor modifications. The S3PQR family models [15], used as an alternative LES model, self-adapt to the presence of walls and are sensitive to quasi-2D flow structures, in contrast to Smagorinsky model in [14].

However, despite the fact that new advanced models and length scales have been recently developed, the so-called gray area problem [12] (transition between RANS and LES) still remains to a certain extent. The main efforts are now aimed at its further mitigation.

3. Parallel Computing

From the parallel computing perspective, the following is required: no scalability limitations, distributed-memory parallelization with hiding of communication overhead, efficient shared-memory parallelization for manycore CPUs, full compatibility with stream processing on GPUs, heterogeneous co-execution on both CPU and GPU. These properties enable the use of numerous computing devices, CPUs and GPUs, and open the way to the exascale level. We use hierarchical parallelization based on multilevel mesh decomposition. The Message-passing interface (MPI) is used at the upper level to couple hybrid cluster nodes and devices inside nodes. To reduce network traffic, the mesh is decomposed first among hybrid nodes, then among computing devices, manycore CPUs and GPUs. To hide the data transfer overhead, overlapping communications and computations is used. At the lower level, OpenMP shared memory decomposition-based parallelization is used for manycore CPUs and accelerators. The mesh subdomains of CPUs are further decomposed among parallel threads of MPI processes (instead of using loop-based parallelism, which is less efficient on NUMA systems). Finally, to compute on GPUs, the OpenCL standard is used. In contrast to the NVIDIA CUDA framework, it can engage GPUs from different vendors, including NVIDIA, AMD, Intel. To use CPUs and GPUs concurrently, their mesh subdomains are balanced according to the actual performance ratio. The heterogeneous parallel algorithm is implemented in the NOISEtte code [8]. Further details on parallel algorithm, adaptation of the numerical algorithm and software implementation to GPU computing can be found in [7–9]. Examples of parallel speedups are shown in Fig. 3 for CPUs, GPUs, and CPU+GPU co-execution (numerical configuration: EBR5 scheme, implicit BDF2 scheme, hybrid RANS-LES approach). Relatively coarse meshes of up to 80 million nodes are used in tests in order to observe the degradation of the parallel efficiency of the available rather modest computational resources (on finer meshes, the parallel efficiency is too high to evaluate the limits of parallelism).



(a) Intel Xeon 8160 (b) NVIDIA V100 (c) Intel Xeon E5-2697v3 + NVIDIA K40

Figure 3. Parallel speedups on supercomputers in CPU, GPU, and CPU+GPU computing modes

The summary of performance tests: execution on 36 NVIDIA V100 GPUs is as fast as on 10,000 CPU cores, about 0.27 s per implicit time step; co-execution on CPUs and GPUs on Lomonosov-2 supercomputer (14-core CPU and NVIDIA K40 GPU per node) gives 25–30% speedup compared to GPU-only execution; high parallel efficiency is observed when payload per CPU core is above 10 thousand mesh nodes, or above 1 million nodes per GPU NVIDIA V100 (this load is more than enough to hide most of the exchanges behind computations). On meshes of few billion nodes, several hundred thousand CPU cores or several thousand GPUs can be engaged, which corresponds to computing resources of tens of PFLOPS. When performing a series of simulations for multiple variants of geometry or flow conditions, an entire exascale supercomputer can be fully occupied with high efficiency (if we live to see those bright days when such systems will be available to us).

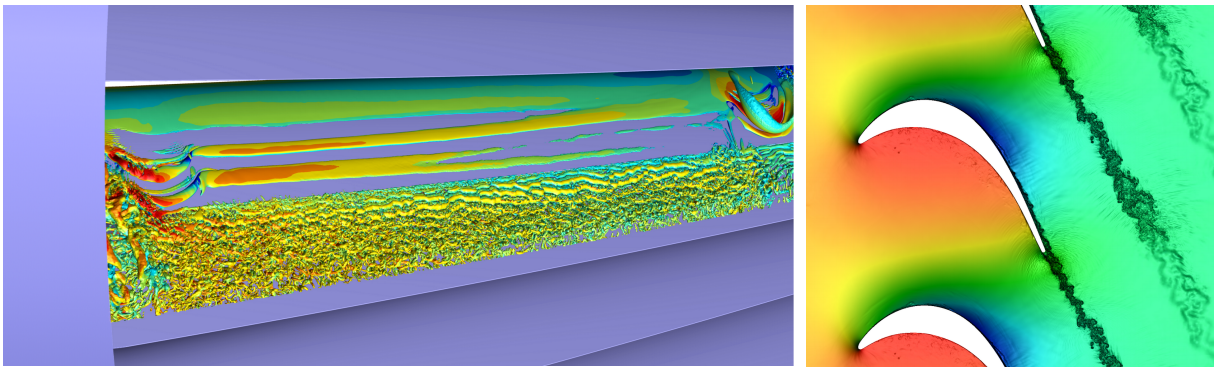
4. Applications

Below are typical examples of our scale-resolving simulations of problems in which stationary RANS methods are either inapplicable or too problematic and inaccurate.

Modeling low-pressure turbine blades of turbofan engines is shown in Fig. 4. The presence of laminar-turbulent (LT) transition and flow separation on the suction side of the blades makes this configuration very problematic for RANS methods, even using LT models, since accurate capturing of the transition location is critical for predicting integral characteristics. For instance, RANS was unable to correctly reproduce the effect of total pressure loss observed in the experiment, so scale-resolving simulations had to be performed, in which sufficiently accurate results were obtained. Details on these simulations can be found in [5].

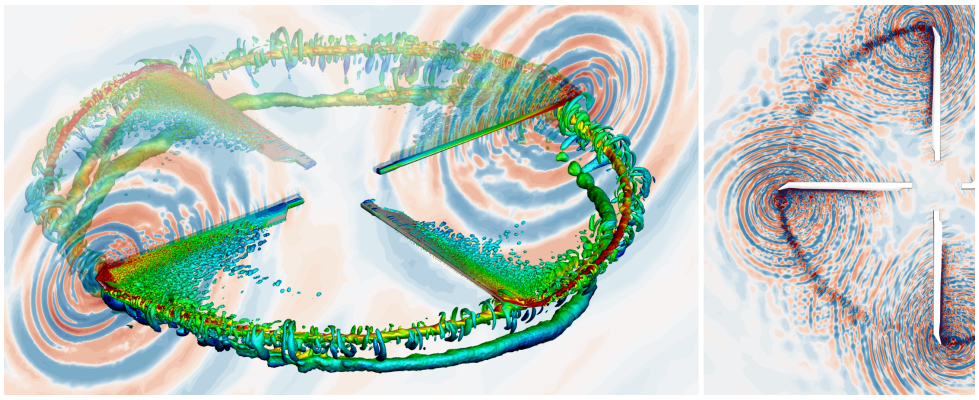
Due to the lack of computational resources, we usually consider only a section of a blade in a linear cascade with periodic conditions when performing a series of simulations. An entire blade has been simulated on meshes up to 150 million nodes so far, which is rather coarse. For accurate simulations of entire blades, meshes with more than a billion of nodes are required.

Modeling aerodynamics and aeroacoustics of helicopter rotors and drone propellers is shown in Fig. 5. In this case, RANS approaches are inapplicable for predicting broad-band aerodynamic noise. This requires high-fidelity scale-resolving simulations. Meshes of about 100 million nodes per blade are needed for resolving relevant flow structures. In our simulations, meshes of up to 400 million nodes have been used so far. More information, including validation and comparison of RANS and DES results, can be found in [3, 4].

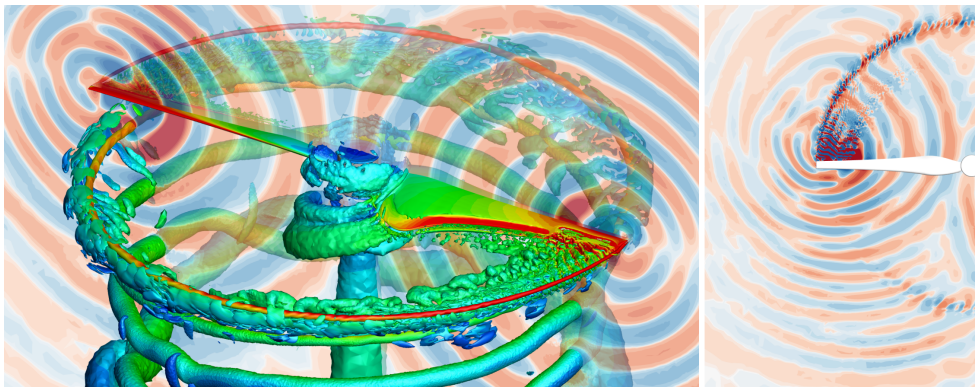


(a) Q-criterion isosurfaces[1] (b) density field in middle section

Figure 4. Flow in a low pressure turbine with LT transition present on the suction side



(a) Helicopter rotor



(b) Drone propeller

Figure 5. Simulation of rotors: turbulence (Q-criterion) and acoustics (time derivative of pressure)

Currently available supercomputer resources allow modeling only separate fragments of aircrafts. We typically use meshes of several hundred million nodes for industrial applications yet (for meshes of the order of a billion nodes, only test runs were performed to demonstrate operability and robustness). The demonstrated parallel efficiency, as well as the inherent potential of multilevel parallelism and the absence of scalability constraints, suggest that future exaflop supercomputers will allow us to use meshes dozens of times more detailed and simulate such complex configurations as an entire aircraft.

[1] $Q = 0.5(\|\Omega\|^2 - \|S\|^2)$, where Ω and S are the vorticity and strain rate tensors, respectively.

Acknowledgements

This work was supported by Moscow Center of Fundamental and Applied Mathematics, Agreement with the Ministry of Science and Higher Education of the Russian Federation, No. 075-15-2019-1623. Results in Section 3 were obtained within the RSF project 19-11-00299. The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University [17], the equipment of Shared Resource Center of KIAM RAS (http://ckp.kiam.ru). The authors thankfully acknowledge these institutions.

References

1. Abalakin I., Bakhvalov P., Kozubskaya, T.: Edge-based reconstruction schemes for unstructured tetrahedral meshes. International Journal for Numerical Methods in Fluids 81(6), 331–356 (2016), https://doi.org/10.1002/fld.4187

2. Bakhvalov, P.A., Kozubskaya, T.K.: EBR-WENO scheme for solving gas dynamics problems with discontinuities on unstructured meshes. Computers & Fluids 157, 312–324 (2017), https://doi.org/10.1016/j.compfluid.2017.09.004

3. Bobkov, V., Abalikin, I., Kozubskaya, T.: Simulation of Helicopter Rotors On Unstructured Mixed Meshes Using Edge-Based Reconstruction Schemes. WCCM-ECCOMAS2020 (2020). https://doi.org/10.23967/wccm-eccomas.2020.308

4. Bobkov, V., Gorobets, A., Kozubskaya, T., et al.: Supercomputer Simulation of Turbulent Flow Around Isolated UAV Rotor and Associated Acoustic Fields. RuSCDays 2021, CCIS 1510 (2021). https://doi.org/10.1007/978-3-030-92864-3_20

5. Duben, A.P., Kozubskaya, T.K., Marakueva, O.V., et al.: Simulation of flow over high-lifted turbine cascade at low Reynolds numbers. Journal of Physics: Conference Series 1891, 012018 (2021). https://doi.org/10.1088/1742-6596/1891/1/012018

6. Fröhlich, J., von Terzi, D.: Hybrid LES/RANS methods for the simulation of turbulent flows. Progress in Aerospace Sciences 44(5), 349–377 (2008). https://doi.org/10.1016/j.paerosci.2008.05.001

7. Gorobets, A.V., Bakhvalov, P.A., Duben, A.P., et al.: Acceleration of NOISEtte Code for Scale-Resolving Supercomputer Simulations of Turbulent Flows. Lobachevskii Journal of Mathematics 41(8), 1463–1474 (2020), https://doi.org/10.1134/S1995080220080077

8. Gorobets, A.: Parallel Algorithm of the NOISEtte Code for CFD and CAA Simulations. Lobachevskii Journal of Mathematics 39(4), 524–532 (2015), https://doi.org/10.1134/S1995080218040078

9. Gorobets, A., Bakhvalov, P.: Improving Reliability of Supercomputer CFD Codes on Unstructured Meshes. Supercomputing Frontiers and Innovations 6(4), 44–56 (2020). https://doi.org/10.14529/jsfi190403

10. Guseva, E.K., Garbaruk, A.V., Strelets, M.K.: Assessment of Delayed DES and Improved Delayed DES Combined with a Shear-Layer-Adapted Subgrid Length-Scale in Separated Flows. Flow, Turbulence and Combustion 98, 481–502 (2017), https://doi.org/10.1007/s10494-016-9769-7

11. Heinz, S.: A review of hybrid RANS-LES methods for turbulent flows: Concepts and applications. Progress in Aerospace Sciences 114, 100597 (2020). https://doi.org/10.1016/j.paerosci.2019.100597

12. Mockett, C., Haase, W., Schwamborn, D. (eds.): Go4Hybrid: Grey Area Mitigation for Hybrid RANS-LES Methods. Springer International Publishing (2018). https://doi.org/10.1007/978-3-319-52995-0

13. Pont-Vílchez, A., Duben, A., Gorobets, A., et al.: New strategies for mitigating the gray area in delayed-detached eddy simulation models. AIAA Journal pp. 1–15 (2021). https://doi.org/10.2514/1.j059666

14. Shur, M.L., Spalart, P.R., Strelets, M.K., et al.: An enhanced version of DES with rapid transition from RANS to LES in separated flows. Flow, Turbulence and Combustion 95(4), 709–737 (2015). https://doi.org/10.1007/s10494-015-9618-0

15. Trias, F.X., Folch, D., Gorobets, A., et al.: Building proper invariants for eddy-viscosity subgrid-scale models. Physics of Fluids 27, 065103 (2015), https://doi.org/10.1007/s10494-016-9769-7

16. Trias, F.X., Gorobets, A., Silvis, M.H., et al.: A new subgrid characteristic length for turbulence simulations on anisotropic grids. Physics of Fluids 29(11), 115109 (2017). https://doi.org/10.1063/1.5012546

17. Voevodin, Vl., Antonov, A., Nikitenko, D., et al.: Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community. Supercomputing Frontiers and Innovations 6(2), 4–11 (2019), https://doi.org/10.14529/jsfi190201

DOI: 10.14529/jsfi210402

Improving the Computational Efficiency of the Global SL-AV Numerical Weather Prediction Model

Mikhail A. Tolstykh[1,2], ***Rostislav Yu. Fadeev***[1,2], ***Vladimir V. Shashkin***[1,2], ***Gordey S. Goyman***[1]

The recent works on improving the efficiency of the Russian SL-AV global numerical weather prediction model both for medium- and long-range forecasts are described. The algorithmic improvements of SL-AV dynamical core, implementation of parallel I/O and several code optimizations are presented. We investigate the impact of single precision computations in some parts of the code on present climate simulations. As a result of efforts described in this article, we are now able to compute a 24-hour forecast for the model version having about 10 km horizontal resolution and 104 vertical levels in 13 min using 2916 processor cores of Cray XC40 system. This timing allows multiple experiments for tuning this new model and fits the requirements for operational weather forecast. The single long-range forecast with low-resolution SL-AV version now takes just 89 minutes instead of 111. We have also verified that the partial utilization of single precision computations produces approximately the same model climate as the previous version with fully double precision computations.

Keywords: numerical weather prediction, global atmosphere model, computational efficiency, I/O optimization.

Introduction

The common ways to improve medium-range (3–10 days) numerical weather prediction is, first, to increase the prognostic model resolution, second, to take into account the model uncertainty, and, third, to replace the atmosphere model with the coupled many-component model incorporating atmosphere, ocean sea-ice models called Earth system model. Accounting for model uncertainty is accomplished by the ensemble prediction that uses 20–100 runs of the same model incorporating some perturbations and starting from the perturbed initial conditions [12]. Using ensemble technique makes it also possible to produce a probabilistic forecast of area-averaged anomalies of weather parameters for months ahead. Both medium-range and long-range forecasting is usually done with the same global atmosphere model. As smaller scales are less predictable than larger scales, a lower-resolution model is applied for long range prediction. All the above mentioned applications require huge computer resources for timely forecast delivery. Many computer systems of the world weather forecasting centres are present in the supercomputer Top500 list [13].

A modern global atmosphere model should be able to use efficiently up to hundred of thousands of processor cores. At the same time, the new concerns about climate change require to weigh the advantages in weather prediction quality gained by the increase of resolution and/or model complexity with respect to electric power consumption [5]. All these considerations lead to increasing demands to the parallel efficiency of the atmosphere model code with different resolutions, along with its portability to different architectures.

In this paper, we describe recent works on improving the computational efficiency of the SL-AV global numerical weather prediction model both for medium-range and long-range forecasts

[1]Marchuk Institute of Numerical Mathematcis RAS, Moscow, Russian Federation
[2]Hydrometeorological Research Center, Moscow, Russian Federation

applications. This model is developed at Marchuk Institute of Numerical Mathematics Russian Academy of Sciences and Hydrometcenter of Russia. This model is applied for operational medium-range and long-range weather forecasts [17]. Algorithms and their parallel implementation using one-dimensional MPI decomposition and OpenMP loop parallelization are described in [15, 16]. The earlier code version demonstrated 53% efficiency at 9072 processor cores for 3024×1513×126 grid (without I/O) [20].

There is a new version of SL-AV model with horizontal resolution of about 10 km and 104 vertical levels (SL-AV10). Some works on its optimizations are presented in [18, 19]. The elapsed time necessary to run a 24-hour forecast had reached 32 min at 4000 processor cores, without I/O. The results achieved earlier are not sufficient for operational application of this version of the SL-AV model that requires the 24-hour forecast to be computed in less than 20 minutes using less than 3000 processor cores. Furthermore, it is very time-consuming to carry out complex tuning of all model parametrizations for subgrid-scale processes that are mostly resolution-dependent. Such a tuning requires multiple numerical experiments involving a series of forecasts for different seasons.

In this paper, recent algorithmic improvements of SL-AV dynamical core (Section 1), implementation of parallel I/O (Section 2) and some code optimizations (Section 3) are presented. In Section 4, we study the impact of the partial use of single precision computations introduced earlier [19] on present climate simulation. All these works are summarized in Conclusions.

Cray XC40 system installed at Roshydromets Main Computing Center is used in all the tests described in this article. It consists of 936 nodes with two Intel Xeon E2697v4 18-core CPUs and 128 GB memory. All the nodes are connected with Cray ARIES inter-connect. The peak performance is 1.29 PFlops. The system includes Lustre parallel file system. We use Cray Fortran compiler version 10.0.3. We have also tried Intel Fortran Compiler version 19.1.254, similar results are obtained. We also use NetCDF library version 4.7.4 in this study.

1. Algorithmic Improvements

For a long time during the development of SL-AV model, we observed a noise in the numerical solution over the mountainous regions when using large values of the time step. To alleviate this problem, we had to decrease the time step that compromised model efficiency. Actually, the resulting time step value in operational medium range weather forecast was more than two times smaller than in the ECMWF IFS model [7] that is similar to SL-AV model in many aspects. In this section, we describe the modifications that allowed to get rid of the noise and consequently increase the time step value and hence improve model efficiency.

Upon inspecting the orographic noise in the model, it turned out that it consists of approximately 100–200 km-scale stationary wave modes in geopotential height field. Initially, we attributed this behaviour to the spurious orographic resonance – the known problem of semi-implicit semi-Lagrangian atmospheric models [11]. However, the spurious modes were insensitive to the common techniques of spurious orographic resonance damping [10, 11] (e.g., time off-centering of Crank-Nicolson scheme).

It was noted that the noise amplitude is sensitive to the settings of the horizontal diffusion block. Application of this kind of fields filtering is a common practice in atmospheric modelling caused by the need to avoid enstrophy clustering near the smallest resolved scales due to nonlinear cascade [8]. The biharmonic hyper-diffusion operator with the implicit time-integration scheme is applied in SL-AV model [16].

The interesting fact is that this noise amplifies with the increase of diffusion coefficients, contrary to the behavior one could normally expect. We then come to the conclusion that the orographic noise in SL-AV model depends on the details of the diffusion implementation. To investigate this effect, the diffusion-driven stationary orographic noise model is developed. This model is based on the linearized shallow water model of [9] used to investigate properties of the spurious resonant response of a semi-implicit semi-Lagrangian model to the orographic forcing. The considerations [9] are modified to account for SL-AV specifics, the vorticity-divergence representation of the flow and horizontal diffusion implementation.

We start from 1D non-linear shallow water equations system:

$$u_t = -uu_x - gh_x - gb_x, \tag{1}$$

$$h_t = -uh_x - hu_x, \tag{2}$$

where u is the flow speed, h is the fluid layer thickness, b is bottom elevation (terrain height), g is the gravity acceleration, subscripts indicate partial derivatives in space (x) and time (t).

This system is linearised with respect to the reference state with uniform fluid thickness H, and wind speed U (u and h are now perturbations to the reference state):

$$u_t = -Uu_x - gh_x - gb_x, \tag{3}$$

$$h_t = -Uh_x - Hu_x. \tag{4}$$

We apply then the standard semi-implicit semi-Lagrangian time integration scheme based on the Crank-Nicolson method:

$$u^{n+1} = -\frac{\Delta t}{2}\left(gh_x^{n+1} + gb_x\right) + A\left(u^n - \frac{\Delta t}{2}\left(gh_x^n + gb_x\right)\right), \tag{5}$$

$$h^{n+1} = -\frac{\Delta t}{2}Hu_x^{n+1} + A\left(h^n - \frac{\Delta t}{2}Hu_x^n\right), \tag{6}$$

where A is the linearised semi-Lagrangian advection operator: $(Af)(x) = f(x - U\Delta t)$. Equations (5), (6) are linearised shallow water counterparts of the 3D equations used in SL-AV model. The SL-AV model reformulates this system in terms of vorticity and divergence (in 1D case only divergence is relevant variable). The prognostic equation for the divergence $D = u_x$ is obtained after differentiation of wind equation (5) in x:

$$D^{n+1} = -\frac{\Delta t}{2}\left(gh_x^{n+1} + gb_x\right)_x + \left[A\left(u^n - \frac{\Delta t}{2}\left(gh_x^n + gb_x\right)\right)\right]_x. \tag{7}$$

The height equation is also formulated using divergence, i.e. D is substituted for u_x type terms.

Consider system (6), (7) for one Fourier harmonic e^{ikx}: $(D, h)^T = (\hat{D}, \hat{h})^T e^{ikx}$ and $b = \hat{b}e^{ikx}$. The resulting equations are:

$$\hat{D}^{n+1} = -\frac{\Delta t}{2}(ik)^2\left(g\hat{h}^{n+1} + g\hat{b}\right) + e^{-ikU\Delta t}\left(\hat{D}^n - (ik)^2\frac{\Delta t}{2}\left(g\hat{h}^n + g\hat{b}\right)\right), \tag{8}$$

$$\hat{h}^{n+1} = -\frac{\Delta t}{2}H\hat{D}^{n+1} + e^{-ikU\Delta t}\left(\hat{h}^n - \frac{\Delta t}{2}H\hat{D}^n\right), \tag{9}$$

where $e^{-ikU\Delta t}$ is the Fourier image of semi-Lagrangian advection operator A.

The SL-AV solution procedure for system (8, 9) is as follows. First, $\hat{h}^{n+1}$ is excluded from (8) using (9) and the Helmholtz problem is solved for the divergence:

$$\hat{D}^{n+1} + k^2 \frac{\Delta t^2}{4} gH\hat{D}^{n+1} = \hat{R}_{helm}, \tag{10}$$

where $\hat{R}_{helm}$ is the combination of known time step n terms.

Second, artificial biharmonic diffusion (small scale filtering) is applied to the divergence:

$$\hat{D}^{n+1*} = \hat{D}^{n+1}/(1 + \Delta t C k^4), \tag{11}$$

where C is the diffusion coefficient. After divergence filtering, $\hat{h}^{n+1}$ is calculated from equation (9) by substitution of $\hat{D}^{n+1*}$ for D^{n+1}. After all, h^{n+1} is filtered using the same scheme as for divergence (11) (in the 3D model h is not filtered, the temperature that is closely related to h by hydrostatic equation is filtered instead).

From shallow-water considerations, it may seem that filtering divergence at the same time with h after solution of system (8, 9) will be more consistent. However, this early divergence filtering makes sense in 3D model because it implies filtering of updates for some derived fields like surface pressure and vertical velocity which will be left unfiltered otherwise (see Section 4 of [16]).

The calculation of $\hat{D}^{n+1}$, $\hat{h}^{n+1}$ using the procedure described above can be summarized by the following equation:

$$\begin{pmatrix} \hat{D} \\ \hat{h} \end{pmatrix}^{n+1} = Q \begin{pmatrix} \hat{D} \\ \hat{h} \end{pmatrix}^{n+1} + R \begin{pmatrix} \hat{b} \\ 0 \end{pmatrix}, \tag{12}$$

where Q and R are the matrices with complex entries describing the solution procedure. We are interested in stationary orography-forced solutions of this system and their dependence on the wave number k and diffusion coefficients. Stationary solutions can be found with setting $\hat{D}^{n+1} = \hat{D}^n = \hat{D}_s$, $\hat{h}^{n+1} = \hat{h}^n = \hat{h}_s$ in equation (12):

$$\begin{pmatrix} \hat{D}_s \\ \hat{h}_s \end{pmatrix} = (I - Q)^{-1} R \begin{pmatrix} \hat{b} \\ 0 \end{pmatrix}. \tag{13}$$

We investigate stationary orographic response properties for three different options of diffusion application. The first is original SL-AV diffusion where the divergence is filtered right after the Helmholtz problem solution and h field is not filtered at all (the diffusion coefficient for the temperature in SL-AV is significantly smaller than for divergence). This option will be referred to as 'reference'. With the second option, the divergence is filtered after the calculation of h^{n+1}, h itself is not diffused (this will be referred to as 'semi-consistent'). The third option is diffusion application for both D^{n+1}, h^{n+1} after their calculation with the same coefficient ('consistent' option).

Figure 1 shows the amplitude of height field response to orographic forcing obtained numerically using equation (13). The following non-dimensional parameters are used: $U = 0.3$, $H = 1$, $g = 1$, $\Delta t = 0.1$, $\hat{b} = 1$, $C = 10^{-7}$. The black curve in the figure shows the exact response amplitude, independent of the wavenumber k. The most spectacular feature in Fig. 1 is the blue curve showing perfect spurious orographic resonance in the absence of any dissipation mechanisms. The response amplitude reaches infinite values at about wave number $k = 29\pi$ that corresponds to the shortest scale motions in the real model. However, this curve is not relevant for the

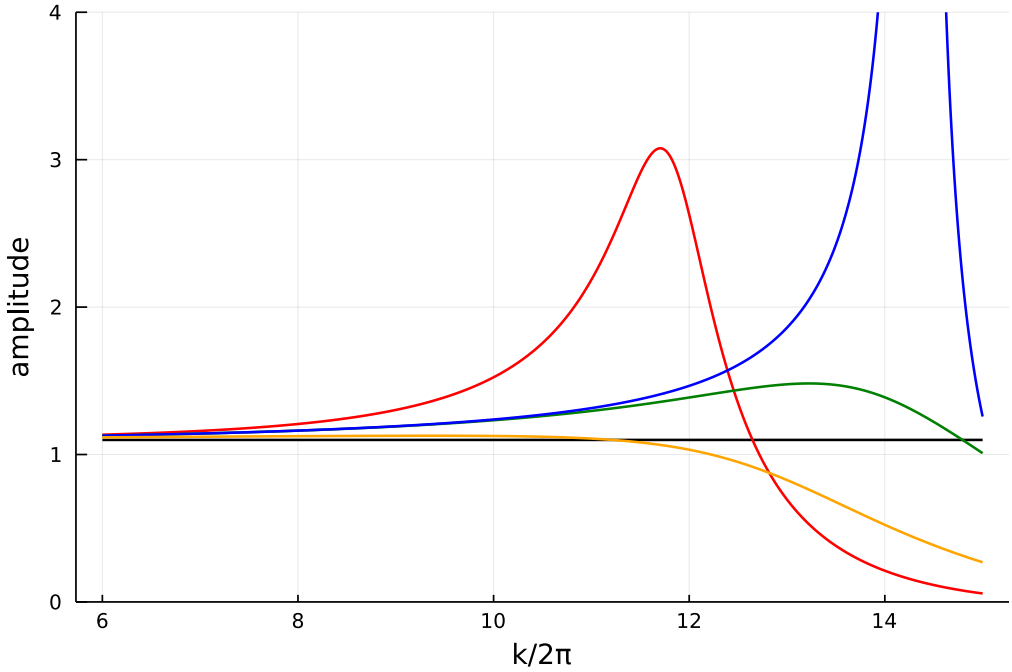


Figure 1. The amplitude of height field response to orographic forcing. Red curve – 'reference' scheme of diffusion application, green – 'semi-consistent' diffusion, blue – no diffusion, orange – 'consistent' diffusion, black – analytic response amplitude

real SL-AV model because both explicit (diffusion, Cranck-Nicolson off-centering) and implicit (advection scheme damping) numerical dissipation will prevent the model from instability.

The red curve shows the response of the 'reference' scheme. This curve shows weak resonance with wavelengths that are short enough, but still far from the shortest scales resolved on the grid. We believe that exactly this scenario takes place in the model simulations. The reason for the amplitude growth is the inconsistent application of diffusion breaking the balance between D and h that makes orographic mode stationary and this is compensated by the exaggerated growth of h amplitude. At the shortest scales, the resonance is effectively eliminated.

The green and orange curves in Fig. 1 show the response amplitude for 'semi-consistent' and 'consistent' schemes. The response of 'consistent' scheme is very close to the exact curve for large and intermediate wavelengths, orographic waves are dumped out at the shortest scales. The 'semi-consistent scheme' shows very weak amplification of response for intermediate and short wave lengths.

Linearised shallow-water study, therefore, indicates that the 'reference' diffusion scheme can spuriously amplify the orographic response at the scales well-resolved on the grid. 'Semi-consistent' diffusion leads to much more accurate solutions. The best result is achieved with 'consistent' diffusion, suggesting that using the same diffusion coefficients for all fields might be favourable.

The difference between diffusion application schemes can be noticed in the non-linear 3D SL-AV model simulations as well as in the shallow-water model. The typical picture is given in Fig. 2 that compares two SL-AV20 (approximately 24 km horizontal resolution) six-hour forecasts of 500 hPa geopotential height field. Both forecasts are computed using the time step value $\Delta t = 540\,s$ that is 2.25 times greater than the operational time step. The forecast using 'reference' diffusion depicted in Fig. 2b suffers from evident orographic noise over North-Eurasia.

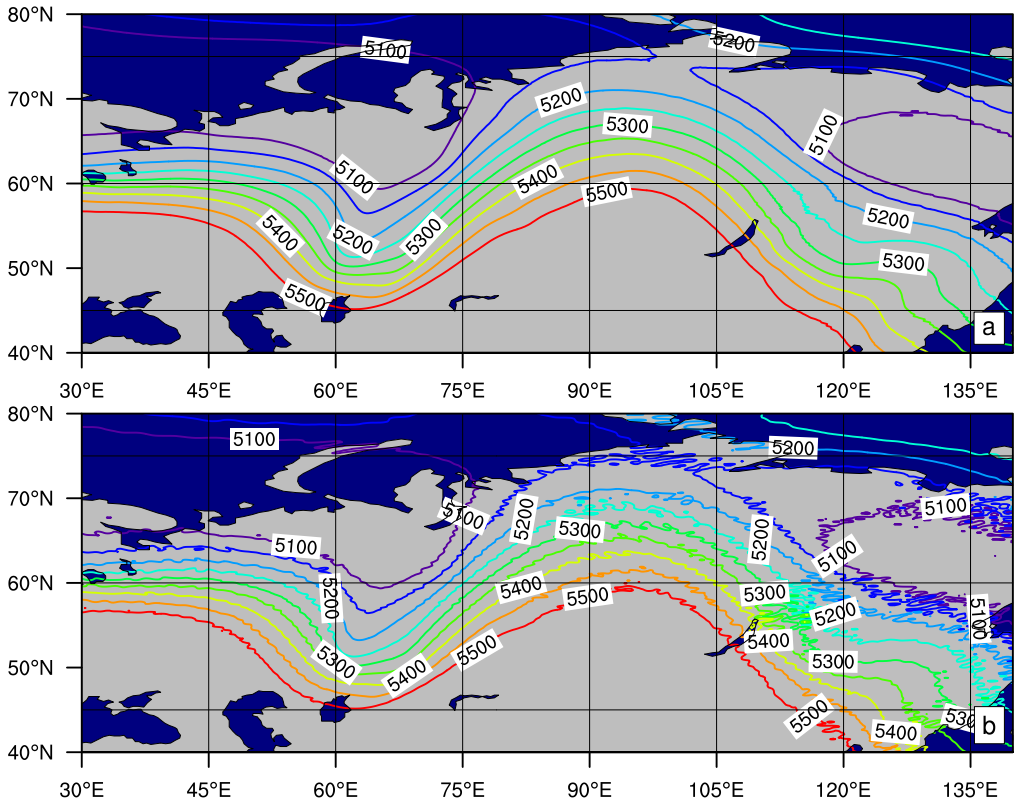


Figure 2. 6-hour forecast of 500 hPa geopotential height field using SL-AV20 model configuration with a) 'semi-consistent diffusion', b) 'reference diffusion'

At the same time, the run with the 'semi-consistent' diffusion (Fig. 2a) is free of this deficiency. The run with the 'consistent' diffusion scheme (not shown) is very similar to the 'semi-consistent' one. That means there is no strong evidence for using the same diffusion coefficients for all fields in 3D model. The implementation of the 'semi-consistent' horizontal diffusion has allowed to increase the time-step size of the SL-AV model by a factor of 2.25 with respect to the one previously used. The elapsed time of the 24-hour forecast (without I/O) has reduced by the same factor.

2. Code Optimizations

Historically, SL-AV model used a single array for the state vector of the model $\vec{\varphi} = (u, v, T, q, D, \xi, \ln p_s)^T$ containing zonal and meridional components of wind field, temperature, specific humidity, horizontal divergence, relative vorticity, logarithm of the surface pressure, respectively. This array had the indices arrangement as $(Nlon, 6Nlev + 5, Nj)$ in earlier versions of the model, where $Nlon$ – is the number of grid points along longitudinal direction, $Nlev$ – is the number of vertical levels and Nj – is the number of grid points along latitudinal direction for a given MPI-process (the second dimension number is explained by the fact that the surface pressure is a two-dimensional variable stored along with its derivatives). Such a data storage organization was convenient in terms of computations parallelization (MPI and OpenMP parallelization along the latitudinal direction was used) since it allowed to perform MPI exchanges for all state vector fields at a time without using buffer arrays. A demand to use more processor

cores required to switch to use OpenMP loop parallelization along the other dimension (longitude or Fourier space wave numbers). Taking into account the large volume of the model source code, the simplest and most efficient way to organize such a transition required changing the structure of the state vector array to $(6Nlev + 5, Nlon, Nj)$. This allowed to increase the maximum theoretical number of cores used from 865 to more than 10000 for the version of the SL-AV model with the horizontal resolution of about 24 km [15]. However, new indices arrangement is not optimal in terms of memory access, since it spoils the localization in memory of a given grid-point field, especially in SL-AV10 model with about 10 km horizontal resolution. Indeed, previously $Nlon \cdot Nlev$ values laid sequentially in memory, while now it is only $Nlev$ values. This led to a slowdown in the execution time of individual parts of the code, which at that time was not very significant and was an acceptable price to pay for increasing the model scalability.

In the last few years, a number of works were carried out [18–20] that allowed to significantly speed up the subgrid scale parametrizations and semi-Lagrangian advection blocks, being the most time consuming parts of the model. The recent profiling of the model showed that the slowdown associated with the use of a non-optimal state vector storage structure can no longer be considered insignificant. Thus, we decided to replace the above mentioned state vector with the individual arrays having dimensions $(Nlev, Nlon, Nj)$ for each grid-point field in the state vector.

The implementation of these changes results in a 16–22% speed up of the time step elapsed time for the model with the grid dimensions $400 \times 251 \times 96$. In particular, the time needed to compute a single long-range forecast (in fact, single ensemble member) has decreased from 111 to 89 minutes. The effect of these optimizations is even more significant for the version of the SL-AV model with the horizontal resolution of about 10 km and 104 vertical levels ($3600 \times 1946 \times 104$ grid dimension). Experiments using 2916 processor cores (81 nodes with 6 MPI-processes and 6 OpenMP threads) show a decrease in execution time of a model time step without I/O by about 30%, which leads to a 7-minute reduction of a runtime needed to deliver a single 24-hour forecast.

3. I/O Optimizations

The typical horizontal resolution of a modern global atmosphere model (7–10 km) having the problem size of order 10^9 requires high I/O efficiency as the typical size of the initial data file is about some tens of gigabytes. Indeed, from 10 to 12 3D variables and about 20 2D variables need to be stored in this file. 3D variables in a modern atmosphere model include wind speed components, temperature, specific humidity, 4–5 hydrometeors (i.e., rain droplets and ice particles concentrations), ozone concentration, turbulent kinetic energy. Then the output forecast information with a size of 3 GB needs to be stored every 1–3 hours depending on forecast lead-time. This information usually consists of five 3D fields defined at isobaric surfaces (geopotential, temperature, wind speed components, relative humidity) and 2D fields (precipitation, near surface temperature, wind components, relative humidity, snow depth, etc).

It is known since long ago that the implementation of parallel input-output of data in an atmospheric model can significantly accelerate its execution. Gradual establishment of MPI-IO moved the focus towards interfaces convenient for atmosphere and Earth system models. So the incorporation of parallel capabilities based on MPI-IO into NetCDF freeware library commonly used in Earth system models and its model components [3] was natural. NetCDF file contains

meta-data information making it portable and searchable. This format is supported by many software packages used to manipulate, analyse and plot.

Historically, the GRIB (and GRIB2) formats [1] are generally accepted in the numerical weather prediction community, contrary to the NetCDF format widely used in climate modelling [6]. GRIB2 format allows to significantly compress data thus reducing the file size. Typically, a file in GRIB2 format is 2–3 times smaller than the file with the same single-precision information written in NetCDF4 format without compression. Unfortunately, the compression algorithm used in GRIB is essentially sequential. Recent NetCDF libraries (starting from version 4.7.4) include parallel compression [2, 3], still the size of NetCDF file obtained with parallel compression is significantly larger than GRIB2 file size.

There are advanced parallel I/O systems based on NetCDF format applied for many Earth system models coupling many component models (atmosphere, ocean, sea ice, etc.) [4, 21].

We have earlier implemented parallel I/O in the SL-AV model using parallel NetCDF standard library routines [14]. We have now implemented an improved version of the parallel I/O in the model but also in all technological accompanying code (preprocessing, postprocessing of the output fields). The parallel I/O is tested for the new SL-AV medium-range forecast version having horizontal resolution of about 10 km and 104 vertical levels.

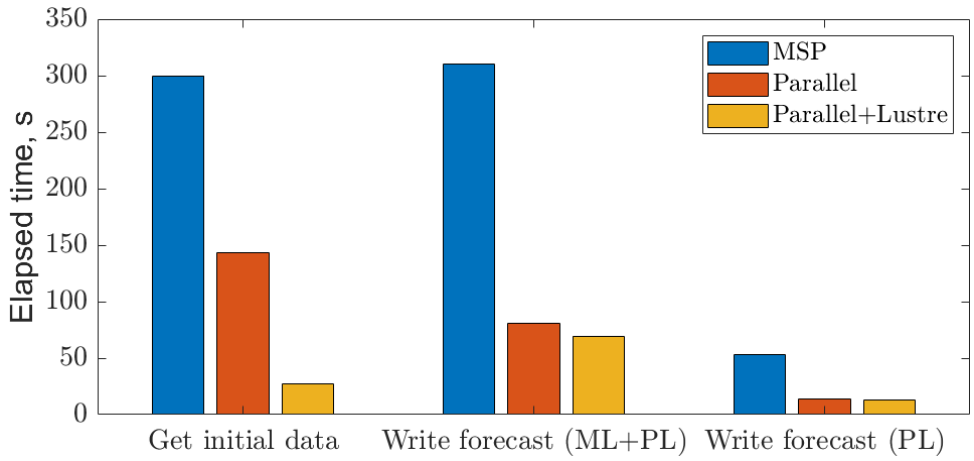


Figure 3. Elapsed time in seconds for different I/O steps of SL-AV model code while using 2916 cores at Cray XC40. ML means writing the information at model levels; the content mostly coincides with the initial data file. PL means writing prognostic information at pressure levels. MSP means sequential I/O at the master process with gather/scatter data from/to all other processes

The elapsed time for different I/O tasks in the SL-AV model using sequential and parallel I/O is shown in Fig. 3. The results of using Lustre file system capabilities for accelerating parallel I/O are also shown there.

One can see that using parallel I/O significantly accelerates this part of the code. Further acceleration is achieved while using Lustre file system options. For example, the procedure of reading the file with initial data is accelerated by a factor of 2.1 for parallel I/O alone and by factor of 10 if this file is physically located at different hard disk drives as set by `lfs setstripe` command.

Now the breakdown between different I/O components is as follows. Reading initial data at the beginning of the forecasts takes approximately 30 seconds. 70 seconds is required to write a file similar to the file with initial data which is used as a first guess file at the next forecast

cycle (6-hour forecast), and just 14 seconds is needed to write postprocessed data at pressure levels for the forecasts at all other lead times. This can be compared with the elapsed time of the usual model time step that takes 2.3 s for the time steps without radiation computations and 4.9 s for the time step including radiation calculations (this is every eighth step). Given that the output of forecast fields at pressure levels is required every three hours, the I/O elapsed time per forecast day is reduced from 440 sec to 120 sec for all forecast days other than the first one. The similar numbers for the first forecast day are 715 and 182 sec, respectively.

4. Evaluating Single Precision Computations for Climate Simulation

Earlier, the single precision calculations were introduced in some parts of the SL-AV model, namely, in the parts solving elliptic equations on the sphere, semi-Lagrangian advection and respective parallel data exchanges [19]. These parts of the SL-AV model are time consuming and include intensive parallel communications. We have found the impact of reduced accuracy on medium-range forecasts to be negligible [18], however, the impact of these changes on the model climate has not yet been investigated. We address this issue in this Section.

Details on implementation of single precision in the SL-AV program complex are presented in [19], so, we only briefly outline the main points here. The algorithm for solving elliptic equations requires a global data transposition (parallel communication of the all-to-all type), before and after the execution of this part of the code. To perform this communication, buffer arrays with compile-time defined data type (single or double precision) are used. That is, when using single precision, typecasting occurs during copying data to and from the buffer array within parallel communication phase. In the semi-Lagrangian advection block, the values of the grid-point fields to be interpolated to the departure points of the particles trajectories are stored at the single array. This array is directly used when performing parallel halo exchanges in this block, and its data type also can be switched to single precision. This allows to halve the size of data to be sent.

The following versions of the SL-AV model are considered in this Section: the model version with the reduced accuracy in the above mentioned parts, and a reference version of SL-AV with double precision computations in respective parts. Experiments are also performed for the 'intermediate' versions of the SL-AV model, where single precision is used either in the data transposition procedure (hereafter RATRAN experiment) or in the semi-Lagrangian scheme (RASL experiment). The horizontal resolution is 0.72 by 0.9 degrees in latitude and longitude respectively, the model has 96 vertical levels. The SL-AV model with this resolution is integrated for 5 years of model time, using I/O setup typical for long-range forecasts. It is worth to note that the new version of the model completes a year integration in 146 minutes as compared to 167 minutes for the version with double precision computations.

We define here the relative deviation of function f_2 from function f_1 as the l_2 norm $f_2 - f_1$ divided by the l_2 norm of f_1. Figure 4 illustrates the relative deviation evolution for some model characteristics for the new version of the SL-AV model with reduced accuracy with respect to its reference version with double precision. The following fields are shown in this figure: the zonal wind at 250 hPa, precipitation, zonal wind, temperature and geopotential at 850 hPa. The averaging was performed over a period of integration time. It can be seen that the relative deviation decreases with time for all of the variables (including those not presented in the figure).

The maximum values of the relative deviation are reached on variables with large gradients: the zonal wind and precipitation. Large gradients in these fields are achieved at the boundaries of jets (for the zonal wind) and mountains (for precipitation). It should be noted that the relative deviation of the zonal wind at 850 hPa is much larger than at 250 hPa, where much stronger wind values are achieved. This means that the sensitivity of the atmospheric circulation in the SL-AV model with respect to the accuracy of the calculations decreases with height.

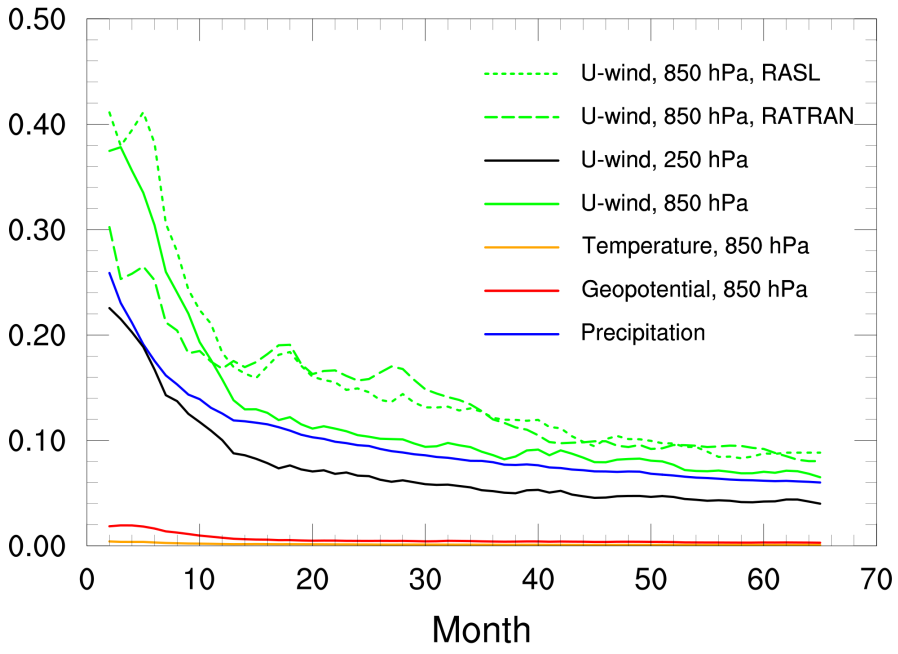


Figure 4. Relative deviation as a function of time (months) of the new version of the SL-AV model with reduced accuracy with respect to its reference version with double precision

The maximum value of the relative deviation of the 5-year averaged fields does not exceed 10%. Figure 5a illustrates the 5-year averaged zonal wind in the experiment based on the new version of the SL-AV model with reduced accuracy. The deviation of this variable from that obtained using the reference version of the SL-AV model with double precision is shown in Fig. 5b. It can be seen that the maximum value of the relative deviation of the averaged fields does not exceed 10% and is achieved due to a small shift of the jets. However, the magnitude of the shift is not large and therefore it does not significantly affect most of the forecast fields.

The dashed and dotted green curves in Fig. 4 correspond to the relative deviation of the time-averaged zonal wind at 850 hPa obtained in experiments performed using a version of the SL-AV model with a partial transition to the reduced accuracy. The dotted curve here corresponds to the application of single precision in the transposition procedure (RATRAN experiment), and the dashed curve corresponds to single precision in the semi-Lagrangian scheme (RASL experiment). It can be seen that the relative deviation of individual modifications compared to the reference version of the SL-AV model is larger. The cumulative effect of introducing single-precision calculations leads to a reduction in the relative deviation.

One can conclude that the introduction of the single precision calculations in the above mentioned parts of the model does not affect the model climate to a significant extent.

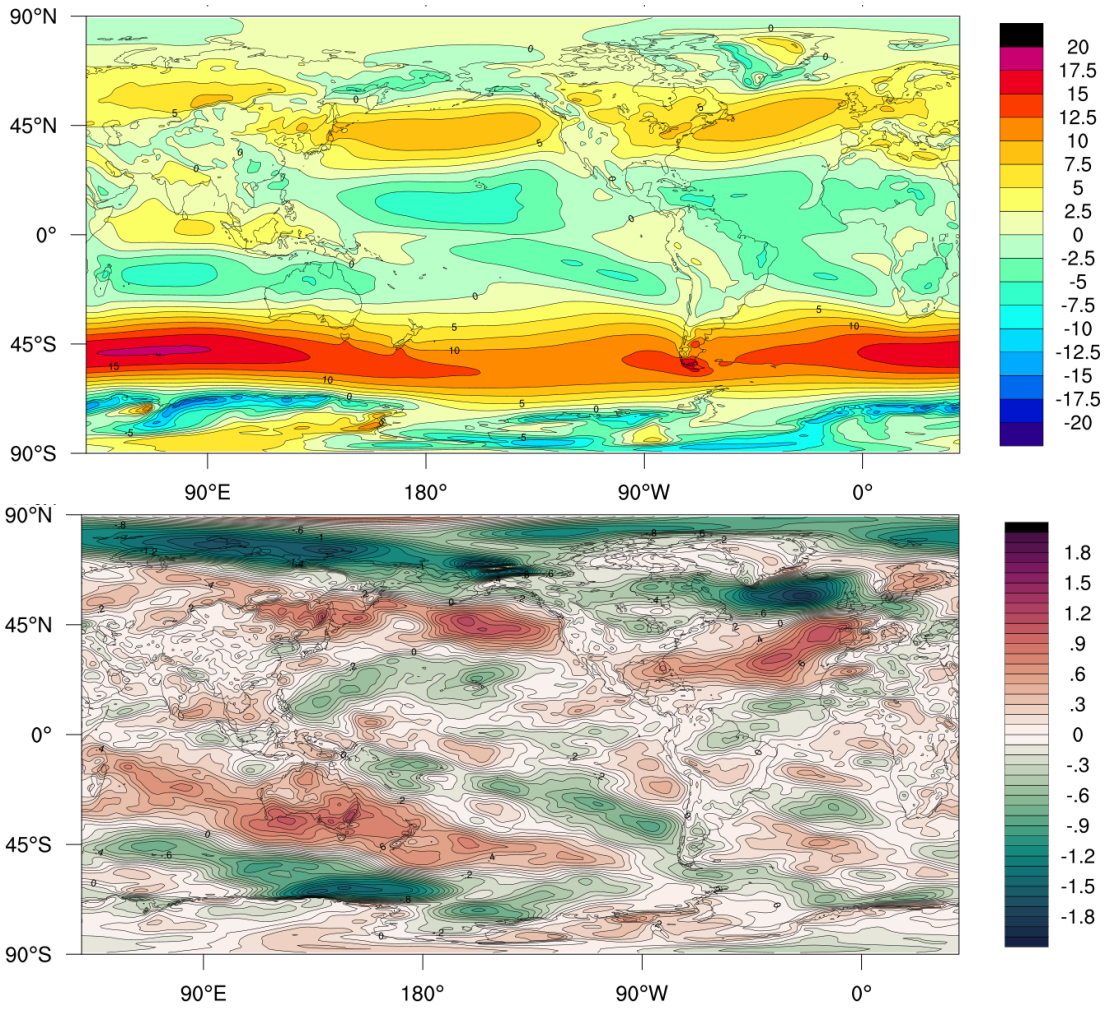


Figure 5. The 5-year time-averaged zonal wind at 850 hPa (top) and its absolute deviation in the version of the SL-AV model with the reduced accuracy compared to its reference version (bottom)

Conclusions

Initially, the elapsed time necessary to compute the 24-hour forecast with the SL-AV10 model with the horizontal resolution of about 10 km and 104 vertical levels using about 4000 processor was 42 minutes without time for I/O. The efforts undertaken in 2020 reduced this time to 32 min [19]. As mentioned above, this was still too much as the operational requirements impose a limit of no more than 20 minutes. The number of processor cores had to be reduced to less than 3000. As a result of the efforts described in this article, we are now able to compute the 24-hour SL-AV10 forecast in 13 min using 2916 processor cores. This timing allows multiple experiments for tuning this new model and fits operational requirements. Also important is the fact that the single long-range forecast (i.e., one ensemble member) with low-resolution SL-AV version now takes just 89 min instead of 111 min.

We have investigated the impact of the partial use of single precision computations on present climate simulations. It turns out that these model changes do not affect the model climate significantly.

The results described in this paper allow us to extend and accelerate the work on further model improvements.

Acknowledgements

The authors are grateful to Vassily Mizyak, Radomir Zaripov, Svetlana Travova, Vladimir Rogutov from Hydrometcenter of Russia who participate in model development and research. The numerical experiments with the SL-AV model are carried out using Cray XC40 installed at the Roshydromet Main Computer Center (MCC). Most of study (except for Section 4) was performed at Hydrometcenter of Russia and supported by the Russian Science Foundation, project 21-17-00254, https://rscf.ru/project/21-17-00254/.

References

1. Manual on Codes - International Codes, Volume I.2, Annex II to the WMO Technical Regulations: Part B Binary Codes, Part C Common Features to Binary and Alphanumeric Codes WMO-TD 611. Geneva, Switzerland (1994)

2. Learning HDF5. `https://portal.hdfgroup.org/display/HDF5/Learning+HDF5` (2021), accessed: 2021-10-21

3. Network common data form (NetCDF). `https://www.unidata.ucar.edu/software/netcdf/` (2021), accessed: 2021-10-21

4. Xios. XML-IO-Server. `https://portal.enes.org/models/software-tools/xios` (2021), accessed: 2021-10-21

5. Bauer, P., Stevens, B., Hazeleger, W.: A digital twin of Earth for the green transition. Nature Climate Change 11, 80–83 (2021). `https://doi.org/10.1038/s41558-021-00986-y`

6. Dennis, J.M., Edwards, J., Loy, R., et al.: An application-level parallel I/O library for Earth system models. Int. J. High Perf. Comput. Appl. 26, 43–53 (2012). `https://doi.org/10.1177/1094342011428143`

7. Hortal, M.: Aspects of the numerics of the ECMWF model. In: Procs. of the ECWMF Seminar, September 7-11, 1998. Reading, UK (1999)

8. Jablonowski, C., Williamson, D.L.: The Pros and Cons of Diffusion, Filters and Fixers in Atmospheric General Circulation Models, chap. 13, pp. 381–493. Springer (2011). `https://doi.org/10.1007/978-3-642-11640-7_13`

9. Lindberg, K., Alexeev, V.A.: A study of the spurious orographic resonance in semi-implicit semi-Lagrangian models. Monthly Weather Review 128(6), 1982–1989 (2000). `https://doi.org/10.1175/1520-0493(2000)128<1982:ASOTSO>2.0.CO;2`

10. Ritchie, H., Tanguay, M.: A comparison of spatially averaged Eulerian and semi-Lagrangian treatments of mountains. Mon. Wea. Rev. 124, 167–181 (1996). `https://doi.org/10.1175/1520-0493(1996)124$<$0167:ACOSAE$>$2.0.CO;2`

11. Rivest, C., Staniforth, A., Robert, A.: Spurious resonant response of semi-Lagrangian discretizations to orographic forcing: Diagnosis and solution. Monthly Weather Review 122(2), 366–376 (1994). https://doi.org/10.1175/1520-0493(1994)122<0366:SRROSL>2.0.CO;2

12. Slingo, J., Palmer, T.: Uncertainty in weather and climate prediction. Phil. Trans. R. Soc. A 369, 4751–4767 (2011). https://doi.org/10.1098/rsta.2011.0161

13. Strohmaier, E., Dongarra, J., Simon, H., Meuer, M.: Top500 list. https://top500.org/lists/top500/ (2021), accessed: 2021-10-21

14. Tolstykh, M., Fadeev, R., Mizyak, V.: Parallel program complex for numerical weather prediction and climate modeling. In: CEUR Worskshop Proceedings. RuSCDays 2015 - Proceedings of the 1st Russian Conference on Supercomputing Days 2015, Moscow, Russia, September 28-29, 2015. vol. 1482, pp. 356–367 (2015)

15. Tolstykh, M., Goyman, G., Fadeev, R., Shashkin, V.: Structure and algorithms of SL-AV atmosphere model parallel program complex. Lobachevskii Journal of Mathematics 39(4), 587–595 (2018). https://doi.org/10.1134/S1995080218040145

16. Tolstykh, M., Shashkin, V., Fadeev, R., Goyman, G.: Vorticity-divergence semi-Lagrangian global atmospheric model SL-AV20: dynamical core. Geoscientific Model Development 10(5), 1961–1983 (2017). https://doi.org/10.5194/gmd-10-1961-2017

17. Tolstykh, M., Fadeev, R., Shashkin, V., et al.: Multiscale global atmosphere model SL-AV: the results of medium-range weather forecasts. Russ. Meteorol. Hydrol. 43, 773–779 (2018). https://doi.org/10.3103/S1068373918110080

18. Tolstykh, M., Goyman, G., Fadeev, R., et al.: Development of the global multiscale atmosphere model: computational aspects. In: Journal of Physics: Conference Series. vol. 1740, p. 012074. IOP Publishing (2021). https://doi.org/10.1088/1742-6596/1740/1/012074

19. Tolstykh, M.A., Goyman, G., Fadeev, R., Shashkin, V.V.: Implementation of SL-AV global atmosphere model with 10 km horizontal resolution. In: Supercomputing - 6th Russian Supercomputing Days, RuSCDays 2020, Moscow, Russia, September 21-22, 2020, Revised Selected Papers. Communications in Computer and Information Science, vol. 1331, pp. 216–225. Springer (2020). https://doi.org/10.1007/978-3-030-64616-5_19

20. Tolstykh, M.A., Goyman, G., Fadeev, R., et al.: SL-AV model: Numerical weather prediction at extra-massively parallel supercomputer. In: Supercomputing - 4th Russian Supercomputing Days, RuSCDays 2018, Moscow, Russia, September 24-25, 2018, Revised Selected Papers. Communications in Computer and Information Science, vol. 965, pp. 379–387. Springer (2018). https://doi.org/10.1007/978-3-030-05807-4_32

21. Yang, R., Ward, M., Evans, B.: Parallel I/O in Flexible Modelling System (FMS) and Modular Ocean Model 5 (MOM5). Geosci. Model Dev. 13, 1885–1902 (2020). https://doi.org/10.5194/gmd-13-1885-2020

DOI: 10.14529/jsfi210403

The Influence of Autumn Eurasian Snow Cover on the Atmospheric Dynamics Anomalies during the Next Winter in INMCM5 Model Data

Maria A. Tarasevich[1,2], *Evgeny M. Volodin*[2]

The influence of autumn Eurasian snow cover on the atmospheric dynamics anomalies during the following winter is studied based on the INM RAS climate model data. The North Atlantic Oscillation is the leading pattern that causes the weather and climate variability in the Northern hemisphere. We evaluate the up-to-date model version (INMCM5) ability of the autumn Eurasian snow – winter NAO teleconnection simulation on different timescales. Maximum covariance analysis (MCA) is used to find winter atmospheric signals that are significantly correlated with autumn snow cover anomalies. Using MCA we conclude that Autumn Eurasian snow – winter NAO teleconnection is present in INMCM5 experiments on pre-industrial and present-day climate simulation. However, this method fails to show this phenomenon in experiments on a seasonal timescale. We conduct additional experiments on a seasonal timescale to assess the sensitivity of North Atlantic Oscillation index predictability to initial snow cover perturbations. These experiments demonstrate the absence of direct autumn Eurasian snow impact on the NAO index.

Keywords: climate model, seasonal hindcasts, North Atlantic Oscillation, Eurasian snow cover, teleconnection.

Introduction

A leading pattern affecting winter weather and climate variability over Northern hemisphere is the North Atlantic Oscillation (NAO) [17]. The NAO is defined as the fluctuation of the pressure gradient between the Stykkishólmur (Icelandic Low) and the Ponta Delgada (Azores High). The North Atlantic Oscillation represents the redistribution of atmospheric mass between the Arctic and the subtropical Atlantic. So the switch of the NAO phases is accompanied by large changes in surface air temperature, winds, storminess, and precipitation over the Atlantic as well as the adjacent continents. Thus the North Atlantic Oscillation typifies wintertime weather in Northern hemisphere extratropics. That is why the NAO phase prediction on seasonal to decadal timescales is an active goal for climate science centres [8, 24, 25, 36].

Several observations-based studies [4, 5, 23] suggest the Eurasian snow cover in autumn as a source of predictability of the North Atlantic Oscillation in winter. The dynamical mechanism linking autumn Eurasian snow cover to the following wintertime climate is described in [4].

Modelling experiments forced with prescribed observations-based Eurasian snow cover anomalies also reproduce this teleconnection [10, 11, 13]. However, even CMIP5, as well as CMIP3 Earth system models, cannot recover the autumn Eurasian snow – winter NAO relationship with internally-generated snow cover [12, 14].

In this paper we evaluate the INM RAS climate model's ability to simulate the autumn Eurasian snow cover – winter North Atlantic Oscillation teleconnection on different time scales. We also study the response of the model to initial snow cover perturbations.

The organization of this paper is as follows. Section 1 provides an overview of the INM RAS climate model and the numerical experiments' design. Section 2 describes the calculation of the North Atlantic Oscillation index and the simulated data processing methods. Section 3 discusses

[1]Moscow Institute of Physics and Technology, Dolgoprudny, Russian Federation
[2]Marchuk Institute of Numerical Mathematics of the Russian Academy of Sciences, Moscow, Russian Federation

the results of simulating the teleconnection between Eurasian autumn snow and the following winter NAO phase. Finally, the conclusions summarize the results.

1. Model and Data

For all simulations we use the climate model developed in the INM RAS. The model is coupled, i. e., it consists of two global circulation ones: the atmosphere and the ocean models. The atmosphere model performs the solution of the hydrothermodynamic equations with hydrostatic approximation in advective form. The ocean model represents large scale hydrothermodynamic equations with hydrostatic and Boussinesq approximations.

In the study we use the up-to-date version of the INM RAS climate model called INMCM5 [30]. The spatial resolution of its global atmosphere circulation model is $2^\circ \times 1.5^\circ$ in longitude and latitude and 73 in vertical σ-levels. The stratosphere upper bound and its vertical resolution are $\sigma = 0.0002$ (about 60 km) and 500 m respectively. The interactive aerosol module [29] describing the concentration evolution of the 10 substances is included in the atmosphere model. The ocean global circulation model has horizontal resolution of $0.5^\circ \times 0.25^\circ$ in longitude and latitude and 40 vertical σ-levels. It includes dynamics and thermodynamics module [38] for the sea ice with the elastic-viscous-plastic rheology with a single gradation of thickness.

The atmosphere and the ocean global circulation models and the aerosol module are implemented as independent distributed applications that exchange data using MPI (Message Passing Interface) library when working in coupled mode.

The atmosphere global circulation model uses a semi-implicit discretization scheme that requires solving an auxiliary Helmholtz-type equation each dynamical step. In the current version a fast Fourier transform based algorithm is used which parallel implementation requires global data transposing. The scaling ability of this operation was studied in [22]. Different approaches that employ the multigrid method on massively-parallel architecture are shown to scale better [21], but they require special hardware and are not used now.

The oceanic model step consists of several stages. The hardest stage of a step is the barotropic adaptation because it requires solving a system of three implicitly discretized equations for the velocity components and the ocean level. This system is solved iteratively using the PETSc package for distributed computations [27].

The INMCM5 is good in simulation of the present-day climate [30, 31] as well as its changes in 1850–2014 [28]. This version of the INM RAS climate model takes part in the Coupled Model Intercomparison Project Phase 6 (CMIP6). In this study we use data of the following INMCM5 experiments: pre-industrial control (piControl), historical and seasonal hindcasts. The design of the piControl and the historical experiments was supplied by the CMIP6 [9].

The piControl and the historical runs were performed on the supercomputer of the Joint Supercomputer Center of the Russian Academy of Sciences (720 cores of 8-core Intel Xeon E5-2690). The INMCM5 seasonal hindcasts were produced with the INM RAS supercomputer (160 cores of 12-core Intel Xeon Silver 4214).

1.1. The piControl Run

The pre-industrial control simulation is performed under all forcings fixed at conditions of the year of 1850. There are neither naturally occurring (e. g., volcanoes and Earth's orbital characteristics) nor human-induced (e. g., land usage and greenhouse gases emissions) changes in

forcings. As a part of the CMIP6, the piControl experiment is mostly used for the Earth system models evaluation and simulation of the intrinsic climate variability. The INMCM5 piControl run lasts for 1200 model years.

1.2. Historical Runs

The historical experiment is carried out under the evolving external forcings and the anthropogenic changes in atmospheric composition. Time series of the total solar irradiance and solar spectrum, greenhouse gases and stratospheric volcanic sulfate aerosol concentrations, as well as anthropogenic emissions of SO_2, black and organic carbon, are prescribed based on observations.

The ensemble of ten INM RAS climate model historical runs was computed. The runs started with perturbed initial conditions obtained from the piControl one. The duration of each run is 165 model years from 1850 to 2014. The historical experiments demonstrate that INMCM5 simulates extreme climate and weather phenomena well [18, 26, 32].

1.3. Seasonal Hindcasts

Since recently we have been using the INMCM5 not only for climate modelling but also for weather hindcasting on a seasonal timescale. To obtain a hindcast for a winter season we set the initial states on November 1^{st}. The initial states are constructed by eliminating the bias between the simulated climate and the observed one. The bias is eliminated by adding anomalies based on the reanalyses data to the INMCM5 1980–2014 climatology obtained from the ensemble of historical runs [34, 35]. The reanalysis anomaly is the difference between the reanalysis data on November 1^{st} and its climatology. For the atmosphere and the land surface initial states we use ERA-Interim reanalysis [7]. The ocean initial states are obtained from SODA3.4.2 reanalysis [3].

The ensemble of the INMCM5 seasonal hindcasts was performed [34, 35] for every winter in the 1981–2015 period. Each hindcast lasts for 5 model months (November–March). The ensemble consists of 10 members with slightly perturbed initial air temperature and wind speed. The hindcasts data is available upon request from the authors. The correlation analysis of various hindcasted weather fields and the study on the response to the quasi-biennial oscillation is presented in [35]. The North Atlantic Oscillation and the Pacific-North American indices as well as sudden stratospheric warming events predictability are discussed in [33, 34].

1.4. ERA5 Reanalysis

We compare the INMCM5 simulations with the state-of-the-art ERA5 reanalysis [15]. It is based on the Integrated Forecasting System (IFS) Cy41r2 with the incremental 4D-Var [1] data assimilation technique. The ERA5 covers the period from 1979 and continues to be extended forward in near real time. For our purposes we download the reanalysis data with the INMCM5 horizontal resolution for the 1981–2015 winter seasons directly from C3S Climate Data Store.

2. Methods

2.1. Maximum Covariance Analysis (MCA)

Following [37] we use MCA to study the possible Eurasian autumn snow influence on the winter Northern hemisphere atmospheric circulation anomalies. With maximum covariance anal-

ysis two evolving fields $X_i(t), Y_j(t)$ are decomposed as:

$$X_i(t) = \overline{X_i} + X_i^{(1)} a_1(t) + X_i^{(2)} a_2(t) + \ldots,$$
$$Y_j(t) = \overline{Y_j} + Y_j^{(1)} b_1(t) + Y_j^{(2)} b_2(t) + \ldots.$$

Here each new term is obtained by maximization of the covariance between $a_k(t)$ and $b_k(t)$, $X_i^{(k)}, Y_j^{(k)}$ – two families of orthogonal modes with respect to the standard discrete L_2 inner product given by

$$\left(Z_p^{(m)}, Z_p^{(n)}\right)_{L_2} = \sum_p w_p Z_p^{(m)} Z_p^{(n)}, \quad w_p = \cos\varphi_p,$$

where φ_p stands for latitude of the p-th field point and Z is either X or Y. To work with more convenient Euclidean inner product, we scale the fields in the following manner

$$\tilde{Z}_p = \sqrt{w_p} Z_p, \quad \left(Z_p^{(m)}, Z_p^{(n)}\right)_{L_2} = \left(\tilde{Z}_p^{(m)}, \tilde{Z}_p^{(n)}\right) \equiv \sum_i \tilde{Z}_p^{(m)} \tilde{Z}_p^{(n)}.$$

To perform MCA, we use singular value decomposition (SVD) [2]. First the covariance matrix C is constructed:

$$C = \left(c_{ij}\right), \quad c_{ij} = \frac{1}{N_t} \sum_{t=1}^{N_t} \left(\tilde{X}_i(t) - \overline{\tilde{X}_i}\right) \left(\tilde{Y}_j(t) - \overline{\tilde{Y}_j}\right),$$

then we compute the SVD of C:

$$c_{ij} = \sum_{k=1}^{r} \sigma_k \tilde{X}_i^{(k)} \tilde{Y}_j^{(k)}, \quad \sigma_k = \operatorname{cov}\left(a_k(t), b_k(t)\right) \equiv \frac{1}{N_t} \sum_{t=1}^{N_t} a_k(t) b_k(t).$$

The obtained $\tilde{X}_i^{(k)}, \tilde{Y}_j^{(k)}$ are orthonormal as they are left and right singular vectors. The modes $X_i^{(k)}, Y_j^{(k)}$ are rescaled further to satisfy condition $\operatorname{std} a_k(t) = \operatorname{std} b_k(t) = 1$. Under this scaling $\sigma_k = \operatorname{corr}\left(a_k(t), b_k(t)\right)$.

In this research we apply the maximum covariance analysis in the region from 20°N to 80°N for the following fields:

- $\left\{X_i(t)\right\}$ – October–November Eurasian (10°W–170°W) snow cover percentage (SC) and snow water equivalent (SWE) from the piControl and the historical runs; snow water equivalent on November 1$^{\text{st}}$ (SWE^{1Nov}) based on the ERA-Interim reanalysis from the seasonal hindcasts;
- $\left\{Y_j(t)\right\}$ – December–February (DJF) averaged monthly mean sea level pressure (SLP) produced by all of the considered INMCM5 experiments.

2.2. North Atlantic Oscillation (NAO) Index Calculation

The NAO index based on simulated or reanalyses data is usually calculated [16, 17] as the expansion coefficient of the leading Empirical Orthogonal Function (EOF) of the sea level pressure (SLP) anomalies over the Atlantic (20°N–80°N, 90°W–40°E). The leading EOF is the first eigenvector of the SLP anomalies covariance matrix.

We compute the first empirical orthogonal function of winter (DJF) sea level pressure anomalies for the 1981–2015 period. The monthly mean SLP we obtain from the ERA5 reanalysis data.

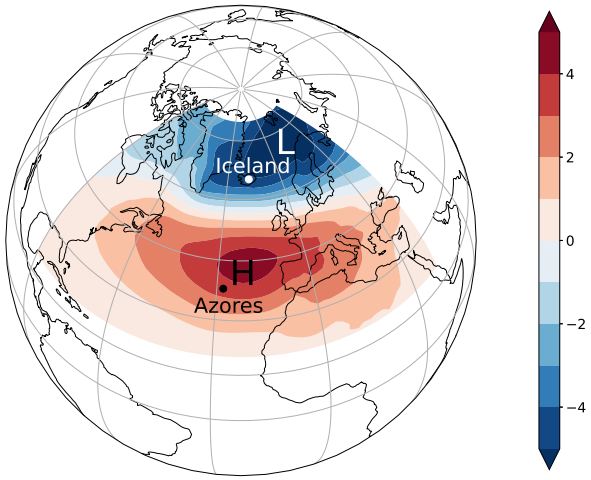


Figure 1. The first empirical orthogonal function of the DJF 1981–2015 sea level pressure anomalies based on ERA5 reanalysis data

The sea level pressure anomalies we compute relative to the reanalysis climatology over the considered period. Figure 1 shows the ERA5 leading EOF of the December–February averaged SLP anomalies.

We calculate the winter North Atlantic Oscillation index as the projection of the DJF sea level pressure anomalies on the ERA5 first empirical orthogonal function. We obtain December–February NAO index 1981–2015 time series $\mathrm{NAO_M}$ and $\mathrm{NAO_R}$ from the INMCM5 seasonal hindcasts ensemble mean and the ERA5 reanalysis SLP anomalies respectively. The $\mathrm{NAO_M}$ and $\mathrm{NAO_R}$ time series are normalized so that their standard deviations are equal to 1.

2.3. Composites of the Anomalies

2.3.1. DJF sea level pressure

We compute the DJF sea level pressure anomalies composite C_{SLP} to find a pattern corresponding to the negative phase of the North Atlantic Oscillation. Using the December–February SLP over the 20°–80°N domain and $\mathrm{NAO_R}$ obtained from the ERA5 reanalysis data we calculate C_{SLP}:

$$C_{\mathrm{SLP}} = \frac{\sum\limits_{y:\ \mathrm{NAO_R}(y)<0} \left(\mathrm{SLP}(y) - \overline{\mathrm{SLP}}\right) \cdot \mathrm{NAO_R}(y)}{\sum\limits_{y:\ \mathrm{NAO_R}(y)<0} \mathrm{NAO_R}(y)}. \tag{1}$$

From here and thereafter the y is the ordinal number of the year from the 1981–2015 period.

2.3.2. Snow water equivalent on November 1st

Snow water equivalent on November 1st $\mathrm{SWE}^{\mathrm{1Nov}}$ is used as initial state in seasonal hindcasts. It is computed from the ERA-Interim reanalysis and INMCM5 historical runs data:

$$\mathrm{SWE}^{\mathrm{1Nov}}(y) = \overline{\mathrm{SWE}_{\mathrm{M}}^{\mathrm{1Nov}}} + \left(\mathrm{SWE}_{\mathrm{R}}^{\mathrm{1Nov}}(y) - \overline{\mathrm{SWE}_{\mathrm{R}}^{\mathrm{1Nov}}}\right) \cdot \frac{\mathrm{std}\left(\mathrm{SWE}_{\mathrm{M}}^{\mathrm{1Nov}}\right)}{\mathrm{std}\left(\mathrm{SWE}_{\mathrm{R}}^{\mathrm{1Nov}}\right)}. \tag{2}$$

With the $\mathrm{SWE}^{1\mathrm{Nov}}$ anomalies in Eurasia over the 25°–80°N latitudes and $\mathrm{NAO_M}$ obtained from the INMCM5 seasonal hindcasts we calculate $C_{\mathrm{SWE}^{1\mathrm{Nov}}}$:

$$C_{\mathrm{SWE}^{1\mathrm{Nov}}} = \frac{1}{35}\sum_{y=1}^{35}\left(\mathrm{SWE}^{1\mathrm{Nov}}(y) - \overline{\mathrm{SWE}^{1\mathrm{Nov}}_{\mathrm{M}}}\right)\cdot\left(\mathrm{NAO_M}(y) - \overline{\mathrm{NAO_M}}\right). \tag{3}$$

Under the assumption of simple linear relation between $\mathrm{NAO_M}(y)$ and $\mathrm{SWE}^{1\mathrm{Nov}}(y)$ given by

$$\mathrm{NAO_M}(y) = \overline{\mathrm{NAO_M}} + \int A\cdot\left(\mathrm{SWE}^{1\mathrm{Nov}}(y) - \overline{\mathrm{SWE}^{1\mathrm{Nov}}_{\mathrm{M}}}\right)\,d\Omega,$$

the $C_{\mathrm{SWE}^{1\mathrm{Nov}}}$ represents the perturbation of snow water equivalent that would result in increasing $\mathrm{NAO_M}$ by 1:

$$\Delta\mathrm{NAO_M} = \int A\cdot C_{\mathrm{SWE}^{1\mathrm{Nov}}}\,d\Omega = \frac{1}{35}\sum_{y=1}^{35}\left(\mathrm{NAO_M}(y) - \overline{\mathrm{NAO_M}}\right)^2 = \mathrm{var}\left(\mathrm{NAO_M}(y)\right) = 1. \tag{4}$$

2.4. Data Processing and Significance Assessment

The data of each INMCM5 experiment are grouped into a sequence of overlapping 35-year intervals. For the piControl run we take every fifth year as interval start thus forming 234 intervals. For the historical runs we take each year as an interval start forming 130 intervals per ensemble member. Finally, for the seasonal hindcasts we have a single 35-year interval per ensemble member. In figures intervals are denoted by their first years.

For each of the 35-year intervals we perform MCA for the SLP and one of the snow describing field (SC, SWE and $\mathrm{SWE}^{1\mathrm{Nov}}$). Only the leading pair of modes $X_i^{(1)}, Y_j^{(1)}$ is considered. We are also interested in the following quantities:

- temporal correlation coefficient for the leading pair

$$\sigma_1 = \mathrm{corr}\left(a_1(t), b_1(t)\right);$$

- fraction of snow field variance, explained by its first mode term

$$v_1^X = \frac{\mathrm{var}\left(X_i^{(1)}a_1(t)\right)}{\mathrm{var}\,X_i(t)};$$

- same for the sea level pressure

$$v_1^Y = \frac{\mathrm{var}\left(Y_j^{(1)}b_1(t)\right)}{\mathrm{var}\,Y_j(t)};$$

- spatial correlation coefficient between the pressure mode and the SLP composite

$$r = \mathrm{corr}_{L_2}\left(Y_j^{(1)}, C_{\mathrm{SLP}}\right).$$

The 95% confidence intervals for each quantity are estimated using bootstrap technique [6].

For each experiment we plot $X_i^{(1)}$ and $Y_j^{(1)}$ averaged by all intervals for which r is greater than the fifth percentile of all r for that experiment, i. e., five percent of intervals with the least correlation between the pressure mode and the SLP composite are excluded from the average.

3. Results

Following (1) we compute the sea level pressure anomaly composite that corresponds to the negative NAO phase from the ERA5 reanalysis data. This composite is shown in Fig. 2. It is later compared with pressure modes obtained by MCA.

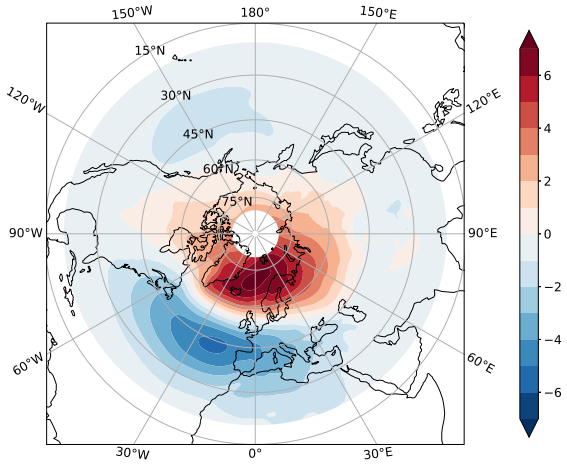


Figure 2. Composite of the sea level pressure anomalies corresponding to the negative NAO phase based on ERA5 reanalysis data (C_{SLP}, hPa)

3.1. The piControl Run

From the piControl data we study the autumn snow – winter pressure teleconnection using internally-generated snow cover percentage (SC) and snow water equivalent (SWE). We repeat MCA for 234 overlapping 35-year intervals that span the 1200-year run period.

The results of MCA applied to SC–SLP and SWE–SLP pairs are summarized in Tab. 1 and Tab. 2. Each table contains σ_1, the temporal correlation between the leading modes of snow and pressure; $v_1^{\mathrm{SC}}, v_1^{\mathrm{SWE}}$, the fraction of variance that is explained by the first mode of snow; v_1^{SLP}, the fraction of variance that is explained by the first mode of pressure and r, the spatial correlation between the SLP mode and C_{SLP}.

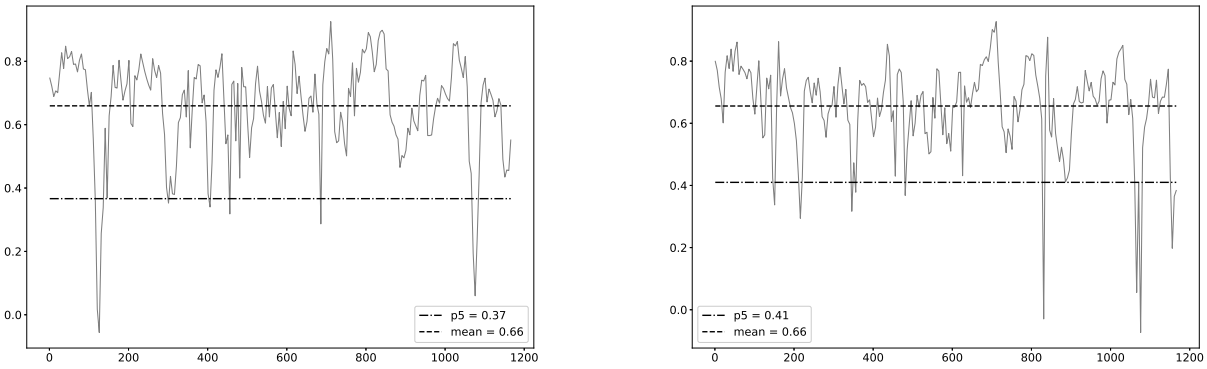
Table 1. Results of MCA applied to SC–SLP pair from the piControl run

	$\sigma_1, \%$	$v_1^{\mathrm{SC}}, \%$	$v_1^{\mathrm{SLP}}, \%$	$r, \%$
mean	74.1	7.1	28.2	65.9
95% confidence interval	[73.4, 74.8]	[6.9, 7.4]	[27.4, 29.1]	[63.8, 67.8]

Table 2. Results of MCA applied to SWE–SLP pair from the piControl run

	$\sigma_1, \%$	$v_1^{\mathrm{SWE}}, \%$	$v_1^{\mathrm{SLP}}, \%$	$r, \%$
mean	76.8	6.0	28.6	65.6
95% confidence interval	[76.3, 77.4]	[5.8, 6.1]	[27.8, 29.4]	[63.6, 67.2]

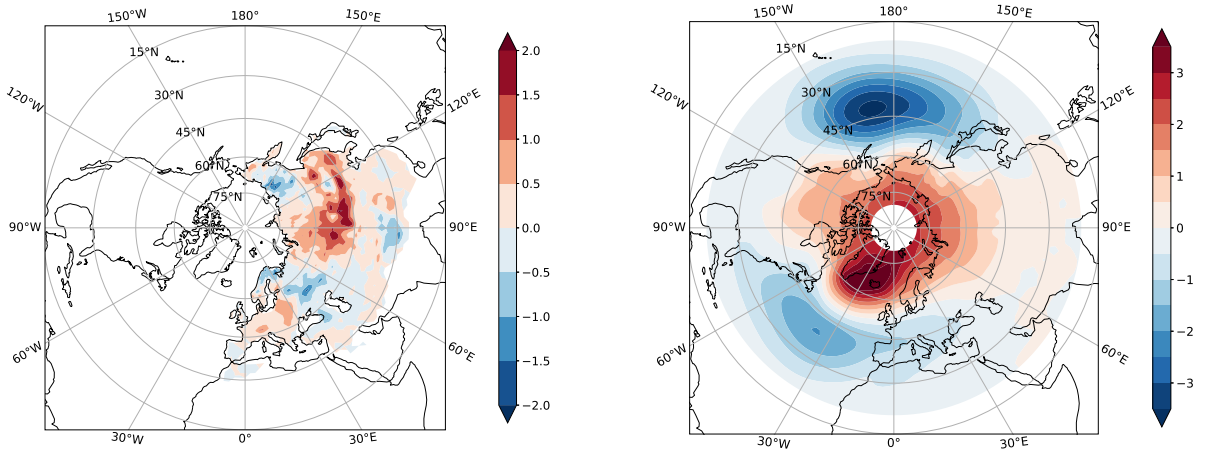
Figure 3 shows the spatial correlation coefficient r plotted against the first year of the corresponding 35-year interval. For the most of intervals the correlation coefficient stays above 40%.



(a) SC–SLP pair (b) SWE–SLP pair

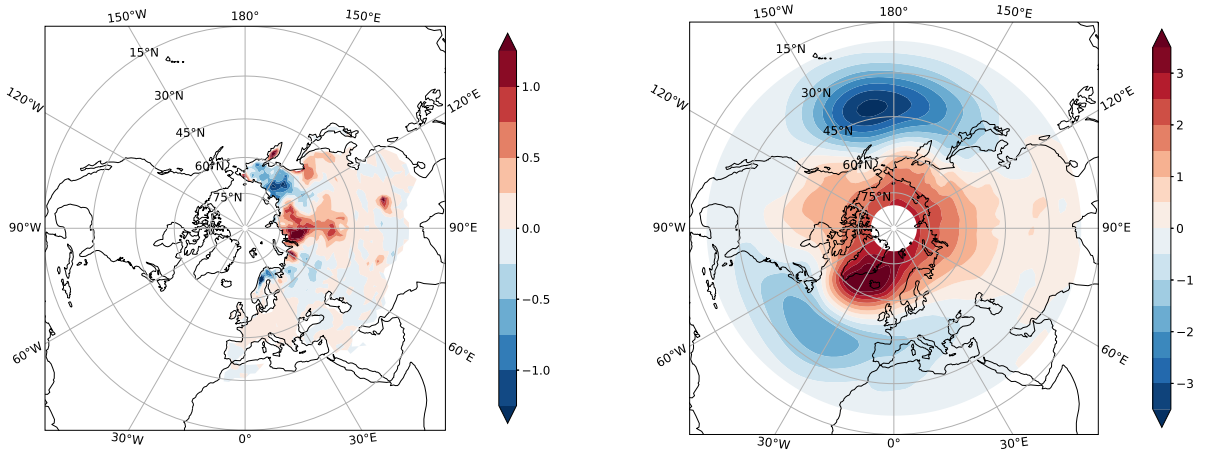
Figure 3. Spatial correlation coefficient r between the pressure mode and the SLP composite in piControl run for series of 234 MCA experiments

Figures 4 and 5 show leading MCA modes for SC–SLP and SWE–SLP pairs averaged by all intervals except for those where spatial correlation r was extremely low (below its fifth percentile).



(a) SC leading mode, % (b) SLP leading mode, hPa

Figure 4. Averaged leading MCA modes for SC–SLP pair in piControl run



(a) SWE leading mode, mm (b) SLP leading mode, hPa

Figure 5. Averaged leading MCA modes for SWE–SLP pair in piControl run

The SLP leading mode for the SC–SLP and SWE–SLP pairs is almost the same. It demonstrates typical North Atlantic Oscillation and Pacific-North American patterns.

The SC MCA first mode has its maximum in Siberia and its minimum in the East European Plain and is in good agreement with the results from [37].

The SWE mode demonstrates similar to SC distribution, but extrema are located closer to the North Pole.

3.2. Historical Runs

The key difference between piControl and historical runs is that the latter is carried out under evolving forcings. Like in the piControl run the data is grouped in 35-year intervals, except that now we take every model year as an interval start and have 10 ensemble members for each interval. Following results are obtained by treating each ensemble member separately, i. e., no ensemble averaging is performed to reduce the smoothing of the extrema.

Similar to piControl the MCA results for historical runs are summarized in Tab. 3 and Tab. 4

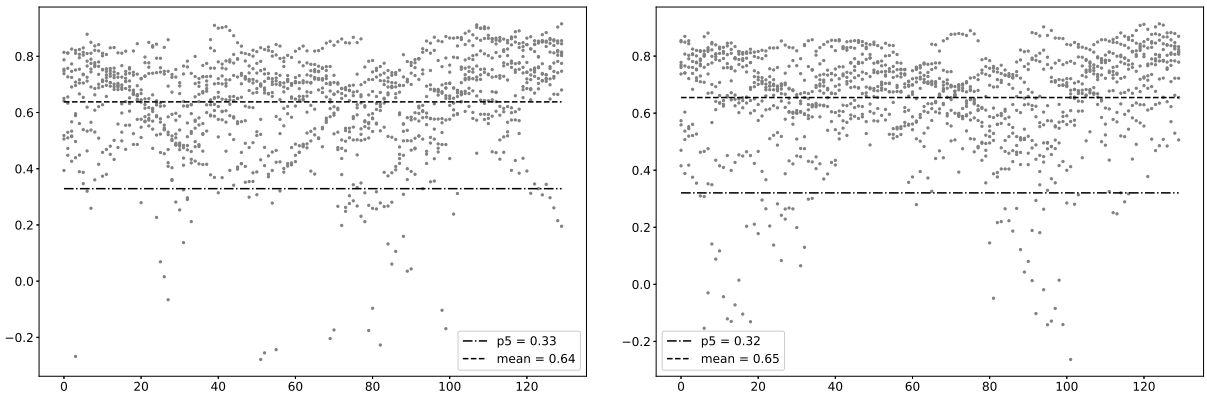
Table 3. Results of MCA applied to SC–SLP pair from the historical runs

	$\sigma_1, \%$	$v_1^{\mathrm{SC}}, \%$	$v_1^{\mathrm{SLP}}, \%$	$r, \%$
mean	72.9	6.9	29.0	63.8
95% confidence interval	[72.7, 73.2]	[6.8, 7.0]	[28.7, 29.3]	[62.8, 64.7]

Table 4. Results of MCA applied to SWE–SLP pair from the historical runs

	$\sigma_1, \%$	$v_1^{\mathrm{SWE}}, \%$	$v_1^{\mathrm{SLP}}, \%$	$r, \%$
mean	77.0	6.0	29.6	65.5
95% confidence interval	[76.7, 77.2]	[5.9, 6.1]	[29.3, 30.0]	[64.5, 66.4]

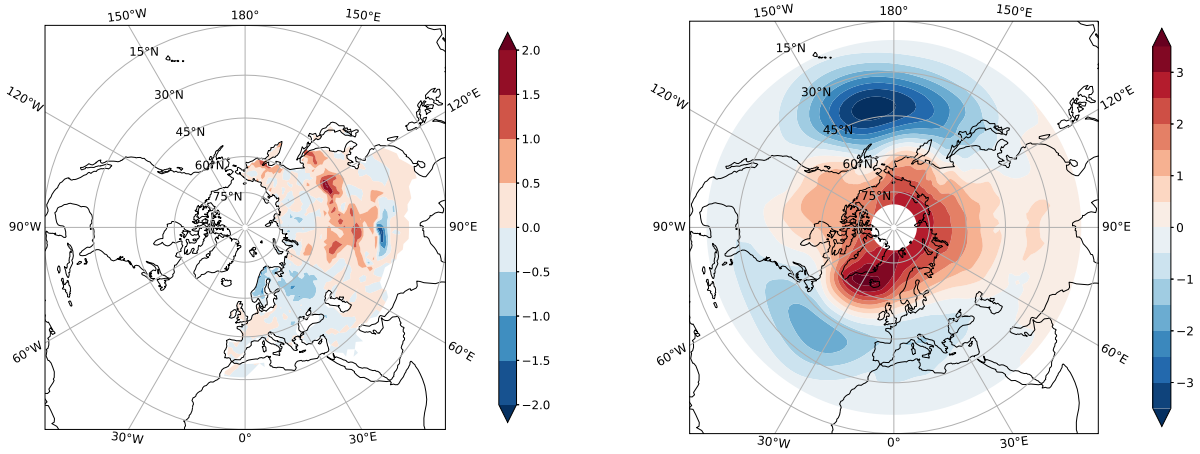
Figure 6 shows a scatter plot of the spatial correlation coefficient r for each of the 10 ensemble members of the historical runs drawn against the first year of the corresponding interval.



(a) SC–SLP pair (b) SWE–SLP pair

Figure 6. Spatial correlation coefficient r between the pressure mode and the SLP composite in 10 historical runs for series of 130 MCA experiments

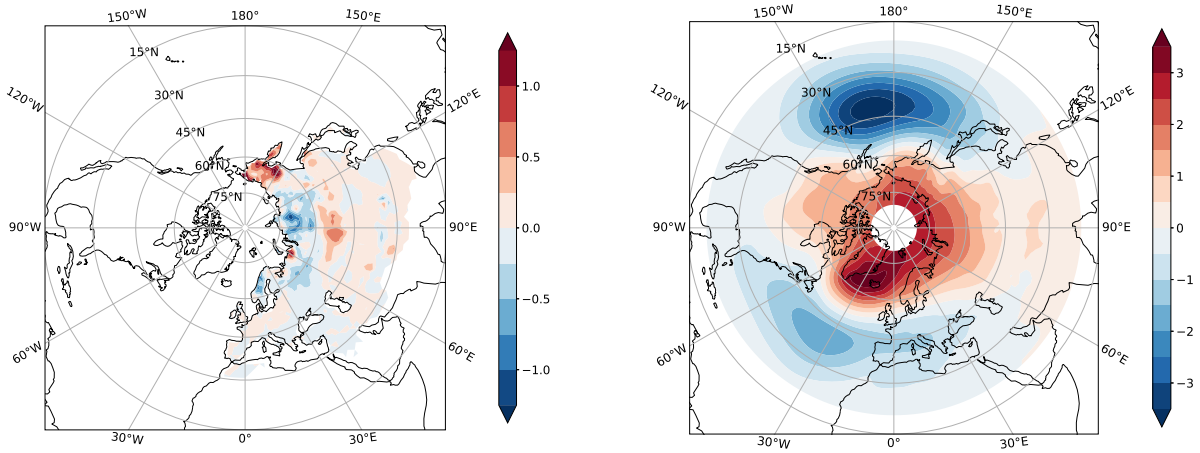
Just like in the piControl case we drop 5% of outliers with the least value of the spatial correlation coefficient r for each ensemble member and average the resulting set of pressure and snow modes. The averaged modes are presented in Fig. 7 and Fig. 8.



(a) SC leading mode, %

(b) SLP leading mode, hPa

Figure 7. Averaged leading MCA modes for SC–SLP pair in historical runs



(a) SWE leading mode, mm

(b) SLP leading mode, hPa

Figure 8. Averaged leading MCA modes for SWE–SLP pair in historical runs

The results obtained from historical runs are in excellent agreement with the ones from the piControl run, except for the snow modes that are slightly smoother. We believe this is due to averaging them over the individual ensemble members.

Comparing the results for piControl and historical runs we conclude that evolving forcings do not produce any significant impact on the simulation of the considered phenomenon.

3.3. Seasonal Hindcasts

From the results for piControl and historical runs it becomes clear that there is no significant difference whether we perform MCA on SC–SLP or SWE–SLP pair. In both cases we get quite close results for numerical quantities (σ_1, v_1, r) as well as for averaged SLP mode.

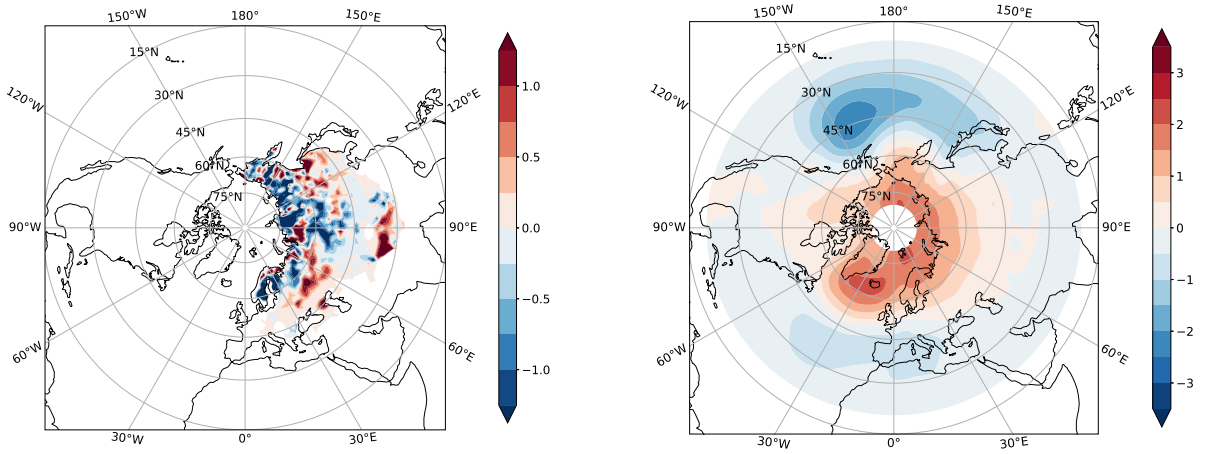
Thus for the seasonal hindcasts we do not consider SC and only focus on SWE^{1Nov}. The other reason to do so is that SC is a diagnostic variable while SWE^{1Nov} is a prognostic variable. SWE^{1Nov} is explicitly set in the initial conditions for the November 1st via (2).

For the seasonal hindcasts we have only one 35-year interval and 10 ensemble members. The results of MCA performed for SWE^{1Nov} and SLP are summarized in Tab. 5.

Table 5. Results of MCA applied to SWE^{1Nov}–SLP pair from the seasonal hindcasts

	$\sigma_1, \%$	$v_1^{SC}, \%$	$v_1^{SWE^{1Nov}}, \%$	$r, \%$
mean	53.6	26.0	18.5	41.2
95% confidence interval	[50.3, 57.5]	[23.6, 27.5]	[15.3, 21.2]	[26.1, 55.2]

The SLP leading mode pattern presented in Fig. 9b shows resemblance to the patterns obtained from piControl and historical runs, but the extrema in the Atlantic have less absolute value. In contrast, the SWE^{1Nov} pattern shown in Fig. 9a is noisy and lacks many features that it has in piControl and historical runs, for example in the Eastern Europe and the Southern Siberia regions.



(a) SWE^{1Nov} leading mode, mm

(b) SLP leading mode, hPa

Figure 9. Averaged leading MCA modes for SWE^{1Nov}–SLP pair in seasonal hindcasts

The results from seasonal hindcasts significantly differ from the results in piControl and historical runs. There may be several explanations for this inconsistency:

- SWE^{1Nov} field is not internally generated by the model but supplied from reanalysis instead;
- model run is not continuous because it is restarted each November 1st with new initial conditions;
- snow itself might not be the only cause of the teleconnection.

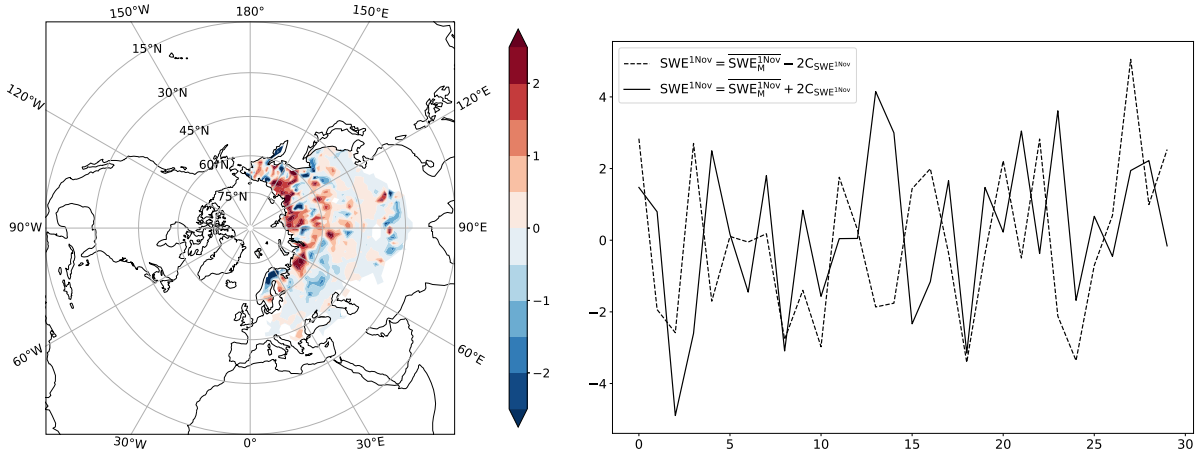
3.4. NAO Sensitivity to Perturbations of Initial Snow Water Equivalent

To study the North Atlantic Oscillation sensitivity to snow water equivalent perturbations we carry out an ensemble of runs similar to seasonal hindcasts. The snow water equivalent anomaly is substituted with $\pm 2 \cdot C_{SWE^{1Nov}}$ and the other initial states are taken equal to INMCM5 climatology computed from the historical runs. The ensemble consists of 30 runs for each of the

$\mathrm{SWE}^{1\mathrm{Nov}} = \overline{\mathrm{SWE}_{\mathrm{M}}^{1\mathrm{Nov}}} + 2 \cdot C_{\mathrm{SWE}^{1\mathrm{Nov}}}$ ("addition") and $\mathrm{SWE}^{1\mathrm{Nov}} = \overline{\mathrm{SWE}_{\mathrm{M}}^{1\mathrm{Nov}}} - 2 \cdot C_{\mathrm{SWE}^{1\mathrm{Nov}}}$ ("subtraction") experiments. The $C_{\mathrm{SWE}^{1\mathrm{Nov}}}$ is obtained by (3) and is presented in Fig. 10a.

Assuming linear relation between NAO index and $\mathrm{SWE}^{1\mathrm{Nov}}$ we expect the NAO index to be increased by 2 in the "addition" experiment and to be decreased by 2 in the "subtraction" experiment according to (4).

The obtained values of NAO index for each ensemble member for both experiments are presented in Fig. 10b. The mean values of NAO index are -0.07 (95% CI $[-0.81, 0.70]$) for the "subtraction" experiment and 0.23 (95% CI $[-0.56, 0.97]$) for the "addition" experiment. Mann-Whitney test [19], applied to the two series of NAO values produces pvalue = 0.46 which means that there is no significant response in NAO index to altering $\mathrm{SWE}^{1\mathrm{Nov}}$.



(a) Snow water equivalent composite ($C_{\mathrm{SWE}^{1\mathrm{Nov}}}$, mm)

(b) NAO index plotted against the number of run

Figure 10. Sensitivity experiments to snow water equivalent perturbations

Conclusion

The piControl and historical runs show that INMCM5 is capable of simulating the autumn Eurasian snow – winter NAO teleconnection. This was not the case for the previous version of the model (INMCM4) which participated in CMIP5 [12, 20]. A possible explanation is that INMCM5 has a higher upper atmosphere bound than INMCM4 and has a finer vertical resolution in the stratosphere. According to [4], the stratosphere plays a key role in the mechanism of the teleconnection.

There is almost no difference in results obtained by applying MCA to SC–SLP and SWE–SLP pair which indicate that both snow coverage and snow water equivalent may be equally used to study the autumn snow influence on winter atmospheric circulation anomalies. The leading MCA mode of the sea level pressure has both North Atlantic Oscillation and Pacific-North American patterns.

A possible reason for INMCM5 seasonal hindcasts not capturing the phenomenon may be the way we feed the model with the reanalysis data. Instant change in prognostic variables can cause perturbations that might not settle on the seasonal time scale or might destroy the initial state pattern. These perturbations can be reduced by employing continuous observational or reanalysis data assimilation.

The experiments of altering initial snow water equivalent in INMCM5 seasonal hindcasts indicate that Eurasian autumn snow itself does not significantly affect winter NAO. However, good teleconnection simulation obtained in piControl and historical runs might also indicate that autumn snow is only a manifestation of other phenomena that affect winter North Atlantic Oscillation in INMCM5.

Acknowledgements

The research was supported by the Russian Science Foundation, project No. 20-17-00190 (analysis of the piControl and historical runs) and by the Russian Foundation for Basic Research, project No. 20-05-00673 (analysis of the seasonal hindcasts and NAO sensitivity). The winter seasonal hindcasts and the sensitivity experiments were obtained using the HPC computation cluster at the Marchuk Institute of Numerical Mathematics of the Russian Academy of Sciences.

References

1. Bonavita, M., Hólm, E., Isaksen, L., Fisher, M.: The evolution of the ECMWF hybrid data assimilation system. Quarterly Journal of the Royal Meteorological Society 142(694), 287–303 (2016). https://doi.org/10.1002/qj.2652

2. Bretherton, C.S., Smith, C., Wallace, J.M.: An Intercomparison of Methods for Finding Coupled Patterns in Climate Data. Journal of Climate 5(6), 541–560 (1992). https://doi.org/10.1175/1520-0442(1992)005<0541:AIOMFF>2.0.CO;2

3. Carton, J.A., Chepurin, G.A., Chen, L.: SODA3: A New Ocean Climate Reanalysis. Journal of Climate 31(17), 6967–6983 (2018). https://doi.org/10.1175/JCLI-D-18-0149.1

4. Cohen, J., Barlow, M., Kushner, P.J., Saito, K.: Stratosphere-Troposphere Coupling and Links with Eurasian Land Surface Variability. Journal of Climate 20(21), 5335–5343 (2007). https://doi.org/10.1175/2007JCLI1725.1

5. Cohen, J., Entekhabi, D.: Eurasian snow cover variability and northern hemisphere climate predictability. Geophysical Research Letters 26(3), 345–348 (1999). https://doi.org/10.1029/1998GL900321

6. Davison, A.C., Hinkley, D.V.: Bootstrap Methods and their Application. Cambridge University Press, Cambridge (1997). https://doi.org/10.1017/CBO9780511802843

7. Dee, D.P., Uppala, S.M., Simmons, A.J., et al.: The ERA-Interim reanalysis: configuration and performance of the data assimilation system. Quarterly Journal of the Royal Meteorological Society 137(656), 553–597 (2011). https://doi.org/10.1002/qj.828

8. Dunstone, N., Smith, D., Scaife, A.: Skilful predictions of the winter North Atlantic Oscillation one year ahead. Nature Geoscience 9(11), 809–814 (2016). https://doi.org/10.1038/ngeo2824

9. Eyring, V., Bony, S., Meehl, G.A., et al.: Overview of the Coupled Model Intercomparison Project Phase 6 (CMIP6) experimental design and organization. Geoscientific Model Development 9(5), 1937–1958 (2016). https://doi.org/10.5194/gmd-9-1937-2016

10. Fletcher, C.G., Hardiman, S.C., Kushner, P.J., Cohen, J.: The Dynamical Response to Snow Cover Perturbations in a Large Ensemble of Atmospheric GCM Integrations. Journal of Climate 22(5), 1208–1222 (2009). https://doi.org/10.1175/2008JCLI2505.1

11. Fletcher, C.G., Kushner, P.J., Cohen, J.: Stratospheric control of the extratropical circulation response to surface forcing. Geophysical Research Letters 34(21), L21802 (2007). https://doi.org/10.1029/2007GL031626

12. Furtado, J.C., Cohen, J.L., Butler, A.H., et al.: Eurasian snow cover variability and links to winter climate in the CMIP5 models. Climate Dynamics 45(9), 2591–2605 (2015). https://doi.org/10.1007/s00382-015-2494-4

13. Gong, G., Entekhabi, D., Cohen, J.: Modeled Northern Hemisphere Winter Climate Response to Realistic Siberian Snow Anomalies. Journal of Climate 16(23), 3917–3931 (2003). https://doi.org/10.1175/1520-0442(2003)016<3917:MNHWCR>2.0.CO;2

14. Hardiman, S.C., Kushner, P.J., Cohen, J.: Investigating the ability of general circulation models to capture the effects of Eurasian snow cover on winter climate. Journal of Geophysical Research: Atmospheres 113(D21), 123 (2008). https://doi.org/10.1029/2008JD010623

15. Hersbach, H., Bell, B., Berrisford, P., et al.: The ERA5 global reanalysis. Quarterly Journal of the Royal Meteorological Society 146(730), 1999–2049 (2020). https://doi.org/10.1002/qj.3803

16. Hurrell, J.W.: Decadal Trends in the North Atlantic Oscillation: Regional Temperatures and Precipitation. Science 269(5224), 676–679 (1995). https://doi.org/10.1126/science.269.5224.676

17. Hurrell, J.W., Deser, C.: North Atlantic climate variability: The role of the North Atlantic Oscillation. Journal of Marine Systems 78(1), 28–41 (2009). https://doi.org/10.1016/j.jmarsys.2008.11.026

18. Kim, Y.H., Min, S.K., Zhang, X., et al.: Evaluation of the CMIP6 multi-model ensemble for climate extreme indices. Weather and Climate Extremes 29, 100269 (2020). https://doi.org/10.1016/j.wace.2020.100269

19. Mann, H.B., Whitney, D.R.: On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. The Annals of Mathematical Statistics 18(1), 50–60 (1947). https://doi.org/10.1214/aoms/1177730491

20. Martynova, Y.V.: October snow cover and winter atmospheric conditions in Siberia. In: International Young Scientists School and Conference on Computational Information Technologies for Environmental Sciences, May 27 – June 6, 2019. IOP Conference Series: Earth and Environmental Science, vol. 386, p. 012001. IOP Publishing (2019). https://doi.org/10.1088/1755-1315/386/1/012001

21. Mortikov, E.: The efficiency of the implementation of iterative methods for the solution of elliptic equations in atmospheric general circulation models on massively parallel systems. In: Sobolev, S., Voevodin, V. (eds.) 1st Russian Conference on Supercomputing Days 2015, RuSCDays 2015s. CEUR Workshop Proceedings, vol. 1482, pp. 528–534. CEUR-WS (2015), http://ceur-ws.org/Vol-1482/528.pdf

22. Mortikov, E.V.: Improving scalability of the high spatial resolution earth system model software complex. In: Parallelnye vychislitelnye tekhnologii (PaVT 2015). pp. 431–435 (2015), http://omega.sp.susu.ru/books/conference/PaVT2015/short/102.pdf, (in Russian)

23. Saito, K., Cohen, J., Entekhabi, D.: Evolution of Atmospheric Response to Early-Season Eurasian Snow Cover Anomalies. Monthly Weather Review 129(11), 2746–2760 (2001). https://doi.org/10.1175/1520-0493(2001)129<2746:EOARTE>2.0.CO;2

24. Scaife, A.A., Arribas, A., Blockley, E.: Skillful long-range prediction of European and North American winters. Geophysical Research Letters 41(7), 2514–2519 (2014). https://doi.org/10.1002/2014GL059637

25. Smith, D.M., Scaife, A.A., Eade, R.: Seasonal to decadal prediction of the winter North Atlantic Oscillation: emerging capability and future prospects. Quarterly Journal of the Royal Meteorological Society 142(695), 611–617 (2016). https://doi.org/10.1002/qj.2479

26. Tarasevich, M.A., Volodin, E.M.: Influence of various parameters of INM RAS climate model on the results of extreme precipitation simulation. In: International Young Scientists School and Conference on Computational Information Technologies for Environmental Sciences, May 27 – June 6, 2019. IOP Conference Series: Earth and Environmental Science, vol. 386, p. 012012. IOP Publishing (2019). https://doi.org/10.1088/1755-1315/386/1/012012

27. Terekhov, K.M., Volodin, E.M., Gusev, A.V.: Methods and efficiency estimation of parallel implementation of the σ-model of general ocean circulation. Russ. J. Numer. Anal. Math. Modelling 26(2), 189–208 (2011). https://doi.org/10.1515/rjnamm.2011.011

28. Volodin, E.M., Gritsun, A.S.: Simulation of observed climate changes in 1850–2014 with climate model INM-CM5. Earth System Dynamics 9(4), 1235–1242 (2018). https://doi.org/10.5194/esd-9-1235-2018

29. Volodin, E.M., Kostrykin, S.V.: The aerosol module in the INM RAS climate model. Russian Meteorology and Hydrology 41(8), 519–528 (2016). https://doi.org/10.3103/S106837391608001X

30. Volodin, E.M., Mortikov, E.V., Kostrykin, S.V., et al.: Simulation of modern climate with the new version of the INM RAS climate model. Izvestiya, Atmospheric and Oceanic Physics 53(2), 142–155 (2017). https://doi.org/10.1134/S0001433817020128

31. Volodin, E.M., Mortikov, E.V., Kostrykin, S.V., et al.: Simulation of the present-day climate with the climate model INMCM5. Climate Dynamics 49(11), 3715–3734 (2017). https://doi.org/10.1007/s00382-017-3539-7

32. Volodin, E.M., Tarasevich, M.A.: Simulation of Climate and Weather Extreme Indices with the INM-CM5 Climate Model. Russian Meteorology and Hydrology 43(11), 756–762 (2018). https://doi.org/10.3103/S1068373918110067

33. Vorobyeva, V., Volodin, E.: Analysis of the predictability of stratospheric variability and climate indices based on seasonal retrospective forecasts of the INM RAS climate model. Russian Journal of Numerical Analysis and Mathematical Modelling 36(2), 117–126 (2021). https://doi.org/10.1515/rnam-2021-0010

34. Vorobyeva, V., Volodin, E.: Evaluation of the INM RAS climate model skill in climate indices and stratospheric anomalies on seasonal timescale. Tellus A: Dynamic Meteorology and Oceanography 73(1), 1–12 (2021). https://doi.org/10.1080/16000870.2021.1892435

35. Vorobyeva, V.V., Volodin, E.M.: Experimental Studies of Seasonal Weather Predictability Based on the INM RAS Climate Model. Mathematical Models and Computer Simulations 13(4), 571–578 (2021). https://doi.org/10.1134/S2070048221040232

36. Wang, L., Ting, M., Kushner, P.J.: A robust empirical seasonal prediction of winter NAO and surface climate. Scientific Reports 7(1), 279 (2016). https://doi.org/10.1038/s41598-017-00353-y

37. Wu, Q., Hu, H., Zhang, L.: Observed Influences of Autumn-Early Winter Eurasian Snow Cover Anomalies on the Hemispheric PNA-like Variability in Winter. Journal of Climate 24(7), 2017–2023 (2011). https://doi.org/10.1175/2011JCLI4236.1

38. Yakovlev, N.G.: Reproduction of the large-scale state of water and sea ice in the Arctic Ocean in 1948–2002: Part I. Numerical model. Izvestiya, Atmospheric and Oceanic Physics 45(3), 357–371 (2009). https://doi.org/10.1134/S0001433809030098

DOI: 10.14529/jsfi210404

Representation of Spatial Data Processing Pipelines Using Relational Database

Igor G. Okladnikov[1,2]

A methodology for representation of spatial data processing pipelines using relational database within the framework of the computing backend of the online information-analytical system "Climate" (`http://climate.scert.ru`) is proposed. Each pipeline is represented by a sequence of instructions for the computing backend describing how to run data processing modules and pass datasets between them (from the output of one module to the input of another one), including raw data and final computational results obtained in graphical or binary formats. Using relational database for storing descriptions of processing pipelines used in the "Climate" system provides flexibility and efficiency while adding and developing spatial data processing modules. It also provides computing pipelines scaling for further implementation for multiprocessor systems.

Keywords: spatial data, information systems, databases, workflow, directed multigraph, processing pipeline, climate research.

Introduction

Usually, data analysis process represents a set of sequential operations starting from data search and retrieval and ending with the output of results in the required format. Depending on the complexity of the research method chosen, such a sequence might consist of three or more relatively simple computational procedures where intermediate results are passed from one to another. For instance, to calculate a quite simple climatic index "Monthly maximum of minimum daily temperatures", first it is necessary to read data from corresponding files (`https://www.climdex.org/learn/indices/#index-Tnx`), then to find the minimum daily values, then in the obtained values to find the monthly maximum value and finally to display and save the result, for example, in the graphical format. In the case when it is additionally necessary to calculate the trend of the index over several years one more operation is added. Practically in most studies such a procedure is either completely or partially a manual process when each such operation is performed by a researcher independently using various software products, starting from the very beginning every time. To automate this process specialized software products aimed at eliminating the need for regular routine actions and thereby speeding up the research might be used. One of such software products is the information-analytical system "Climate" [1] which allows solving problems of various range of complexity for the Earth science field, so that hiding complex technical and routine operations from the system user. However, even to be able to use the automation tools for computational processes, user (or developer) needs special skills and considerable time to form the necessary sequences of operations, namely, computing pipelines, for each separate type of data processing and analysis. Thus, an urgent task is to formalize the representation of computing (data processing) pipelines in a convenient and standardized form, that makes it possible to facilitate and force the process of their formation, modification, and reuse. This will contribute to the implementation of the current FAIR principles used for the management of scientific data and results (`https://www.go-fair.org/fair-principles/`), within the framework of any information-

[1]Institute of Monitoring of Climatic and Ecological Systems of the Siberian Branch of the Russian Academy of Sciences, Tomsk, Russian Federation
[2]Federal Research Center for Information and Computational Technologies, Tomsk, Russian Federation

analytical system. This paper describes a methodology for representing computing pipelines as modified labeled oriented multigraphs with their subsequent translation to relational database. The methodology is quite universal and might be adapted for other information and analytical systems.

1. General Approach

Within the framework of the digital information-analytical system "Climate", the developed computing modules providing unified programming interface are used for spatial data internal batch processing. A computing module is an isolated set of internal data structures and functions (class) that has an application programming interface (API) for input arguments (input data and control parameters), returning results (output data), and running module itself. A computational module can be represented as a function f, such that $Y = f(X)$, where X is a vector of arguments, Y is a vector of results, and each element of the vector is a multidimensional array of spatial data. A sequence of computing module calls where the results of one module are passed to the input of another one forms a computing pipeline. The pipeline described using one of the conventional technical formats (XML, JSON, etc.) is passed to the computing backend of the system which sequentially runs modules providing them with the necessary data and parameters. As a rule, the first module in the conveyor receives as an input one or more datasets, representing multidimensional arrays of climatic data obtained as a result of numerical modeling or observations. Subsequently, the datasets, representing intermediate processing results, are transferred along the pipeline from module to module. This procedure, aimed at performing a specialized processing of climatic data by "running" them through the computing pipeline, is known within the framework of the system by the name "processor". Each type of data processing (for example, the calculation of the specific climatic index) has its own processor. Parameters of the processor consist of the set parameters of the corresponding computing modules. Each parameter can take one of several valid values (or range of values). These can be threshold values of some measurable quantity (for example, the daily maximum temperature that must be greater than the threshold value of 25°C for calculating the climatic index "Number of summer days" and less than 0°C for calculating the index "Number of frost days"), or the choice of the one of the predetermined time period types for which an index is defined (day, month, year, etc.). Different values of the processor parameters affect the operation of its constituent computing modules and lead to different results. "Processor configuration" is a set of parameters of the computing pipeline that took one of the valid values. As part of the interaction with the "Climate" system the user selects a processor using a graphical interface and sets its parameters which are then passed to the computing modules thereby forming a specific configuration. Processor parameters have two types: variables (user-defined) and constants. The values of the variable parameters depend on the choice the user made using the interface. Constant parameter values do not change and are the inherent properties of each processor.

In the process of developing the digital information-analytical system "Climate", the first version of the metadata database was developed, built on the basis of the MySQL DBMS and designed to store information about the spatial datasets and processors available to the user [2]. As a result of the evolutionary development of the system it became obvious that the information about the processors in the metadata database should be extended using descriptions of the computing pipelines associated with these processors. The previously used approach to representing computing pipelines as separate XML files [3] proved to be inconvenient and unpromising from

the point of view of further system development. The adding of the computing pipeline descriptions to the metadata database provides flexibility and efficiency to the procedure of extending the computing backend functionality. It also provides the "client-server" model realization at the computing backend level, that, in turn, increases flexibility and reliability of its functioning within the framework of the distributed computing technology implemented in the "Climate" system.

To represent a computing pipeline within the scope of the relational database such as MySQL, initially it is convenient to display it as a graph reflecting the workflow [4]. Such a graph should describe not only the sequence of operations, but also the directions of the datasets transferred between them, thus separating the data flow and the control flow [5]. As it is shown in the number of related works, this task is solved by introducing additional properties to the graph (for instance, state vectors or "messages" of various types to pass the information between operations), labels of vertices and arcs, etc. [5–8]. Within the framework of the digital information-analytical system "Climate" to represent the computing pipeline in the form of a graph with its following translation to relational database, only the necessary additional constructions are introduced: labels of vertices and arcs. Using these, a specific computing module is assigned to each vertex, and dataset passed from one module to another is assigned to each arc. The dataset might represent processor parameters as well as intermediate or final results of calculations. Each arc has additional numeric labels that indicate the position of each dataset in the result data vector and in the vector of computing module arguments (thus defining the order of arguments and results for each dataset and computing module). Then, key-value pairs are associated with some vertices to specify the condition of their presence in the graph. It is generally accepted to use a labeled oriented multigraph [9] as a basis for representing such a workflow. The multigraph should be modified by adding extra labels for vertices and end labels for arcs.

The following is the methodology for representing the workflow of the data processing pipeline using modified labeled oriented multigraph as well as an information model that provides its implementation in the advanced version of the metadata database of the digital information-analytical system "Climate".

2. Methodology

2.1. Data Processing Pipeline Graph Representation

First, the workflow of an arbitrary computing pipeline might be presented as a simple labeled oriented multigraph $G\,(V, A, s, t, \Sigma_V, \Sigma_A, l_V, l_A)$, where V is the set of the graph vertices; A is a set of arcs connecting them; $s : A \to V$, $t : A \to V$ — two mappings that define the source and target vertices of an arc; Σ_V, Σ_A are two sets of labels of vertices and arcs containing the names of computing modules and datasets, respectively; $l_V : V \to \Sigma_V$, $l_A : A \to \Sigma_A$ are two mappings that assign labels to vertices and arcs. The vertices of such a graph correspond to the associated computing module calls, and the arcs correspond to the operations of transferring control, datasets, and processor parameters between modules. Let us modify this graph so that it fully reflects the computing pipeline of the "Climate" system.

Conditional vertices. Let us assume that K is a set of names of processor parameters, W is a set of corresponding allowed parameter values and the mapping $Z : K \to W$ defines which values each parameter can take. Let us introduce the mapping $F : V^c \to Z$ that defines

for graph vertices belonging to the set $V^c \subset V$ the conditions in the form of "key-value" pairs which consist in the equality of the parameter $k \in K$ to the one of the allowed values $Z(k)$. The vertex $v^c \in V^c$ is suggested to be called "conditional", and the set $F(V^c)$ is the "set of graph conditions". A graph G^G containing conditional vertices will be called "generalized". The generalized graph takes its final form only at runtime of the computing backend. Due to this fact, different computing pipelines can be formed from the same generalized graph "on the fly" both for different processors as well as for different configurations of the same processor depending on the user's choice. The graph $G^D \subseteq G^G$ that has taken its final form at runtime of the computing backend will be called "determined". An example of the generalized graph containing a conditional vertex (highlighted by a dotted line) is shown in Fig. 1.

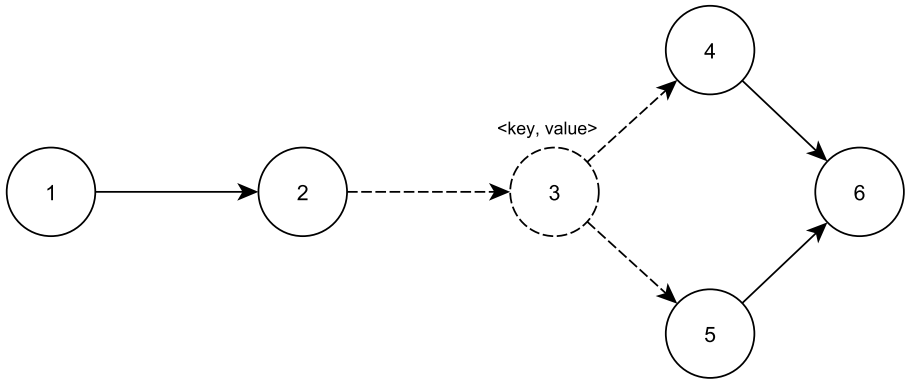


Figure 1. Generalized graph containing the conditional vertex 3. The condition of vertex existence in the processor current configuration is the presence of the parameter "key" with the value "value"

Let us introduce a conditional vertex deleting rules: 1) if a conditional vertex is removed from the generalized graph, then the incoming arc of the deleted vertex is redirected to the vertex to which the outgoing arc of the deleted vertex is directed; 2) if there are several incoming arcs, then all of them are redirected to the vertex to which the outgoing arc of the deleted vertex is directed; 3) if there are several outgoing arcs, then the incoming arc is redirected to each vertex where the outgoing arcs belonging to the removed vertex are directed; 4) if a vertex has several incoming and outgoing arcs, then such a vertex cannot be deleted and, therefore, cannot be conditional. Arcs corresponding to data transmission only are not considered and are removed along with the vertex.

If the condition associated with the conditional vertex is met for the processor configuration selected by the user, then at runtime the conditional vertex is preserved in the graph ($G^D \equiv G^G$) (see Fig. 2).

If the condition associated with the conditional vertex is not met for the processor configuration selected by the user (the configuration does not contain the corresponding parameter-value pair), the conditional vertex at runtime is removed from the graph according to the introduced vertex deleting rule thereby forming a subgraph $G^D \subseteq G^G$ (see Fig. 3).

Thanks to this technique different processors can use the same generalized graph, building on its basis the required computing pipeline using their own configuration.

End labels of arcs. The arcs A of the labeled oriented multigraph G representing a computing pipeline of the "Climate" system correspond either to the operations of transferring control and datasets between the computing modules or only to the operations of transferring data that determine the configuration of the processor. The vertex of the graph arc comes out

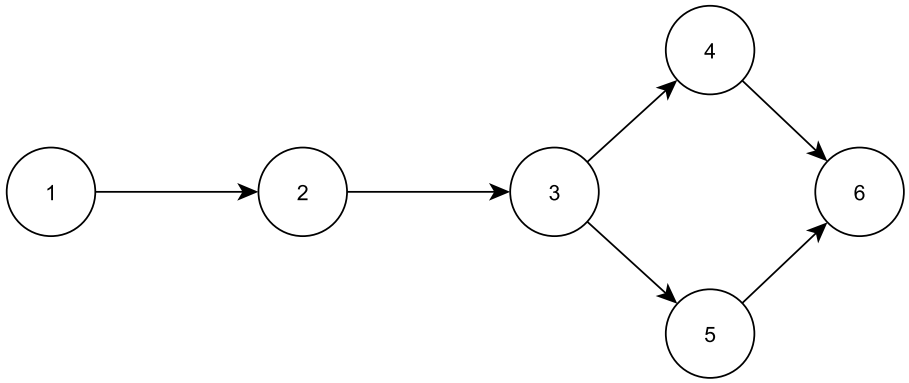


Figure 2. Graph with the vertex 3 when the condition is met

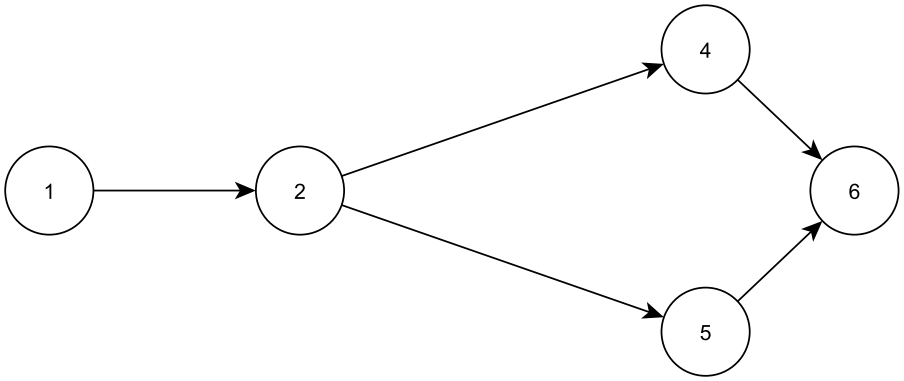


Figure 3. Subgraph when the vertex 3 is absent since the condition is not met

from will be called "source", and the vertex into which the arc enters – "target". The set Σ_A contains special labels corresponding to different datasets or processor properties. If several datasets are passed from one module to another, multiple arcs with their own identification are introduced between the corresponding vertices of the graph, using one for each dataset. Since the order of the arguments and results is important to the module, each arc should be assigned two additional "end" labels: "output index" and "input index". When considering the dataset corresponding to an arc, the output index is its position in the vector of the source vertex results, and the input index is the position in the vector of the target vertex arguments. To connect arcs with these labels two mappings should be introduced $l_o : A \to N$, $l_i : A \to N$, which determine the source vertex output index and the target vertex input index of each arc, where N is the set of natural numbers. The generalized graph with added arc end labels is shown in Fig. 4.

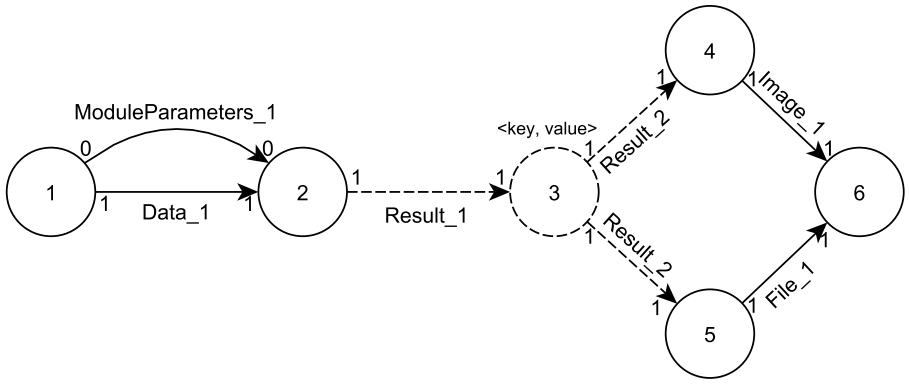


Figure 4. Processing pipeline graph containing conditional vertex and arc labels

Thus, the computing pipeline of the "Climate" system might be represented as a generalized modified labeled oriented multigraph $G^G\,(V, A, Z, F, s, t, \Sigma_V, \Sigma_A, l_V, l_A, l_o, l_i)$ that takes its final form (forming the required computing pipeline) only at runtime of the computing backend based on the processor configuration specified by the user interacting with the graphical interface. Additionally, it should be noted that due to the specifics of the task manager of the Climate platform, the presence of loops in graphs is not allowed.

2.2. Representation of the Computing Pipeline Graph in the Relational Database

To translate the graph of the computing pipeline to the relational database, an appropriate information model should be developed. The following entities will be required:

- graph vertex,
- computing module,
- graph arc,
- dataset,
- processor,
- processor configuration,
- parameter,
- parameter value,
- computing pipeline.

Several processors might be associated with a single computing pipeline. Each pipeline is represented as a graph consisting of vertices and arcs. Each vertex is associated with the corresponding computational module, and if the vertex is conditional, with a parameter-value pair (combination) specifying the condition. Each arc connects two vertices (source and target), the output index of the source vertex, the input index of the target vertex, and the dataset label associated with the arc. The computing module always has a unique name. The parameter also has a unique name, valid parameter values having unique values. The processor has a unique name and is associated with the processor configuration. The processor configuration is associated with pairs of parameter names and valid values. Each parameter can correspond to several valid values that it can take. And each valid value can be matched by several parameters that can take it. Datasets have unique conditional labels.

The conceptual diagram of the information model displaying a computing pipeline graph is as follows:

1. processor
   (a) processor ID (PK);
   (b) pipeline ID (FK).
2. computing pipeline
   (a) pipeline ID (PK);
   (b) pipeline description.
3. vertex
   (a) vertex ID (PK);
   (b) computing module ID (FK);
   (c) ID of combination specifying the condition (FK).
4. arc
   (a) arc ID (PK);

   (b) pipeline ID (FK);
   (c) source vertex ID (FK);
   (d) target vertex ID (FK);
   (e) source vertex output index;
   (f) target vertex input index;
   (g) dataset ID (FK).
5. computing module
   (a) computing module ID (PK);
   (b) computing module name.
6. parameter
   (a) parameter ID (PK);
   (b) parameter name.
7. parameter value
   (a) parameter value ID (PK);
   (b) parameter value name.
8. dataset
   (a) dataset ID (PK);
   (b) dataset label.
9. processor configuration
   (a) processor configuration ID (PK);
   (b) processor configuration description.
10. combination ("parameter-value" pair)
   (a) combination ID (PK);
   (b) parameter ID (FK);
   (c) parameter value ID (FK).
11. processor has processor configuration
   (a) processor ID (FK);
   (b) processor configuration ID (FK);
12. processor configuration contains a combination
   (a) processor configuration ID (FK);
   (b) combination ID (FK).

Here PK mean primary key attributes while FK mean foreign key attributes. The conceptual ER diagram using Crow's Foot notation [10] is presented in Fig. 5.

2.3. Building the Computing Pipeline Based on the Relational Database

To build a computing pipeline based on the information contained in the relational database, it is necessary to execute several SQL queries. The nature and specification of such queries depends on the specific DBMS and SQL language version, so they are not presented in this work. The result of the SQL queries execution is the information about the vertices and associated computing modules as well as the arcs that connect them. Having this information in mind, it is possible to restore a generalized computing pipeline graph in the computer's RAM using any conventional way (for instance, using an adjacency list by the method of Guido van Rossum, `https://www.python.org/doc/essays/graphs/`).

To be able to obtain a generalized graph specific implementation, it is required having a list of parameters selected by the user, to enumerate all the conditional vertices of the built generalized

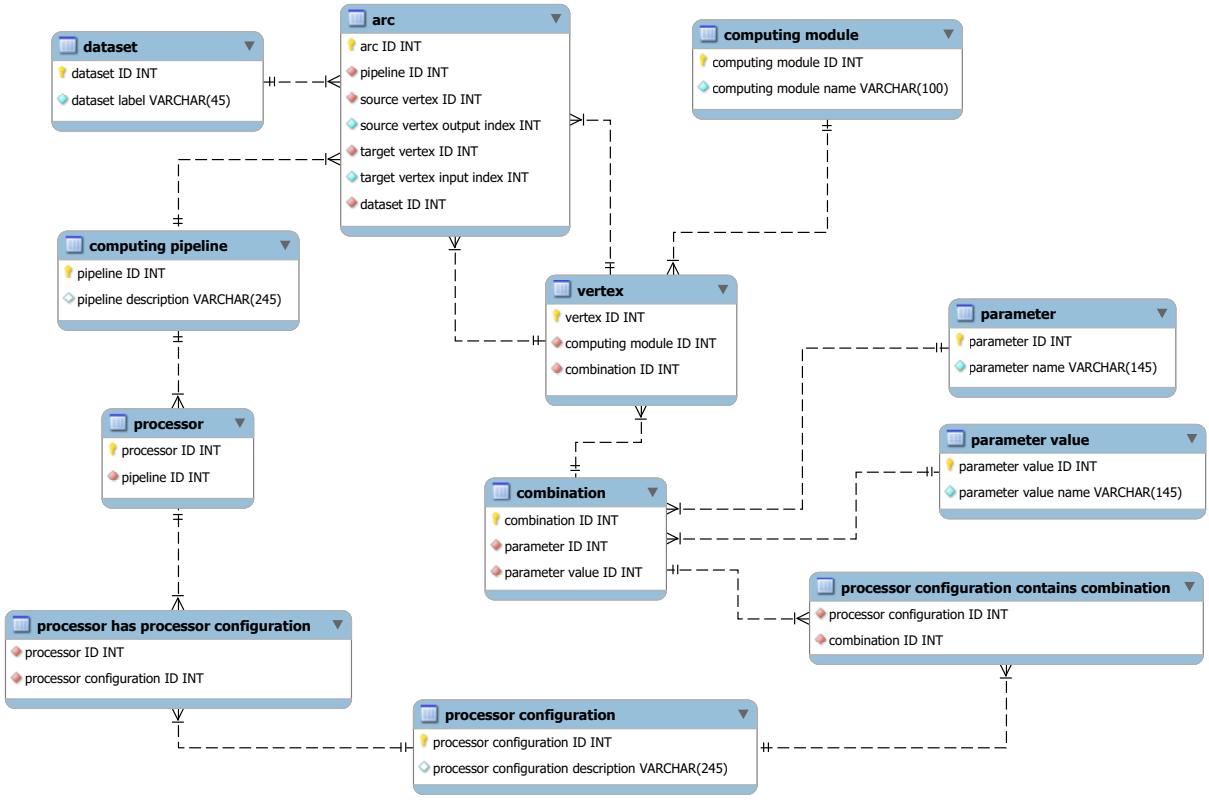


Figure 5. ER model representing a processing pipeline generalized graph

graph and then, according to the methodology presented in this work, to delete those that do not meet the conditions. The graph obtained represents a sequence of operations of the computing pipeline that meets the user's requirements. The sequence might be extracted, for example, by performing starting from the starting vertex a complete traversal of the graph using the breadth-first search method [11]. Then the operation sequence should be transformed into description of the computing pipeline with the following transfer to the corresponding computing unit of the system backend. A single pipeline can be associated with several processors thus providing the reuse of already developed pipelines. In this case the mechanism of conditional vertices allows, using various parameters of the processor, to obtain various specific implementations of computing pipelines. This reduces the time required to add new processors to the system and greatly simplifies the work of developers thus speeding up the process of bringing new features to the system users.

2.4. Climatic Index Example

Let us consider a simple case based on the example of building a computing pipeline for the "Monthly maximum of daily minimum temperatures" index (Fig. 6).

The figure depicts a generalized graph for a given data processing pipeline. The following constant vertices corresponding to the computing modules can be distinguished: cvcCalcMaximum, which basically calculates the index at each point of the geographic grid, and cvcOutput which outputs the results of the computations in the format required. The yellow dashed line outlines the conditional vertices cvcCalcTrendTM calculating the trend values, and cvcCalcTiMean which associated module calculates the average values of the index in the case the user selects more than one annual period for processing (then the index is calculated for the selected

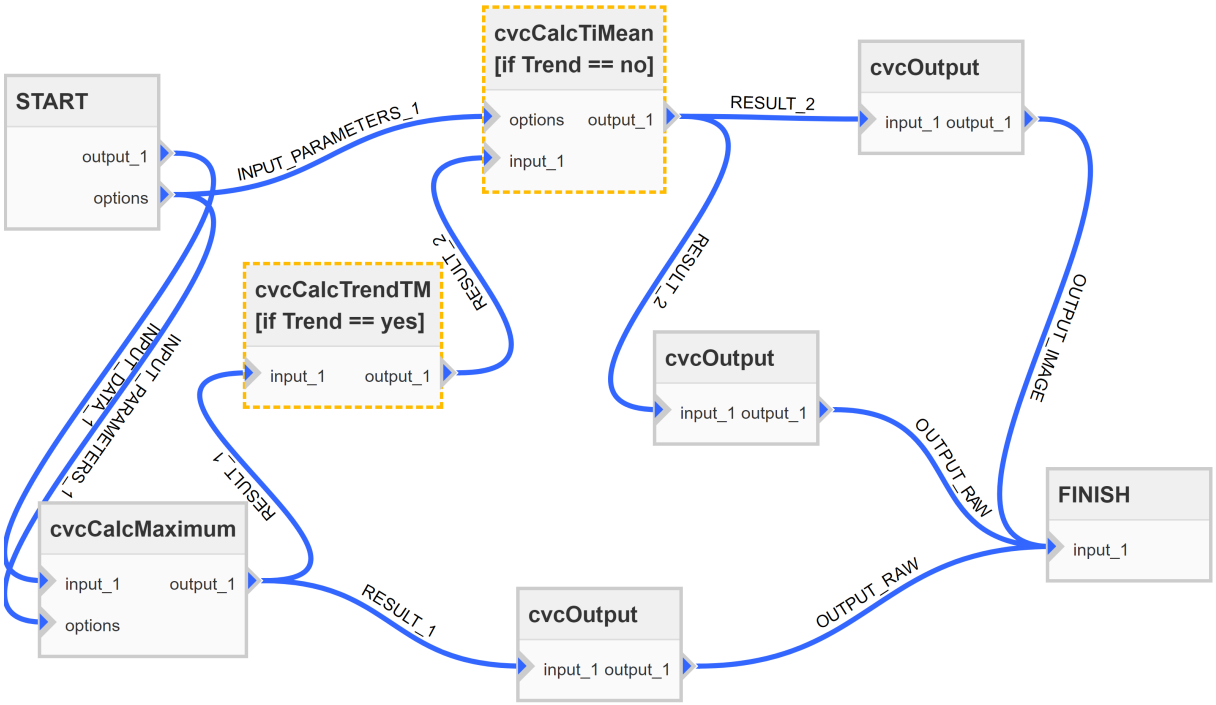


Figure 6. Processing pipeline of the climate index "Monthly maximum of daily minimum temperatures"

month of each year, for example, July). The preserving condition for the cvcCalcTrendTM vertex is the presence of the "Trend" parameter with the "yes" value (the user selected the trend calculation using the graphical interface), and for the cvcCalcTiMean vertex – the "Trend" parameter with the "no" value (the user did not select the trend calculation). Thus, a specific pipeline implementation contains only one of these conditional vertices. The INPUT_DATA_1 arc corresponds to the dataset being processed, for example, the ERA-Interim reanalysis [12], the INPUT_PARAMETERS_1 arc corresponds to the configuration parameters of the processor (computing modules), the RESULT_1 and RESULT_2 arcs correspond to the intermediate calculation results in the form of datasets, OUTPUT_IMAGE arc corresponds to the calculation result in the graphical format (GeoTIFF, Shapefile), and OUTPUT_RAW arcs correspond to the calculation results in the format of a multidimensional binary array of spatial data (NetCDF).

Conclusion

The proposed methodology allows to effectively represent spatial data processing pipelines in the relational metadata database of the digital information-analytical system "Climate". Thanks to the "generalized" graph approach different processors can use the same graph. Based on the graph different computing pipelines are generated for different values of configuration parameters. Translating descriptions of generalized computing pipelines to relational database provides flexibility and efficiency in adding new and revising existing spatial data processing modules as well as provides computing pipelines scaling for further implementation for multiprocessor systems. The methodology is substantially universal and might be adapted for other information and analytical systems, as well as find application in other subject areas.

Acknowledgements

The reported study was funded by the State project No. 121031300158-9.

References

1. Gordov, E., Shiklomanov, A., Okladnikov, I., et al.: Development of Distributed Research Center for analysis of regional climatic and environmental changes. IOP Conference Series: Earth and Environmental Science 48, 012033 (2016). https://doi.org/10.1088/1755-1315/48/1/012033
2. Okladnikov, I.G., Gordov, E.P., Titov, A.G.: Development of climate data storage and processing model. IOP Conference Series: Earth and Environmental Science 48, 012030 (2016). https://doi.org/10.1088/1755-1315/48/1/012030
3. Okladnikov, I.G.: Computing core of a software package for "cloud" analysis of climate change and the environment. IOP Conference Series: Earth and Environmental Science 611, 012058 (2020). https://doi.org/10.1088/1755-1315/611/1/012058
4. Swamy, M.N.S., Thulasiraman, K.: Graphs, Networks, and Algorithms. Wiley-Blackwell, New York (1980)
5. Allamanis, M., Brockschmidt, M., Khademi, M.: Learning to represent programs with graphs. International Conference on Learning Representations (ICLR) (2018). https://openreview.net/pdf?id=BJOFETxR-
6. Chang, C.L.: Interpretation and execution of fuzzy programs. In: Zadeh, L.A., et al. (eds.) Fuzzy Sets and Their Applications to Cognitive and Decision Process, pp. 191–218. Academic Press, New York (1975)
7. Averkin, A.N., Batyrshin, I.Z., Blishun, A.F., et al.: Fuzzy sets in models of control and artificial intelligence. Nauka, Moscow (1986)
8. Li, Yu., Tarlow, D., Brockschmidt, M., Zemel, R.: Gated graph sequence neural networks. International Conference on Learning Representations (ICLR) (2015). https://arxiv.org/pdf/1511.05493.pdf
9. Balakrishnan, V.K.: Graph Theory. McGraw-Hill (1997)
10. Halpin, T.: Entity Relationship modeling from an ORM perspective: Part 1. Object Role Modeling. http://www.orm.net/pdf/JCM11.pdf, accessed: 2020-02-25
11. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill, Cambridge (2009)
12. Dee, D.P., Uppala, S.M., Simmons, A.J., et al.: The ERAInterim reanalysis: Configuration and performance of the data assimilation system. Quarterly Journal of the royal meteorological society 137(656), 553–597 (2011). https://doi.org/10.1002/qj.828

DOI: 10.14529/jsfi210405

Direct Numerical Simulation of Stratified Turbulent Flows and Passive Tracer Transport on HPC Systems: Comparison of CPU Architectures

Evgeny V. Mortikov[1,2,3], *Andrey V. Debolskiy*[1,3,4]

In this paper we assess the influence of CPU architectures commonly used in HPC systems on the efficiency of the implementation of algorithms used for direct numerical simulation (DNS) of turbulent flows. We consider a stably stratified turbulent plane Couette flow as a benchmark problem supplemented with the additional transport of passive substances. The comparison includes the Intel Xeon, AMD Rome x86 CPU architecture processors and the Huawei Kunpeng ARM CPU processor. We discuss the role of memory-oriented optimizations on the efficiency of tracer transport implementation on each platform.

Keywords: turbulence, direct numerical simulation, ARM, supercomputing.

Introduction

Numerical simulation of geophysical turbulent flows today remains one of the most challenging computational problems. The large-scale complex climate and weather forecast models, research studies of the physics of turbulence dynamics and engineering computational fluid dynamics (CFD) applications belong to foremost demanding tasks for all HPC centers. All of these problems involve the numerical solution of hydrodynamic equations obtained from the well-known Navier-Stokes system of equations with additional simplifications implied for specific problem.

While large body (which to the author's knowledge is an understatement) of scientific literature is devoted to the development of efficient numerical methods, algorithms for solutions of CFD problems, the emergence of exascale computing presents new and continuing challenges: how well do commonly used and well-known approaches are suited for the current and next-generation HPC systems? Major challenges are related to the massive concurrent and highly heterogeneous environments in terms of both memory and computations used or expected to be used in supercomputers, resulting from the development of more energy- and computationally-efficient hardware. Even the often considered algorithms for solution of wide range of problems in CFD applications, e.g., FFT or multigrid methods, to name a few, may require reformulation and optimization for such HPC architectures [2, 8].

Though a notable development of supercomputers in the last decades is the advent of co-processors, e.g., GPU-based computing, a more recent approach for building HPC systems is associated with the ARM-based CPUs designed with energy-efficiency considerations kept at the forefront [19]. These considerations are essentially tied with the computational performance achievable by large-scale HPC systems highly restricted by power consumption. A notable example in this regard is the Fugaku supercomputer with ARM-based A64FX 48C 2.2GHz CPUs at the RIKEN Center for Computational Science, which leads the June 2021 TOP500 list of fastest HPC systems.

In this paper we consider DNS (Direct Numerical Simulation) of turbulence as a computational benchmark to compare the performance of CPUs with distinct architecture: Intel Xeon

[1]Lomonosov Moscow State University Research Computing Center, Moscow, Russian Federation
[2]Marchuk Institute of Numerical Mathematics RAS, Moscow, Russian Federation
[3]Moscow Center of Fundamental and Applied Mathematics, Moscow, Russian Federation
[4]A.M. Obukhov Institute of Atmospheric Physics RAS, Moscow, Russian Federation

Gold, AMD Rome and the ARM-based Huawei Kunpeng 920. DNS of turbulence implies the numerical solution of Navier-Stokes system of equations *as is* and consequently requires to resolve all energy significant scales of turbulent motions down to the smallest scales, where the dissipation of energy takes place. While LES (Large-Eddy Simulation) and RANS (Reynolds-Averaged Navier-Stokes) models may be used to study flows on larger scales (up to planetary one), they rely on additional turbulence closure assumptions to represent the momentum, heat and other scalar fluxes attributed to the small scales, which are not resolved on the computational grid or excluded from the coarse physical model itself. On the contrary DNS models are devoid of such assumptions and provide invaluable data for insight into turbulence dynamics, development of RANS and LES turbulence models and parameterizations albeit at the expense of very high computational cost [11, 12]. These high computational requirements of DNS are especially pronounced for geophysical turbulence. For example, a crude estimate may be obtained considering the Reynolds number Re, which represents the ratio of largest to smallest scales of motion and may reach values of around 10^7–10^9 in the atmospheric and oceanic boundary layers [22, 31]. The size of a computational grid for a DNS of 3D turbulence is known to scale as $\mathrm{Re}^{9/4}$, which shows that doubling of the Reynolds number increases the computational cost by almost an order of magnitude when accounting also for the time dependence of equations of motion and assuming linear computational complexity (in terms of number of grid cells) of the algorithm. To put that into perspective, numerical simulations, for example, of turbulent channel flows require on the order of 10^3 CPU cores for Re values exceeding 10^5 and feasible computational time frame, see, e.g. [16].

The computational benchmark considered in this study is based on stably stratified turbulent plane Couette flow, which is extensively used in studies of turbulence dynamics [7, 17, 29] and verification of turbulence closures developed for large-scale models of atmosphere and ocean [16, 30]. The Couette flow is a shear flow of viscous fluid between two plane parallel walls moving relative to each other in the absence of external pressure gradient. The simple formulation and geometry of the flow eases implementation of numerical algorithms and their comparison.

The problems of urban air quality, chemical pollutant dispersion, aerosol distribution forecast add substantial computational complexity in the atmospheric models. In general they may require numerical solution of more than one hundred of additional prognostic equations for concentrations of different tracer species [20]. The similar computational problem also arises in ocean-biochemistry coupled models [26]. In this study we consider the transport of passive tracers as a proxy of such models, supplementing the DNS model of turbulent plane Couette flow with advection-diffusion type equations for substances concentration. This allows us to assess the efficiency of the implementation of the algorithms for solving scalar transport equations on different CPU architectures and the role of memory optimizations.

The paper is structured as follows. In Section 1 we introduce the governing equations of the DNS model and the numerical methods. The implementation aspects of the DNS code used for comparison study are reviewed in Section 2. The setup of numerical experiments and the details of code tuning and compilation for specific CPUs are given in Sections 3 and 4. The results of performance comparison and the influence of memory optimizations are discussed in Section 5, followed by summary and conclusions.

1. Governing Equations and Numerical Method

The dynamics of thermally stratified fluid are governed by the Navier-Stokes system of equations in the Boussinesq approximation. The corresponding momentum, continuity and heat advection-diffusion equations in divergent form are written as:

$$\frac{\partial u_i}{\partial t} + \frac{\partial u_i u_j}{\partial x_j} = -\frac{1}{\rho_0}\frac{\partial p}{\partial x_i} + \nu\frac{\partial^2 u_i}{\partial x_j \partial x_j} + \alpha g \delta_{i3}\Theta, \tag{1}$$

$$\frac{\partial u_i}{\partial x_i} = 0, \tag{2}$$

$$\frac{\partial \Theta}{\partial t} + \frac{\partial u_i \Theta}{\partial x_i} = \chi\frac{\partial^2 \Theta}{\partial x_i \partial x_i}, \tag{3}$$

where $\mathbf{u}(\mathbf{x},t) = (u_1, u_2, u_3)^T \equiv (u, v, w)^T$ is the velocity vector, $p(\mathbf{x},t)$ – pressure, $\Theta(\mathbf{x},t)$ – potential temperature, ν and χ are molecular coefficients of kinematic viscosity and thermal diffusivity respectively, ρ_0 is the reference density, $\mathbf{x} = (x_1, x_2, x_3)^T \equiv (x, y, z)^T$ and t is time. The last term on the right-hand size of the equation (1) describes buoyancy forces acting on the fluid in the vertical direction, where α – thermal expansion coefficient, g – acceleration due to gravity and δ_{ij} – Kronecker delta.

Besides the equations (1)–(3) we consider additional passive tracer transport equations for substances concentrations $C_k(\mathbf{x},t)$ expressed as:

$$\frac{\partial C_k}{\partial t} + \frac{\partial u_i C_k}{\partial x_i} = \chi_k\frac{\partial^2 C_k}{\partial x_i \partial x_i}, \tag{4}$$

where $1 \le k \le n_c$ and no summation over index k is assumed, n_c – number of tracer species and χ_k is molecular diffusivity coefficient for k-th tracer.

The projection method [4] is applied for the time advancement of momentum equations (1) coupled with incompressibility constraint (2). At the first stage the intermediate velocity $\tilde{u}_i$ is calculated using the flow state at the n-th time step:

$$\frac{\tilde{u}_i - u_i^n}{\Delta t} = \frac{3}{2}R_i^n - \frac{1}{2}R_i^{n-1} - \frac{1}{\rho_0}\frac{\partial p^n}{\partial x_i} + \alpha g \delta_{i3}\Theta^{n+1}. \tag{5}$$

Here explicit in time second-order Adams-Bashforth approximation is used for the right-hand side R_i, which contains the contributions from advection and diffusion fluxes. The velocity u_i on the new $(n+1)$-th time step is calculated at the second stage of projection method as:

$$\frac{u_i^{n+1} - \tilde{u}_i}{\Delta t} = -\frac{\partial q}{\partial x_i}, \tag{6}$$

where Δt is the discrete time step and q is a scalar correction to the pressure field and $p^{n+1} = p^n + \rho_0 q$. The pressure correction is obtained by applying the divergence operator to equation (6) and solving the Poisson equation:

$$\frac{\partial^2 q}{\partial x_i \partial x_i} = \frac{1}{\Delta t}\frac{\partial \tilde{u}_i}{\partial x_i}. \tag{7}$$

This ensures that the velocity field satisfies the continuity equation at each time step. The boundary conditions for the Poisson problem directly follow from the boundary conditions on the velocity field with equations (1) and (2) taken into account. The heat and scalar transport

equations are solved in the same way using the explicit in time Adams-Bashforth second-order method, so the temperature Θ^{n+1} is known at the first stage (5) of projection method.

Second or fourth-order finite-difference schemes [14, 25] on rectangular structured grids with staggered arrangement of nodes are used for approximation of spatial derivatives. The finite-difference approximation is momentum and energy conservative with the advection terms expressed in skew-symmetric form.

Biconjugate gradient stabilized (BiCGstab) iterative method [27] with a V-cycle geometric multigrid preconditioner [24] is used for solving the finite-difference approximation of the Poisson equation (7). On each grid in the multigrid sequence, smoothing iterations are performed by successive upper relaxation method (SOR) for red-black ordering of nodes. The projection onto coarse grid and the prolongation operator onto a fine grid correspond to a bilinear interpolation consistent with the averaging operator used in finite-difference stencils.

2. Implementation

For computational tests and performance comparison we use the unified DNS -, LES -, RANS - code [15, 16] developed at the Research Computing Center of Lomonosov Moscow State University jointly with the Marchuk Institute of Numerical Mathematics RAS, which solves a set of partial differential equations governing the dynamics of geophysical boundary layers, with focus on simulating atmospheric and oceanic boundary layer structure and evolution. The numerical model is especially suited for simulating the flows over a complex terrain, e.g. an urbanized surface *inter alia*, using immersed boundary methods [15]. The code implements a collection of finite-difference and finite-volume schemes, iterative and direct methods for solution of systems of linear equations, turbulence and subgrid closures in the LES and RANS framework, particle transport and tracking submodule with different options of complexity. The DNS -, LES -, RANS - model is extensively used for turbulence dynamics research [16, 29, 30], simulations of the atmospheric flows [7, 9, 23] and studies of lake hydrodynamics [6, 21]. The code is written in C/C++ programming language and supports hybrid architectures of modern day supercomputers using the combined MPI-OpenMP-CUDA stack.

In its simplest form the algorithm for solving the particular discrete problem (1)–(4) corresponds to a linear algebra problem with large sparse matrices, e.g., consists of matrix-vector/matrix multiplications, calculation of linear combinations of vectors, dot and cross products, solution of linear systems of equations, etc. Matrix operations in the code are implemented in "matrix-free" form meaning that the matrix elements are not stored in memory and instead matrix-vector products are represented as functions corresponding to some discretization method. This allows to significantly decrease the number of memory access operations, which is a common bottleneck for these types of problems. We stress that the algorithm implies calculations involving sparse matrices and the benchmarks similar to HPCG [5], in general, will be more representative of the implementation performance, compared with common benchmarks (e.g., HPL) oriented at dense matrix problems.

MPI library is used for 1D, 2D or 3D spatial decomposition of the computational grid among MPI-processes. Different algorithms for MPI exchanges are implemented as their efficiency is known to be dependent on the computational platform, in particular the ones based on MPI derived datatypes or manual packing and unpacking of messages, with blocking or nonblocking MPI primitives, etc. Common optimizations for improving scaling on HPC systems include options for combining MPI data exchanges for a number of arrays (e.g., vector or tensor compo-

nents) or increasing the width of the grid halo region (see, e.g. [3]) for reducing latency of MPI communications. The latter allows to reduce the number of calls to MPI functions but at the cost of additional computational overhead, which may be negligible when the size of the problem on MPI-process is comparatively small. For large-scale simulations (where the grid may contain more than 10^8 cells) the code also makes use of parallel file I/O operations through MPI-IO interface.

A hybrid MPI-OpenMP approach is supported, where the calculations on each MPI-process are performed by OpenMP threads. In this case only the thread support mode `MPI_THREAD_FUNNELED` is implemented, that is each MPI-process is multi threaded, but only the master thread performs calls to MPI communication functions. In parts of the code, where large MPI communication overhead is expected (e.g., iterative methods for solving linear systems), non blocking MPI subroutines are used with overlapping computations and communications where possible. The implementation of MPI communications and computational functions makes use of the OpenMP "orphan" directives, which if necessary allows to decrease the number of implicit thread synchronization points by merging code parts in a single `parallel` region.

The code is structured in a such way as to separate the solution of high-level "numerical" and "physical" problems from the code related to parallelization or low-level algorithm optimization highly dependent on the computational architecture. This is consistent with recent approaches proposed for implementation of next generation Earth system models, see [18], for multi- and many-core heterogeneous HPC systems. The principal advantage of such separation of concerns is ability to tune the code for different architectures without modifying the high-level and problem specific part of the code. The separation of high- and low- level primitives in the DNS -, LES -and RANS -code is based on the C++ template specialization functionality and, in particular, is used for the implementation of code on GPU architecture within the CUDA programming framework.

3. Numerical Model Setup

In what follows we consider the stably stratified turbulent plane Couette flow as a computational benchmark. The flow, while highly idealized, may be seen as model problem for studying the dynamics of thermally stratified sheared fluid in the surface layer of the atmosphere [13], where the total momentum and heat fluxes are approximately constant with height. The experiment setup corresponds to the one used in [7, 16, 29]. The velocity $\mathbf{u}(\mathbf{x}, t)$ and temperature $\Theta(\mathbf{x}, t)$ are prescribed on the upper and lower walls of the channel: $u = -U_0/2$, $\Theta = \Theta_1$, $v = w = 0$ for $z = 0$ and $u = U_0/2$, $\Theta = \Theta_2$, $v = w = 0$ for $z = H$, where the stable stratification is imposed by setting $\Theta_2 > \Theta_1$, U_0 is the relative wall velocity and H is the channel height. The Dirichlet boundary conditions are applied for the scalar concentrations $C_k(\mathbf{x}, t)$ at the walls. In the horizontal directions periodic boundary conditions are used. The flow is characterized by the dimensionless Reynolds number $\mathrm{Re} = U_0 H/\nu$, Richardson number $\mathrm{Ri} = g\alpha(\Theta_2 - \Theta_1)H/U_0^2$, Prandtl number $\mathrm{Pr} = \nu/\chi$ and the Schmidt numbers $\mathrm{Sc}_k = \nu/\chi_k$. Figure 1 shows the characteristic S-shaped vertical distribution of the mean tracer concentration $C(z)$ and streamwise velocity component $U(z)$, averaged in horizontal homogeneous directions and time, and the instantaneous passive tracer concentration at the center of the channel, $z = H/2$, obtained in the high resolution simulations [16] of neutrally stratified Couette flow for $\mathrm{Re} = 8 \times 10^4$ with the DNS code used in this study.

To measure CPU performance of the code, we fix the streamwise and spanwise sizes of the channel in the numerical experiments as $L_x = 6H$ and $L_y = 4H$, respectively, and consider

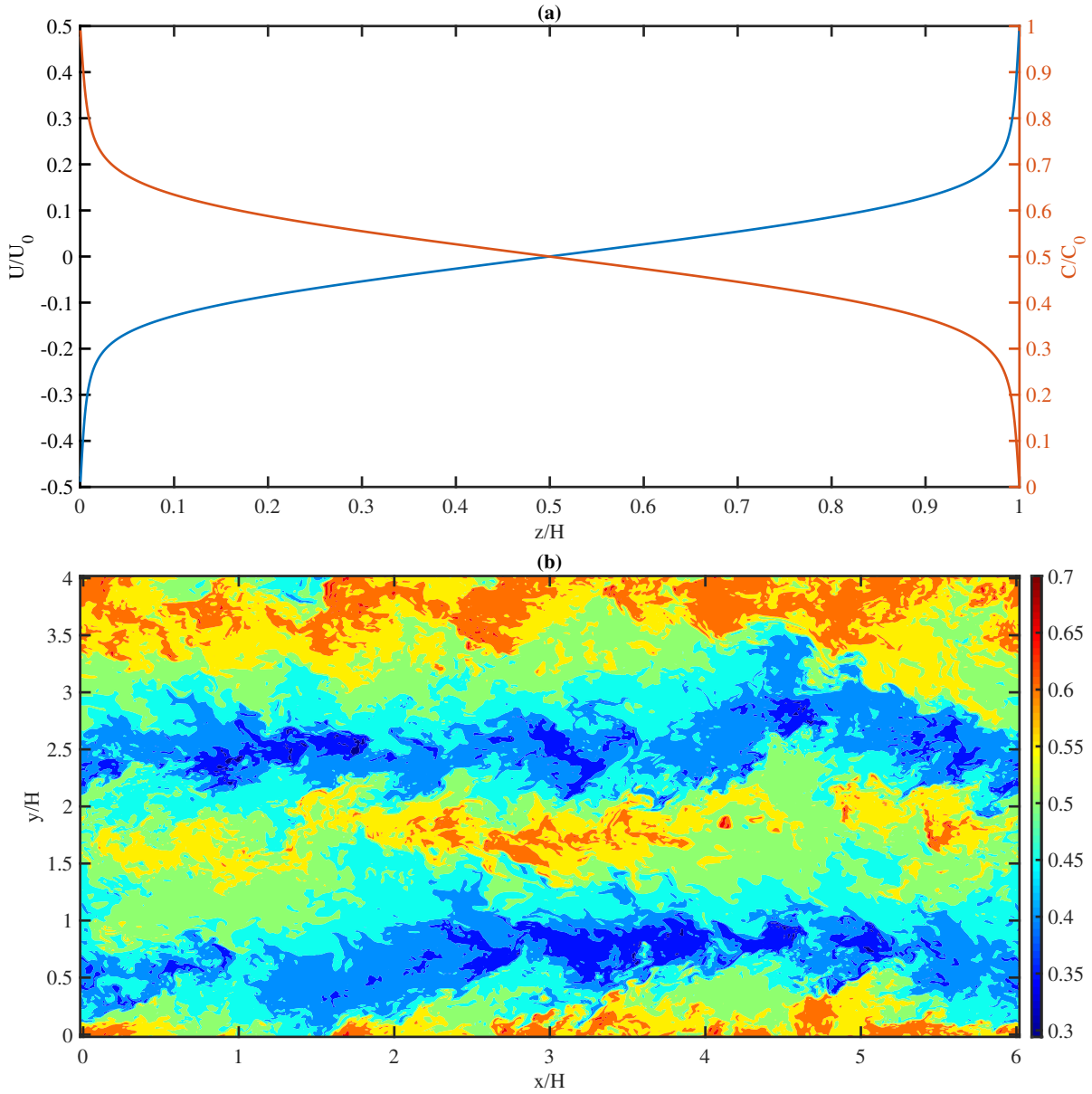


Figure 1. (a) The normalized mean streamwise velocity (blue) and normalized tracer concentration profiles (red); (b) instantaneous normalized tracer concentration in the $-xy$ plane at $z/H = 0.5$ for neutrally stratified turbulent Couette flow

two grid resolutions: *low* resolution model configuration with grid dimensions $(n_x, n_y, n_z) = (96, 64, 64)$ and *high* resolution configuration $(n_x, n_y, n_z) = (192, 128, 128)$ (denoted hereafter as LR and HR in tables and figures), where n_x, n_y and n_z are the number of grid cells in horizontal $-x$, $-y$ and vertical $-z$ directions. The grid step in the vertical direction is refined near the walls according to transformation $z = H/2(1 + \tanh(\eta)/\tanh(\eta_0))$, where $|\eta| < \eta_0$, and the spatial discretization is second-order accurate. In the horizontal directions we use second-order accurate, as well as fourth-order accurate spatial discretizations. The Reynolds number is kept fixed $\mathrm{Re} = 5200$, the bulk Richardson number is set as $\mathrm{Ri} = 0.01$ to maintain the influence of stable stratification on the flow and the Prandtl and Schmidt numbers are set to 0.7. The grid resolution allows to appropriately resolve all necessary spatial scales of the flow in the core of the channel and near the walls for this particular set of parameters. On the other hand the grid

size chosen for the LR and HR cases matches the expected characteristic number of grid cells per computational node ratio in large-scale simulations. Time step is chosen to maintain same CFL number of around 0.1 between all experiments. In addition we set the number of tracer species $n_c = 10$ to mimic reduced acid-base atmospheric chemistry models [28].

For the case of fourth-order accurate method we use only second-order accurate 7-point stencil approximation of the finite-difference Laplace operator in the geometric multigrid iterations to implicitly define the BiCGstab preconditioner. This only mildly affects the overall convergence of the iterative method even for high-resolution simulations, but considerably decreases the MPI communications resulting from parallel implementation of Gauss-Seidel/SOR algorithms. As the computational grid is anisotropic due to smaller grid steps in the vertical direction and the near-wall refinement, we use the approach proposed in [10] for coarsening in the multigrid algorithm.

Additionally we do not consider the possible tuning of the multigrid preconditioner MPI implementation for specific computational architectures and instead use the same parameters obtained by improving the convergence rate alone. The multigrid method performs successive coarsening of the computational grid to reduce the error for a given approximation of solution to the linear system. When the grid is coarsened in the multigrid V cycle and the computational task size on some MPI-process becomes smaller than a prescribed threshold, then the MPI-process is excluded from computations and its domain is merged with one of the neighbouring MPI-processes. This in general improves the performance as for sufficiently small problem the increase in the number of parallel processes leads to increase of the computational time due to communication and synchronization overheads. On the upper branch of the V cycle (consecutive grid refinement starting from the coarsest resolution) an inverse process is implemented – the data from MPI-process is scattered to neighbouring inactive processes when local grid size becomes large enough to benefit from using additional parallel tasks. It is evident that the performance of the mulitgrid algorithm on specific CPU architecture may depend on the depth of grid coarsening (as is the convergence of the method), threshold values defining sufficiently "small" problem size and the number of smoothing iterations. Thus we assume that the contribution of the multigrid method corresponds to a "worst case scenario" and the scaling of the iterative method for the solution of Poisson equation may at least be not optimal. In general, we stress that the runtime share of the iterative method for solution of Poisson equation may vary with the grid resolution, multigrid algorithm parameters and the Re and Ri particular values.

The discussed setup allows us to directly assess the performance of the numerical methods and algorithms for solving the system of equations (1)–(4). We measure the runtime for different components of the algorithm: projection method (eqs. (5), (6) and (7)), Poisson equation (7) solved within the projection method, the heat equation (3) and the tracer transport equations (4). The performance metrics were gathered for $10H/U_0$ dimensionless time units which corresponds to 3200 time steps in HR case and 1600 time steps in the LR case. This is done in order to get sufficient statistical data and to diminish model initialization latency in the results.

4. Code Compilation and Tuning

For performance comparison it is desirable that the code allows to use extended instruction sets available on each CPU architecture. The different code components were thoroughly checked to at least allow compiler autovectorization for each server with specific compiler suite available. In particular, it turned out that some components were not autovectorized, when using the GCC compiler suite. The high potential for code runtime reduction by using vectorized instruction

sets was evident by the results for simple linear algebra math and stencil kernel tests and also for the DNS numerical model on all the CPU systems considered.

To improve autovectorization of the code, we focused on implementation of the projection method, eqs. (5) and (6), and scalar transport, eqs. (3) and (4). This included simplifying memory access patterns in loops and modifying implementation of OpenMP thread decomposition, which in some instances limited compiler optimizations. The subsequent code tuning to allow autovectorization resulted in overall reduction of total computational time over 20% for LR and 25% for HR cases, with the most significant improvements detected for the heat/tracer transport equations – around 50%. While this optimization improved the code performance on Intel Xeon Gold and AMD Rome platforms, it showed even better speed up on the ARM-based Kunpeng 920, especially for the solution of passive transport equations.

To ease comparison of CPU platforms we only use the results obtained with the latest versions of the GCC compiler suite and OpenMPI library available on each server, i.e. GCC 9.3 with OpenMPI 4.0 on AMD Rome, GCC 10.2 with OpenMPI 4.3 on Intel Xeon Gold, and GCC 10.3 with OpenMPI 4.3 on Kunpeng 920 CPUs. The use of Intel compiler suites for the tests on Intel Xeon Gold CPU showed only slightly worse/better performance depending on the code setup.

While we use the same code without any further fine tuning for any specific architecture, the compiler keys were optimized to give the best performance where possible. The default GCC keys supplied with the DNS code were used on the ADM Rome and Intel Xeon Gold systems:

```
mpicxx -c -fopenmp -O3 -fno-strict-aliasing -funroll-loops -march=native
↪  -std=c++0x
```

The porting of the DNS model to the ARM-based 2x48 Kunpeng 920-4826 Taishan server was straightforward and did not require any specific changes to the code. The default GCC compiler keys were supplemented with a general ARM v.8.2-a target key `-march=armv8.2-a`, as GCC provided `-march=armv8.2-a+crypto+fp16` as native target. For optimization additional extensions `+rdma` and `+lse` were considered. Those provide Round Double Multiply Accumulate (RDMA) and Large System Extension (LSE) instructions respectively. LSE key is meant to optimize atomic instructions by replacing the old styled exclusive load-store using a single CAS (compare-and-swap) or SWP (for exchange) and are known to inherently increase performance of applications using atomics. We also enabled `-fomit-frame-pointer` key, which avoids the instructions to save, set up and restore the frame pointer; on many targets it also makes an extra register available. The resulting optimized compiler key set for the Kunpeng 920 platform is:

```
mpicxx -c -fopenmp -O3 -fno-strict-aliasing -funroll-loops
↪  -march=armv8.2-a+rdma+lse -std=c++0x
```

For coarse grid size, in LR case, this optimization yields 25% reduction in total time on average between the number of cores used. This is achieved by faster execution time of projection method and time advancement of heat/tracer transport code. With the grid refinement the acceleration due to optimized keys levels on average at 13% and seems to not affect the heat/tracer transport equation subroutines. Speedup appears to change little when the number of cores is increased.

5. Comparison and Discussion

We run the turbulent Couette flow benchmark with setup presented in the previous sections on three different platforms: 2x64 AMD Rome 7H12 (2.6 GHz) with 256GB DDR4 RAM node of the Mahti CSC supercomputer based on the Atos BullSequana XH2000 system, 2x48 Kunpeng 920-4826 (2.6 GHz) with 512GB DDR4 on TaiShan 200 server and 2x18 Intel Xeon Gold 6140 (2.3 GHz) node with 384GB DDR4. We limit computational tests to the intra-node scalability and run-time analysis.

The benchmark results are presented for the MPI only implementation, except when the full computational node was utilized. In the latter case the results correspond to the MPI-OpenMP hybrid mode of the DNS code with 2 OpenMP threads per MPI process. The OpenMP only implementation showed better scaling compared with MPI only for low core count and with specific thread/core bindings used, e.g., scaling in single NUMA domain on AMD Rome platform. In general the MPI only implementation turned out to be faster than OpenMP only or MPI-OpenMP hybrid and the gap is more pronounced for coarse grids. This is related to thread synchronization overheads, which are as expected more evident for small size problems. On the contrary, when all cores of the two-socket nodes were used, the MPI-OpenMP hybrid implementation showed consistent improvement, in particular, in the multigrid algorithm due to overlapping communications and computational subroutines, and reduced run-time on each of the systems considered. Note that on all nodes the hyperthreading or SMT use was disabled, as it rarely benefits memory bound applications.

The run-time per 1000 time steps for single core tests and when using the maximum number of cores available on each node (2x64 cores on AMD Rome, 2x18 on Intel Xeon Gold and 2x48 on Kunpeng 920) are presented for the LR and HR cases in Tabs. 1 and 2, respectively. Here x2 and x4 denote the second- or the fourth-order finite-difference approximation used in the horizontal directions. For all CPUs the use of higher order scheme results in increase on average of only about 30% in computational time. The difference is even slightly lower for the HR case than for LR coarse grid. The only exception is found for the Kunpeng 920 system, where the maximum run-time increase of about 50% is reached for the HR single core test. The actual number of floating point operations in the algorithm increases by more than twofold for the fourth-order scheme (but still second-order in the vertical direction), e.g., 16 FLOP per grid cell and time step in second-order scheme compared to 38 FLOP when using fourth-order scheme for calculation of diffusion terms. This shows that the implementation of higher-order schemes are much more efficient on all CPUs irrespective of the architecture and in general should be preferred for CFD memory bound applications where it well suits the numerical and physical problem.

The overall three times higher run-time on single core for Kunpeng 920 CPU compared to other two platforms is partially related to its shorter vector instructions length, 128-bit NEON SIMD extension vs. 256/512-bit for AVX2/AVX512 extensions for Intel and AMD processors. However, the difference in run-time between CPUs changes when the full node is utilized with Kunpeng processor outperforming the Intel Xeon Gold node, which alludes to better intra-node scalability on the ARM-based Kunpeng 920 platform.

To delve into more detailed analysis, we look into run-time shares of the model components on different platforms relative to the number of cores used. Figure 2 depicts the corresponding share of the run-time attributed to each code component, where `'proj. method - poisson eq.'` in the figure legend denotes the run-time share of the projection method with the exception of the solution of Poisson equation (7), which is shown as `'poisson eq.'` component. The `'heat eq.'`

Table 1. LR case run-time, in seconds per 1000 time steps

	AMD Rome 7H12	Intel Xeon Gold 6140	Kunpeng 920
single core, x2 and (x4)	39.94 (54.58)	48.53 (59.84)	123.70 (166.28)
max cores, x2 and (x4)	2.16 (2.89)	5.94 (7.94)	2.12 (2.90)

Table 2. HR case run-time, in seconds per 1000 time steps

	AMD Rome 7H12	Intel Xeon Gold 6140	Kunpeng 920
single core, x2 and (x4)	285.23 (372.19)	391.11 (458.09)	1018.81 (1511.12)
max cores, x2 and (x4)	10.23 (13.49)	26.83 (32.81)	18.27 (25.02)

refers to the discrete heat transport eq. (3) and the `'tracers eqs.'` is the run-time share of the time advancement of all n_c tracer transport equations (4). The problem-specific diagnostics in the DNS code constitute the remainder of calculations. Additionally the `'mpi comm'` corresponds to the total run-time share of MPI-communications. Note that the timing of MPI communications is not excluded from other components, so the total sum of all run-time shares of algorithm partition described above is greater than unity.

Figure 3 shows the intra-node scaling of the DNS code and its model components on different platforms attained while increasing the number of CPU cores. The change in the component shares with the increase in the number of cores used is similar for Xeon Gold and Kunpeng processors. The better scaling for scalar transport equations (eqs. (3) and (4)) and worse scaling for Poisson equation (7) solver here leads to decrease of run-time share for the former, while the latter share increases and the BiCGstab algorithm with multigrid preconditioner eventually restricts the overall scaling of the code. For all platforms there appears to be little difference in run-time share of different components between 2-nd order and 4-th order spatial discretizations. In general higher speedup is observed as expected for the HR case compared with coarse LR model configuration for Xeon Gold and Kunpeng CPUs. The intra-node scaling on AMD Rome appears to be worst, especially for high core count used due to lowest ratio of grid cells per core. However, in terms of actual run-time this platform shows the best results, see Tabs. 1 and 2.

The more efficient scaling of the code on Kunpeng 920 CPU appears to be more pronounced for coarse grid LR case in terms of total run-time, probably due to higher cache hit rate. This effect persists for all of the code components, but is more prominent for the tracer transport equations and less significant for projection method algorithm for solving momentum and continuity equations. At the same time, the run-time shares of heat and tracer equations on Kunpeng 920 platform decrease faster with increasing core count rather than for Intel Xeon Gold and, especially, for AMD Rome CPU. This indicates that due to better scaling of those code components Kunpeng 920 hardware is utilized more effectively.

We note that the results of CPU comparison are in line with the simple vector, stencil kernel and MPI bandwidth benchmarks builtin in the DNS model code and used in preliminary tests to assess CPU performance. This showed that the flop/s single core performance on the ARM-based Kunpeng 920 CPU is comparable or outperforms the Intel Xeon Gold processors for relatively small vector sizes and rapidly decreases, when long vectors are considered. The MPI bandwidth ping-pong communication test showed that the Kunpeng 920 CPU outperforms Intel processor for large message size, but the latency of MPI-communications may still affect

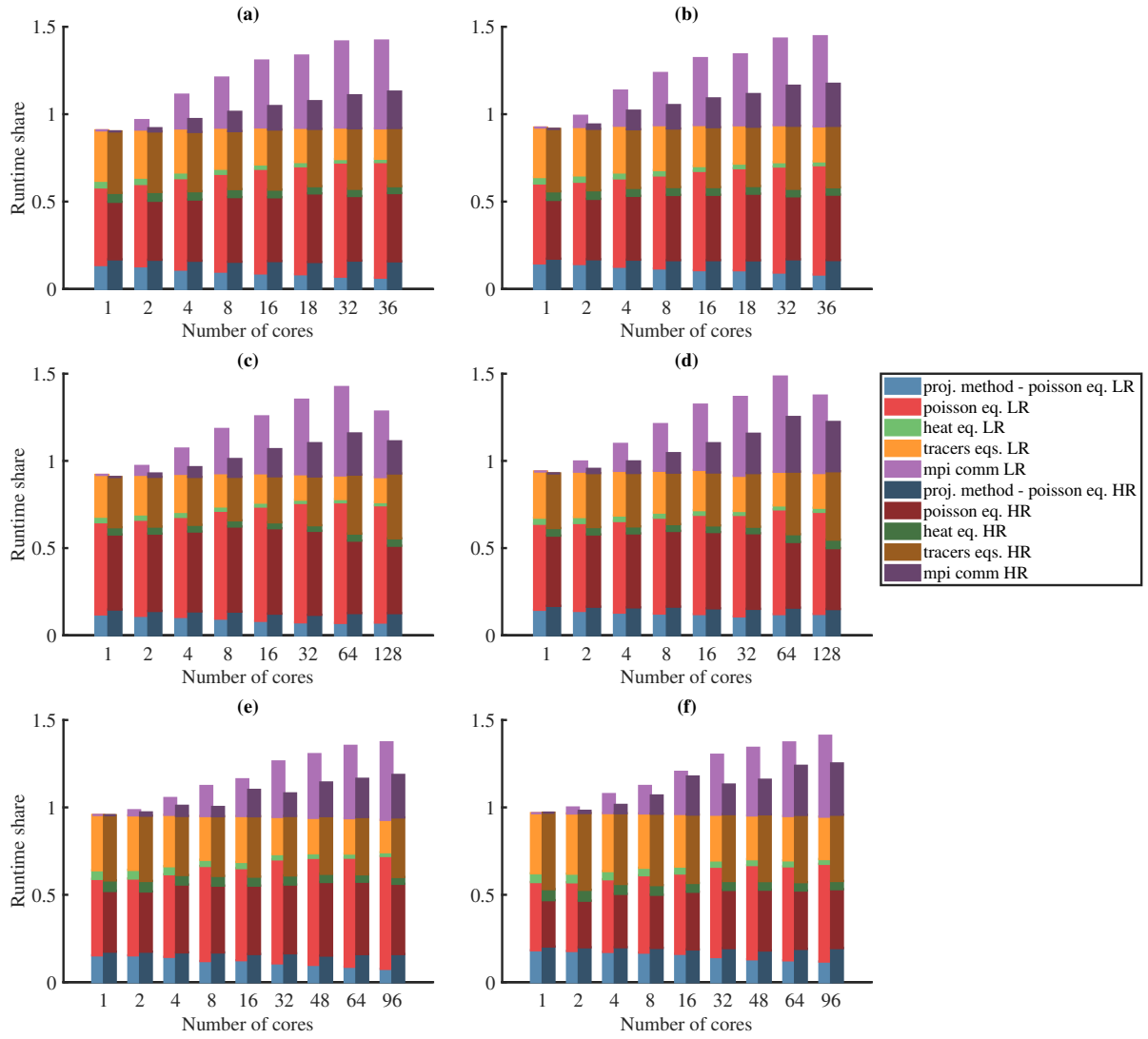


Figure 2. Run-time shares of model components for LR and HR cases on Intel Xeon Gold 1640 (a, b), AMD Rome 7H12 (c, d) and Kunpeng 920 (e, f) CPUs for 2-nd order (a, c, e) and 4-th order accurate finite-difference schemes (b, d, f)

performance, in particular, the implementation of multigrid algorithm, where MPI message size may be small. Our findings are in general also consistent with the performance evaluation of the Kunpeng 920 processor in [1] using specially designed set of benchmarks. This highlights that simple computational benchmarks might be useful to at least make a gross estimate of the expected performance of complex memory- and cache-bound CFD applications on different CPU architectures.

As part of the study we consider potential memory and cache use optimizations for the implementation of tracer transport component of the model. As evident from Fig. 2, the additional 10 passive tracer equations (4) amount for around 30% of total run-time across CPU platforms. This reinforces rationale to investigate the influence of code optimizations on this model component, especially considering that for some CFD applications the number of tracer species n_c may be considerably higher, e.g., atmospheric chemistry.

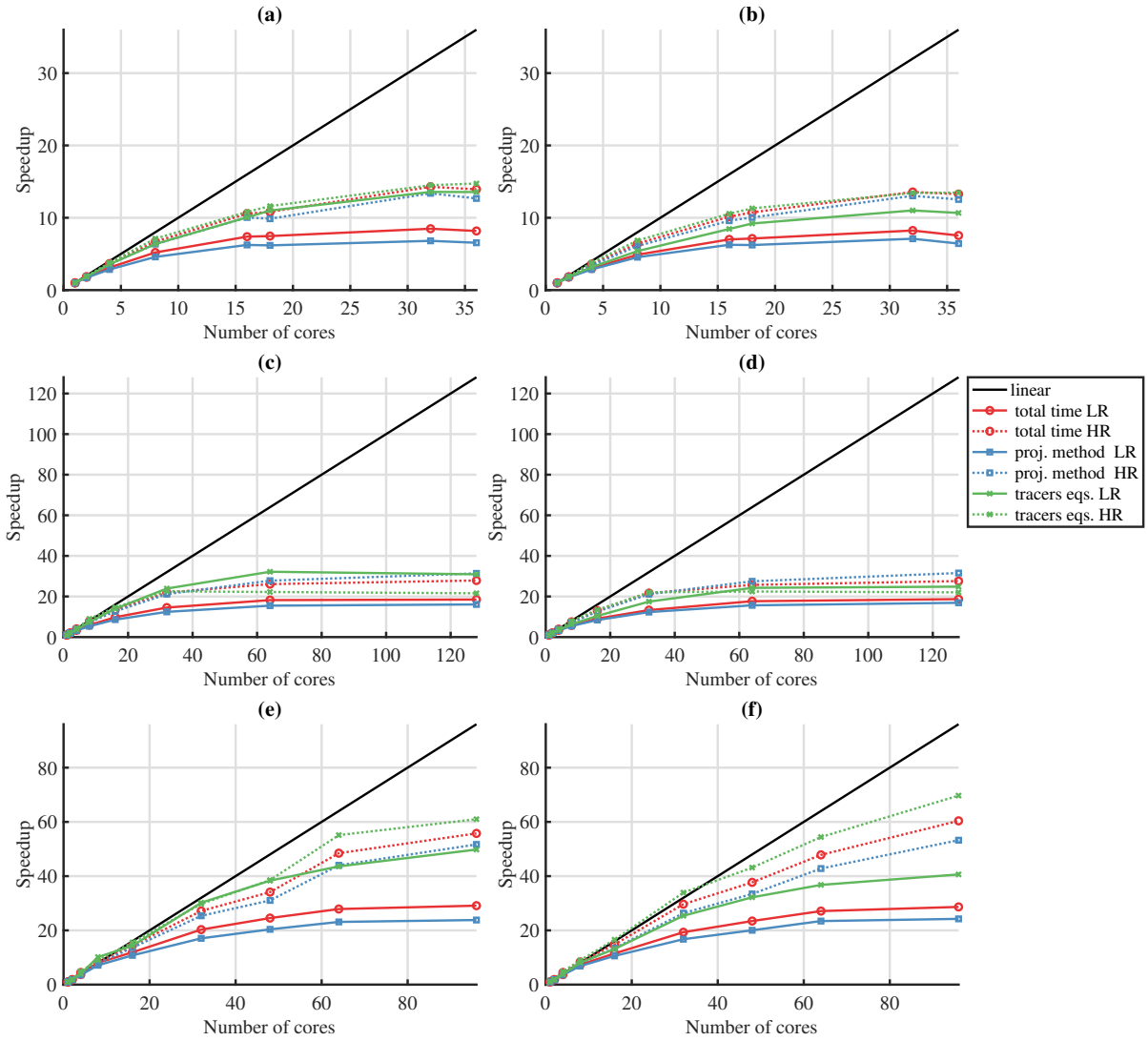


Figure 3. Speedup of DNS model and its components for LR and HR cases on Intel Xeon Gold 1640 (a, b), AMD Rome 7H12 (c, d) and Kunpeng 920 (e, f) CPUs for 2-nd order (a, c, e) and 4-th order accurate finite-difference schemes (b, d, f)

The numerical algorithm for solution of eq. (4) using the explicit in time second-order Adams-Bashforth scheme may be represented by the following computational steps:

$$\mathrm{ADV}^n = -\left[\frac{\partial u_i C_k}{\partial x_i}\right]_h^n, \tag{8.1}$$

$$\mathrm{DIFF}^n = \left[\chi_k \frac{\partial^2 C_k}{\partial x_i \partial x_i},\right]_h^n, \tag{8.2}$$

$$\mathrm{RHS}^n = \frac{3}{2}(\mathrm{ADV} + \mathrm{DIFF})^n - \frac{1}{2}(\mathrm{ADV} + \mathrm{DIFF})^{n-1}, \tag{8.3}$$

$$C_k^{n+1} = C_k^n + \Delta t \mathrm{RHS}^n, \tag{8.4}$$

where $[\cdot]_h^n$ denotes the finite-difference spatial approximation of the term on computational grid using variables on the n-th time step. The algorithm in this form consists of the calculation of the advection (8.1) and diffusion (8.2) terms; computation of tendencies according to time

advancement scheme (8.3) and updating the concentration values C_k for k-th species (8.4) on the next $(n+1)$-th time step.

The implementation, where each of the algorithm steps (8) corresponds to a loop over the 3D computational grid, was used for the CPU performance comparison. While it is expected that this approach is sub-optimal, it represents an upper limit estimate for numerical approaches with explicit synchronization points disallowing loop fusion, e.g., when using implicit in time methods and/or advection/diffusion flux calculation stages. In turn this also allows us to consider the influence of simple optimizations for memory-bound applications based on loop fusion approach, which are known to improve cache reuse. The first such optimization consists of merging calculations of advection (8.1), diffusion (8.2) and tendencies (8.3) parts in a single loop. This represents any potential improvements for models including the species reactions with, as often used, implicit in time discretization. The second optimization involves fusing the entire algorithm in a single loop, estimating the maximal effect such optimizations may yield for fully explicit in time schemes. Finally, we consider the influence of merging MPI point-to-point communications for all n_c species concentrations, which are performed at the end of each time step to fill the halo/ghost cell regions of the local computational grid, on the intra-nodal scaling of the code as it may decrease additional overheads due to communication latency.

Figure 4 shows the results of optimizations described above implemented in the passive tracer transport algorithm for the coarse grid LR case. The run-time for each optimization considered is normalized with the run-time of the default implementation, where the algorithm (8) is separated into four distinct loops. Here `'fusion-adv/diff'` refers to the optimization, where (8.1)–(8.3) are fused, `'fusion'` depicts the case, where loop fusion is applied to all steps of the algorithm (8). The `'fusion + MPI'` refers to the latter optimization coupled with merging of MPI-communications at the end of time step iteration in a single call for all passive tracers. The results for HR case are shown in Fig. 5.

The loop fusion of steps (8.1)–(8.3) results in 23% performance increase on single core of Intel Xeon Gold CPU, and slightly less 15% reduction in run time for the AMD Rome processor. The full fusion of algorithm (8) yields additional 20% and 10% performance gain for Intel and AMD processors, respectively. Increasing the number of cores used diminishes these gains on both platforms as the higher-level cache is used more efficiently and the number of cells per core is reduced. Overall the results show that a significant (up to two-times for grid sizes used in tests) reduction of run-time on Intel and AMD CPUs may be achieved for memory- and cache-aware optimizations. On the contrary, on the Kunpeng 920 platform both optimizations deliver none or only small performance increase. The merging of MPI point-to-point communications for species concentrations provides only marginal improvements in run-time on Intel Xeon Gold, while it may even negatively affect performance on AMD Rome and Kunpeng 920 CPUs, which is especially evident on two-socket tests.

Conclusions

In this study we assessed the efficiency of the implementation of numerical methods used for direct numerical simulation of stratified turbulent flows on a set of CPU architectures commonly used in HPC systems. The DNS model supplemented with passive tracer equations may be seen, at least to some extent, as a proxy for more general RANS and LES models. The stably stratified turbulent plane Couette flow was used as a computational benchmark with two grid resolutions

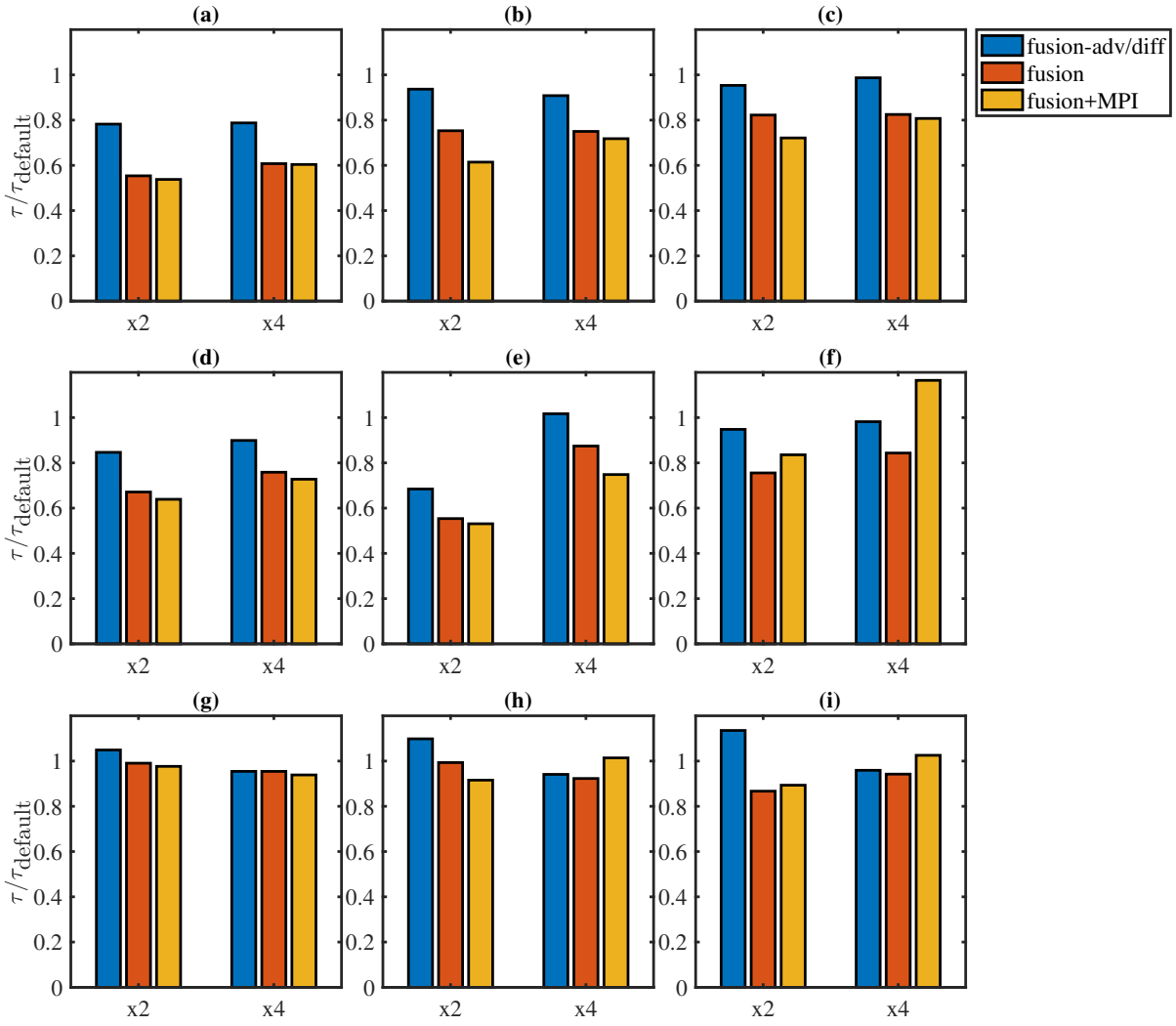


Figure 4. Normalized run-time (relative to default implementation) of the optimized tracer transport model component for LR case on Intel Xeon Gold 1640 (a, b, c), AMD Rome 7H12 (d, e, f) and Kunpeng 920 (g, h, i) CPUs for 2-nd order (`'x2'`) and 4-th order accurate finite-difference schemes (`'x4'`). Tests are performed on a single core (left column), socket (center column) and full node (right column)

matching the characteristic number of grid cells per computational node ratio in large-scale simulations.

We ran the Couette flow benchmark on three different CPU platforms: two with x86 architecture, namely AMD Rome 7H12 and Intel Xeon Gold 6140, and the ARM-based Kunpeng 920-4826. This CPUs differ significantly in core count, SIMD instructions sets, cache size and memory bandwidth. The porting of the DNS code to ARM-based Kunpeng platform was straightforward and did not require any specific code modifications. Nevertheless we fine tuned compiler keys on Kunpeng CPU by enabling RDMA and LSE instruction set extensions (which are not included in the GCC compiler native target provided for this platform). This optimizations led to overall performance increase by 25% and 13% for low (LR) and high grid resolution (HR) cases respectively. We stress the importance of enabling SIMD vector instruction sets in CFD applications irrespective of the CPU architecture. In particular, the GCC autovectorization improved the code performance by over 20% for LR and 25% for HR cases, with the most significant

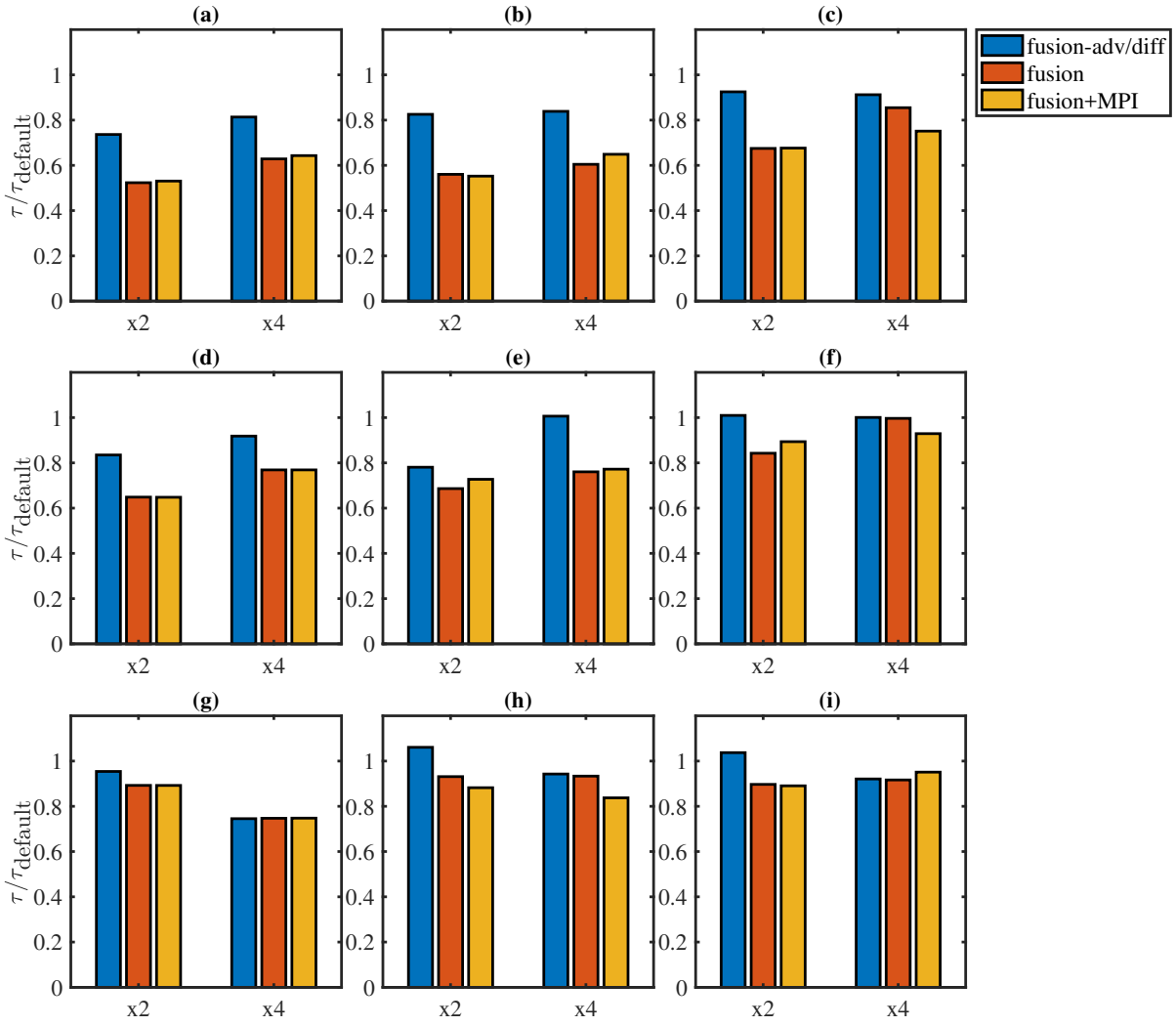


Figure 5. Same as in Fig. 4, but for HR case

improvement found for the numerical solution of heat/tracer transport equations – around 50% reduction of run-time, for all of the platforms.

We compared CPUs by considering intra-node scaling and run-time of the DNS code and its components. We found that the MPI only implementation in general turned out to be faster, compared with OpenMP only or MPI-OpenMP hybrid implementations, in tests where the full node was not utilized with the gap more pronounced for coarse grid case. This is related to thread synchronization overheads, which are especially evident in the multigrid algorithm. For two-socket full node tests the MPI-OpenMP implementation resulted in consistent performance improvement, in part due to overlapping of MPI communications and computational subroutines, and reduced run-time on all processors.

The DNS code exhibits near linear scaling for LR and HR cases with up to 16 cores used on AMD Rome and Kunpeng 920 CPUs, and up to 8 cores used on Intel Xeon Gold. The speedup then drops to half of linear estimate or more when using up to 1/2 number of cores available on each node in coarse resolution simulations. Model scaling improves for the HR case, with the most significant improvement observed for Kunpeng 920 platform. Comparison of the code performance on three different platforms revealed that the Kunpeng CPU underperforms on single core tests, which is consistent with its lower 128-bit length of vector instructions set. The

AMD Rome showed the best single core performance. However, due to more efficient intra-node scaling, on full node tests Kunpeng 920 CPU shows better performance than Intel Xeon Gold by three-fold and reduces the differences in performance with AMD Rome to 50% for the high resolution case, while matching run-time with AMD processor for coarse computational grid.

We emphasize that the Kunpeng 920 allows more effective scaling for the scalar transport equations, especially for smaller grid size simulations. In this regard the ARM-based Kunpeng CPU architecture may be more beneficial for applications involving high number of tracer species, e.g., atmospheric chemistry or ocean biochemistry modeling, where the computational time share of solution of transport equations (4) may be larger than that of solving hydrodynamic equations.

Investigating the influence of the order of spatial discretization on the DNS code performance we found that for all CPU architectures the use of higher-order approximations (fourth-order vs. second-order accurate) results in increase on average of only about 30% in computational time and the difference is lower in HR case than in LR tests. Comparing this to more than two-fold increase in the number of floating point operations for the fourth-order scheme highlights that for CFD memory bound applications the use of higher-order spatial discretizations will lead to more efficient utilization of available CPU resources.

Lastly we explored the impact of memory-oriented optimizations on the Eulerian tracer transport component of the code. We showed that the loop fusion transformation applied for the scalar transport algorithm to improve cache reuse may provide results highly dependent on the CPU platform and the number of grid cells per core ratio. This optimization, which in general may be deemed beneficial, offers negligible effect in the worst case and on the other hand may lead up to 50% improvement in terms of run-time. The improvements are least noticeable on the Kunpeng 920 and are most pronounced on the Intel Xeon Gold CPU, possibly in part due to larger L1/L3 cache size of the former processor, which is also in line with the lesser effect this optimization has in HR case. Another optimization aimed at reducing intra-node communication latency overhead by merging MPI point-to-point communications for different tracer species produced mixed results – slowing the application by 15% in the worst case and improving performance by 12% in the best case scenario.

Acknowledgements

The development of DNS -, LES -, RANS -code was supported by the Russian Science Foundation, grant No. 21-71-30003, research on the efficient implementation of tracer transport for HPC systems was supported by Russian Ministry of Science and Higher Education (agreement No. 075-15-2019-1621). The statistical processing of the DNS results was supported by the RF President grant for young scientists MK-1867.2020.5. The porting and optimization of the DNS code on Kunpeng ARM-platform was supported by Huawei company, agreement No. OAA20100800391587A. Authors acknowledge CSC for providing computational resources on Mahti supercomputer. The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University. Authors would like to thank Victor Stepanenko for fruitful comments and discussion of research presented.

The code of the model used in this study including ported version and auxiliary scripts is available at `http://tesla.parallel.ru` upon request.

References

1. Afanasyev, I., Lichmanov, D.: Evaluating the performance of Kunpeng 920 processors on modern HPC applications. In: Parallel Computing Technologies 2021, Proceedings. pp. 301–321. Springer International Publishing (2021). https://doi.org/10.1007/978-3-030-86359-3_23

2. Ayala, A., Tomov, S., Haidar, A., Dongarra, J.: heFFTe: Highly efficient FFT for exascale. In: Computational Science – ICCS 2020. ICCS 2020. Lecture Notes in Computer Science. pp. 262–275. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-50371-0_19

3. Besnard, J., Malony, A., Shende, S., et al.: An MPI halo-cell implementation for zero-copy abstraction. In: EuroMPI '15: Proceedings of the 22nd European MPI Users' Group Meeting. pp. 1–9. ACM Press (2015). https://doi.org/10.1145/2802658.2802669

4. Brown, D., Cortez, R., Minion, M.: Accurate projection methods for the incompressible Navier-Stokes equations. J. Comp. Phys. 168, 464–499 (2001). https://doi.org/10.1006/jcph.2001.6715

5. Dongarra, J., Heroux, M., Luszczek, P.: A new metric for ranking high performance computing systems. Nat. Sci. Rev. 3(1), 30–35 (2016). https://doi.org/10.1093/nsr/nwv084

6. Gladskikh, D., Stepanenko, V., Mortikov, E.: The effect of the horizontal dimensions of inland water bodies on the thickness of the upper mixed layer. Water Res. 48, 226–234 (2021). https://doi.org/10.1134/S0097807821020068

7. Glazunov, A., Mortikov, E., Barskov, K., et al.: Layered structure of stably stratified turbulent shear flows. Izv., Atmos. Ocean. Phys. 55(4), 312–323 (2019). https://doi.org/10.1134/S0001433819040042

8. Ibeid, H., Olson, L., Gropp, W.: FFT, FMM, and multigrid on the road to exascale: Performance challenges and opportunities. J. Parallel Distrib. Comput. 136, 63–74 (2020). https://doi.org/10.1016/j.jpdc.2019.09.014

9. Kadantsev, E., Mortikov, E., Zilitinkevich, S.: The resistance law for stably stratified atmospheric planetary boundary layers. Q. J. R. Meteorol. Soc. 147(737), 2233–2243 (2021). https://doi.org/10.1002/qj.4019

10. Larsson, J., Lien, F., Yee, E.: Conditional semicoarsening multigrid algorithm for the Poisson equation on anisotropic grids. J. Comp. Phys. 208, 368–383 (2005). https://doi.org/10.1016/j.jcp.2005.02.020

11. LeMone, M., Angevine, W., Bretherton, C., et al.: 100 years of progress in boundary layer meteorology. Meteorological Monographs 59, 9.1–9.85 (2019). https://doi.org/10.1175/AMSMONOGRAPHS-D-18-0013.1

12. Moin, P., Mahesh, K.: Direct numerical simulation: A tool in turbulence research. Annu. Rev. Fluid Mech. 30, 539–578 (1998). https://doi.org/10.1146/annurev.fluid.30.1.539

13. Monin, A., Yaglom, A.: Statistical fluid mechanics: The mechanics of turbulence. MIT Press, Cambridge (1971)

14. Morinishi, Y., Lund, T., Vasilyev, O., Moin, P.: Fully conservative higher order finite difference schemes for incompressible flows. J. Comp. Phys. 143, 90–124 (1998). https://doi.org/10.1006/jcph.1998.5962

15. Mortikov, E.: Numerical simulation of the motion of an ice keel in a stratified flow. Izv., Atmos. Ocean. Phys. 52(1), 108–115 (2016). https://doi.org/10.1134/S0001433816010072

16. Mortikov, E., Glazunov, A., Lykosov, V.: Numerical study of plane Couette flow: turbulence statistics and the structure of pressure-strain correlations. Russ. J. Numer. Analysis Math. Model. 34(2), 119–132 (2019). https://doi.org/10.1515/rnam-2019-0010

17. Pirozzoli, S., Bernardini, M., Orlandi, P.: Turbulence statistics in Couette flow at high reynolds number. J. Fluid Mech. 758, 327–343 (2014). https://doi.org/10.1017/jfm.2014.529

18. Porter, A., Appleyard, J., Ashworth, M., et al.: Portable multi- and many-core performance for finite-difference or finite-element codes – application to the free-surface component of NEMO (NEMOLite2D 1.0). Geosci. Model Dev. 11, 3447–3464 (2018). https://doi.org/10.5194/gmd-2017-150

19. Rajovic, N., Rico, A., Puzovic, N., Adeniyi-Jones, C., Ramirez, A.: Tibidabo: Making the case for an ARM-based HPC system. Future Generation Computer Systems 36, 322–334 (2014). https://doi.org/10.1016/j.future.2013.07.013

20. Sofiev, M., Vira, J., Kouznetsov, R., et al.: Construction of the SILAM Eulerian atmospheric dispersion model based on the advection algorithm of Michael Galperin. Geosci. Model Dev. 8, 3497–3522 (2015). https://doi.org/10.5194/gmd-8-3497-2015

21. Soustova, I., Troitskaya, Y., Gladskikh, D., et al.: A simple description of the turbulent transport in a stratified shear flow as applied to the description of thermohydrodynamics of inland water bodies. Izv., Atmos. Ocean. Phys. 56, 603–612 (2020). https://doi.org/10.1134/S0001433820060109

22. Thorpe, S.: An introduction to ocean turbulence. Cambridge University Press, Cambridge (2007)

23. Tkachenko, E., Debolskiy, A., Mortikov, E.: Intercomparison of subgrid scale models in large-eddy simulation of sunset atmospheric boundary layer turbulence: computational aspects. Lobachevskii Journal of Mathematics 42, 1580–1595 (2021). https://doi.org/10.1134/S1995080221070234

24. Trottenberg, U., Oosterlee, C., Schüller, A.: Multigrid. Academic Press, London (2001)

25. Vasilyev, O.: High order finite difference schemes on non-uniform meshes with good conservation properties. J. Comp. Phys. 157, 746–761 (2000). https://doi.org/10.1006/jcph.1999.6398

26. Vichi, M., Pinardi, N., Masina, S.: A generalized model of pelagic biogeochemistry for the global ocean ecosystem. Part I: theory. J. Mar. Sys. 64, 89–109 (2007). https://doi.org/10.1016/j.jmarsys.2006.03.006

27. Van der Vorst, H.: Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. SIAM J. Sci. Stat. Comput. 13(2), 631–644 (1992). https://doi.org/10.1137/0913035

28. Yli-Juuti, T., Barsanti, K., Hildebrandt Ruiz, L., et al.: Model for acid-base chemistry in nanoparticle growth (MABNAG). Atmos. Chem. Phys. 13(24), 12507–12524 (2013). https://doi.org/10.5194/acp-13-12507-2013

29. Zasko, G., Glazunov, A., Mortikov, E., Nechepurenko, Y.: Large-scale structures in stratified turbulent Couette flow and optimal disturbances. Russ. J. Numer. Analysis Math. Model. 35(1), 37–53 (2020). https://doi.org/10.1515/rnam-2020-0004

30. Zilitinkevich, S., Druzhinin, O., Glazunov, A., et al.: Dissipation rate of turbulent kinetic energy in stably stratified sheared flows. Atmos. Chem. Phys. 19, 2489–2496 (2019). https://doi.org/10.5194/acp-19-2489-2019

31. Zoric, D., Sandborn, V.: Similarity of large reynolds number boundary layers. Bound. Layer-Meteorol. 2, 326–333 (1972). https://doi.org/10.1007/BF02184773

DOI: 10.14529/jsfi210406

Scalability as a Key Property of Mapping Computational Tasks to Supercomputer Architecture

Alexander S. Antonov[1]

When solving complex computational problems on modern supercomputers, an increasingly important role is played by the scalability property, which characterizes the ability of applications to adapt to various degrees of parallelism of computing systems.

Keywords: scalability, supercomputer, AlgoWiki, parallel structure, problems, methods, algorithms, implementations, computing platforms.

1. Description of the Algorithms Properties in the AlgoWiki Encyclopedia

The project of the AlgoWiki Open Encyclopedia [1] implements on the Internet the possibilities for uniting the efforts of the computing community to form a bank of descriptions of the computational algorithms properties according to a single universal scheme. Within the framework of this scheme, two parts of the description are distinguished – the first describes the properties of the algorithms themselves, and the second describes the properties of software implementations and the dynamic characteristics of their execution on specific computing systems [2].

Algorithm descriptions are the most important, but not the only part of the AlgoWiki Open Encyclopedia. Within the framework of the encyclopedia, hierarchical descriptions are built in the form of chains "problem–method–algorithm–implementation–computer" [3], corresponding to the scheme actively used in practice for solving problems on high-performance computing systems of various architectures. Currently, work is underway to close such chains with descriptions of supercomputers in the framework of the Algo500 project [4].

2. Scalability

The scalability property is one of the most studied properties of parallel programs [5]. Different authors interpret this concept in different ways. In the most general approach [6], *scalability* is understood as a property of a parallel program that characterizes the dependence of the entire set of dynamic characteristics of this program on the set of its launch parameters.

From the dynamic characteristics of programs in practice, the achieved value of performance is considered most often, and from the launch parameters – the number of parallel processes and the computational complexity of the problem being solved. Taking these three values together it will determine the most commonly used types of scalability. So, most often in practice, the dependence of performance on the number of application processes is considered, fixing the size of the task (computational complexity). This dependence is commonly referred to as *a strong scalability*. In many cases, unless a special caveat is made, it is strong scalability that is meant.

Many real-life problems are designed in such a way that their computational complexity can be regulated by setting a few simple parameters. If so, then it makes sense to simultaneously increase both the number of computational processes and the computational complexity of the

[1]Lomonosov Moscow State University, Moscow, Russian Federation

problem. If the system performance continues to grow, then the program is said to have *weak scalability*. In addition to strong and weak, some other types of scalability are also known, but it is these two types that are encountered most often in practice.

One of the well-known scalability metrics is the isoefficiency function [7], which shows the required level of growth in the computational complexity of a problem to maintain a given level of efficiency (understood as the ratio of acceleration to the number of processes used). In the framework of the AlgoWiki Open Encyclopedia, another version of the scalability metric of algorithm implementations has been proposed, based on empirical data.

3. Scalability Obstacles

Various obstacles that hinder the achievement of high scalability rates on various high-performance computing systems are of interest for study. It is important that scalability interference can occur at all stages of the mapping of the problem being solved to the target supercomputer, which we propose to fix using the mechanisms of the AlgoWiki encyclopedia. The reasons for poor scalability may lie in the structure of the problem itself, in the chosen method for solving it, in a specific algorithm, its software implementation, as well as on the side of the computer's software and hardware environment.

3.1. Algorithm-Level Obstacles

Sometimes the algorithm itself is designed in such a way that no implementation of it will effectively scale to the available computing system. This can be determined, for example, by a small number of operations of the considered algorithm or by the presence of true information dependencies between the available operations. Ultimately, this leads to the fact that the use of the available parallelism resource is justified only for a relatively small number of computational processes. When scaling to large configurations, the computational load on each process becomes unreasonably small, and the gain from parallelization is lost against the background of additional overhead costs. In this case, they say that the limit of decomposition of the computational domain has been reached, and no additional efforts will give a fundamentally better scalability without changing the algorithmic basis itself, which should be reflected in the description of the algorithm.

3.2. Obstacles at the Software Implementation Level

In some cases, the considered algorithm has a sufficiently large parallelism resource to achieve good scalability, but the situation is spoiled by an ineffective software implementation.

So, in many algorithms that work with matrices of a special type (for example, triangular), the parallelism resource is sufficient to scale to large computing systems. However, when using parallelism of only the outer loop using a static distribution (Fig. 1), the operations of the algorithm will be extremely unevenly distributed among the processes. Some processes will get significantly more iterations of the inner loop than others, and such uneven load will lead to downtime of some processes, which will significantly affect scalability in general.

In other cases, scalability can be affected, for example, by poor choice of functionality provided by a particular parallel programming technology. For example, the overuse of blocking communications in MPI can lead to downtime waiting for data or even deadlocks. Figure 2 shows

```
#pragma omp for schedule (static)
for(i=0; i<N; i++)
   for(j=i; j<N; j++)
      a[i][j]=...
```

Figure 1. Parallelizing a fragment of triangular matrix computations

a typical example of an inefficient implementation of transfers from all communicator processes to process number `0` in a program using MPI technology. This fragment implements a strict order of receiving data in the order of increasing process numbers in the communicator. In many cases, it would be more efficient to abandon this ordering by accepting data from ready-made processes, for which you can use the `MPI_ANY_SOURCE` option. This can especially strongly affect the performance of large configurations of computing systems when the rest of the program is executed asynchronously.

```
if(rank != 0)
   MPI_Send(&a, 1, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD);
else
   for(int i = 1; i < size; i++)
      MPI_Recv(&a, 1, MPI_DOUBLE, i, 1, MPI_COMM_WORLD, &status);
```

Figure 2. Sending data from all processes to one (MPI technology)

Excessive process synchronization also often affects efficiency and scalability. Too many spawned parallel threads can add a significant amount of overhead. Using unsuccessful communication schemes can result in additional wasted time. There can be a lot of such reasons; it is planned to devote a separate study to their analysis.

3.3. Obstacles at the System Software Stack Level

Choosing the right software and configuring it efficiently is also critical to achieving a high level of scalability. Here it is important to correctly configure not only compilers and user libraries, but also, for example, the resource manager used on the computing system – the efficiency of its execution can greatly depend on how it distributes the processes of a user program among computational nodes. Other system settings inaccessible to an ordinary user are also important, for example, routing settings in a communication network. Using different routing algorithms for a large number of processes or for long messages may result in better scalability of applications.

3.4. Obstacles at the Hardware and Environmental Level

In some cases, the very choice of the target supercomputer may be unfortunate in terms of achieving a high level of scalability. For example, the computational structure of a fragment can be effectively implemented on an existing supercomputer, but the amount of RAM and the required speed of working with it for large tasks may be insufficient. In such cases, it is necessary to formulate the requirements for the target computing system, taking into account the tasks to be solved, using the principles of supercomputer code design [8].

Good scalability can be hindered by the use of quite suitable computing systems. Thus, most supercomputers are used in a shared access mode. Most often, computing nodes in such systems are exclusively allocated for the task, but a number of important system resources (such as a communication network, storage system) are shared by all users of the system. Therefore, any currently running task can affect the scalability of our application by organizing heavy traffic over the network or actively engaging in data I/O.

4. AlgoWiki Encyclopedia and Scalability

The description of the scalability properties in the Open Encyclopedia AlgoWiki is given in the section 2.4 of the unified algorithms description. Until now, a scheme for describing the scalability properties of the algorithm itself has not been proposed, therefore, in most descriptions of algorithms, special attention is paid to the scalability of the algorithm implementation. For this, experimental data are obtained from runs of the existing implementation of the algorithm on some high-performance computing system. The key point of this section is to show the real scalability parameters of the implementation of this algorithm on various computing platforms, depending on the number of processors and the size of the task. In this case, it is important to choose such a ratio between the number of processors and the size of the task in order to reflect all characteristic points in the behavior of a parallel program, in particular, the achievement of maximum performance, as well as subtle effects arising, for example, due to the block structure of the algorithm or the memory hierarchy.

It is proposed to implement an integrated approach to describing scalability in the Open Encyclopedia AlgoWiki, reflecting at all stages of the chains "problem–method–algorithm–implementation–computer" all possible obstacles for the effective adaptation of the properties of computational problems to an increase in the degree of parallelism of computing systems.

Conclusion

This article discusses the issue of studying the scalability property, which is one of the key properties characterizing the quality of parallel programs execution. The idea of expanding the description of the algorithms properties within the framework of the AlgoWiki project is proposed, which allows to present the most complete approach to the study of potential problems with scalability by analyzing all steps in solving problems. Possible obstacles to achieving good scalability can be observed at any stage of the mapping of the problems being solved to the architecture of the target supercomputers. A detailed study and description of such obstacles by the user community within the AlgoWiki framework will help to avoid them, achieving a high level of scalability of parallel applications.

Acknowledgements

Described results were obtained at Lomonosov Moscow State University with the financial support of the Russian Science Foundation, agreement № 20–11–20194. The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University [9].

References

1. Open Encyclopedia of Parallel Algorithmic Features. http://algowiki-project.org/en, accessed: 2021-11-22

2. Antonov, A., Voevodin, Vad., Voevodin, Vl., Teplov, A.: A Study of the Dynamic Characteristics of Software Implementation as an Essential Part for a Universal Description of Algorithm Properties. 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing Proceedings, February 17-19, 2016. pp. 359–363. IEEE Computer Society (2016). http://dx.doi.org/10.1109/PDP.2016.24

3. Antonov, A., Frolov, A., Konshin, I., Voevodin, Vl.: Hierarchical Domain Representation in the AlgoWiki Encyclopedia: From Problems to Implementations. Communications in Computer and Information Science, vol. 910, pp. 3–15. Springer (2018). http://dx.doi.org/10.1007/978-3-319-99673-8_1

4. Antonov, A., Nikitenko, D., Voevodin, Vl.: Algo500 – a New Approach to the Joint Analysis of Algorithms and Computers. Lobachevskii Journal of Mathematics 41(8), 1435–1443 (2020). http://dx.doi.org/10.1134/S1995080220080041

5. Scalability. In: Padua, D. (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA (2011). https://doi.org/10.1007/978-0-387-09766-4_2046

6. Antonov, A., Teplov, A.: Generalized approach to scalability analysis of parallel applications. Algorithms and Architectures for Parallel Processing - ICA3PP 2016 Collocated Workshops: SCDT, TAPEMS, BigTrust, UCER, DLMCS, Granada, Spain, December 14-16, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10049, pp. 291–304. Springer (2016). http://dx.doi.org/10.1007/978-3-319-49956-7_23

7. Grama, A.Y., Gupta, A., Kumar, V.: Isoefficiency: measuring the scalability of parallel algorithms and architectures. IEEE Parallel Distrib. Technol. 1(3), 12–21 (1993). https://doi.org/10.1109/88.242438

8. Dosanjh, S.S., Barrett, R.F., Doerfler, D.W., et al.: Exascale design space exploration and co-design. Future Generation Computer Systems 30, 46–58 (2014). http://dx.doi.org/10.1016/j.future.2013.04.018

9. Voevodin, Vl., Antonov, A., Nikitenko, D., et al.: Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community. Supercomputing Frontiers and Innovations 6(2), 4–11 (2019). http://dx.doi.org/10.14529/jsfi190201

DOI: 10.14529/jsfi210407

High-performance Shallow Water Model for Use on Massively Parallel and Heterogeneous Computing Systems

Andrey V. Chaplygin[1], *Anatoly V. Gusev*[1,2,3], *Nikolay A. Diansky*[1,3,4]

This paper presents the shallow water model, formulated from the ocean general circulation sigma model INMOM (Institute of Numerical Mathematics Ocean Model). The shallow water model is based on software architecture, which separates the physics-related code from parallel implementation features, thereby simplifying the model's support and development. As an improvement of the two-dimensional domain decomposition method, we present the blocked-based decomposition proposing load-balanced and cache-friendly calculations on CPUs. We propose various hybrid parallel programming patterns in the shallow water model for effective calculation on massively parallel and heterogeneous computing systems and evaluate their scaling performances on the Lomonosov-2 supercomputer. We demonstrate that performance per a single grid point on GPUs dramatically decreases for small grid sizes starting from 2^{19} points per node, while performance on CPUs scales up to 2^{17} well. Although, calculations on GPUs outperform calculations on CPUs by a factor of 4.7 at 30 nodes using 60 GPUs and 360 CPU cores at 6100×4460 grid size. We demonstrate that overlapping kernel execution with data transfers on GPUs increases performance by 28%. Furthermore, we demonstrate the advantage of using the load-balancing method in the Azov Sea model on CPUs and GPUs.

Keywords: shallow water, supercomputer modeling, heterogeneous computing systems, MPI, OpenMP, CUDA.

Introduction

The current intensive development of climate models, particularly the ocean general circulation models, is associated primarily with the rapid development of computer technology. The emergence of teraflop and petaflop computing systems opened up the possibility of designing ocean models of high spatial resolution, which allows to describe meso- and submesoscales of eddy variability in the scope of long-term simulations. Today, most high-performance computing systems are heterogeneous, combining various computing processors, clearly seen from the TOP 500 list of most powerful supercomputers in the world. Such systems can generally consist of a large number of processors of various types. Nowadays, the main direction of developing heterogeneous systems is the joint use of multi-core Central Processing Units (CPUs) and massively parallel accelerators, such as Graphics Processing Units (GPUs). Supercomputer technology is rapidly developing in Russia, and the development trend is similar to the world one – this can be seen from the TOP 50 list of most powerful supercomputers in the Commonwealth of Independent States (CIS). The most powerful supercomputers of Russia, for example, "Lomonosov-2" at Lomonosov Moscow State University, are heterogeneous computing systems. Creating a model that effectively uses the resources of such heterogeneous computing systems is a complex and relevant problem nowadays [1, 13]. The variety of computing systems leads to more complex parallel programming patterns and challenges porting software for efficient usage.

The shallow water equation set is a key component in ocean general circulation models, which is difficult enough to resolve. This system of equations in ocean models is obtained for

[1]Marchuk Institute of Numerical Mathematics of the Russian Academy of Sciences, Moscow, Russia
[2]P.P. Shirshov Institute of Oceanology of the Russian Academy of Sciences, Moscow, Russia
[3]N.N. Zubov State Oceanographic Institute, Moscow, Russia
[4]Lomonosov Moscow State University, Moscow, Russia

barotropic adaptation by vertically integrating three-dimensional momentum and continuity equations. Due to the high speed of the external gravity waves, the solution of the barotropic adaptation in ocean models performs with a time step smaller by one or two orders of magnitude than the time step for solving three-dimensional equations [2]. Therefore the demanding time for solving the shallow water equation set is a significant part of the total time spent for the complete equation set of ocean hydrothermodynamics. We take the system of shallow water equations as our starting point for evaluating various parallel methods and approaches useful for ocean models. This paper considers the system of shallow water equations in the form presented in the ocean general circulation sigma model INMOM (Institute of Numerical Mathematics Ocean Model). The INMOM model is being developed at the INM RAS (Marchuk Institute of Numerical Mathematics of the Russian Academy Sciences). For more than a decade and a half, the model has been used as the oceanic block of the climate model INMCM (Institute of Numerical Mathematical Climate Model). This coupled model is so far the only representative from Russia at various stages of the international project for comparing climate models CMIP (Coupled Model Intercomparison Project), conducted under the auspices of IPCC (Intergovernmental Panel on Climate Change) [10]. The model is completely written in the Fortran 90/95 programming language. The shallow water model has been formulated that can be used both as a program block of the sigma ocean model INMOM and independently, for example, to calculate tsunami waves, tides, and wind surges [4, 6].

This paper presents a new software architecture of the shallow water model, based on the separation of concerns, which involves using various hybrid parallel programming patterns. Software architecture separates the physics-related code of the model from features of parallel implementation, thereby simplifying the support and development of the entire software package. Our approach is influenced by the PSyKAl approach, which is also based on the separation of concerns [3]. In contrast with the PSyKAl approach, our software architecture preserves the original structure of loops in computational kernels at the lowest software architecture level and adds loops over block data structures at the intermediate parallel level. That allows us to implement the following approaches:

- load-balancing and cache-friendly calculation on CPUs;
- utilizing various parallel approaches on GPUs such as overlapping kernel execution with data transfer, load-balancing, and effective calculation on computing nodes with multiple GPUs per node.

Many atmospheric and oceanic general circulation models use the uniform domain decomposition method as the baseline of parallel implementation [7, 16]. However, due to land in most ocean areas, a block-based decomposition using non-uniform partitions and load-balancing methods is more efficient. There are several implementations of block-based decomposition in ocean models. For example, the parallel version of the finite element model of the Arctic Ocean (FEMAO) uses the logical mask of wet points marking computational points for each CPU core and uses data arrays shared by all blocks [17]. In contrast, Parallel Ocean Program (POP) [20] and High Resolution Operational Model for the Baltic Sea (HIROMB) [24] allocate separate data arrays for each block and organize computations in blocks, which is more cache-friendly on CPUs. We present a block-based decomposition implemented using derived data types containing blocks that are allocated separately and distribute data among processing units. A load-balancing method using the Hilbert space-filling curves is also presented. Our novelty is an organization of computations in blocks both on CPUs and GPUs.

We present various hybrid parallel programming patterns for use on massively parallel and heterogeneous computing systems. A pure MPI and hybrid MPI-OpenMP are presented as calculation patterns on CPUs. Hybrid approaches for calculations on CPUs have recently become increasingly relevant and are used in many hydrodynamic models [1, 8, 21]. Our model uses the task-based hybrid MPI-OpenMP approach, which is more efficient compared to the widespread vector-based hybrid MPI-OpenMP approach in ocean models [6]. General-purpose computing on GPUs is becoming increasingly popular for climate modeling too. There are examples of successful porting of atmosphere and ocean models on GPUs, including shallow water models [12, 18, 25]. We present three hybrid parallel programming patterns for calculations on GPUs: hybrid MPI-CUDA, hybrid asynchronous MPI-OpenMP-CUDA, and multi GPUs per node MPI-OpenMP-CUDA calculation patterns for effective use on heterogeneous computing systems. Our novelty is using block-based decomposition on GPUs to achieve overlapping kernel execution with data transfer, calculations with load-balancing, and effective calculation on computing nodes with multiple GPUs per node. The code for GPUs is written using native CUDA Fortran syntax instead of CUDA C common in ocean modeling.

1. Description of the Shallow Water Model

The model considered in this paper is based on a system of nonlinear shallow water equations, which is written in an arbitrary orthogonal coordinate system in the following form:

$$\begin{gathered}\frac{\partial r_x r_y h u}{\partial t} + T_u(u,v) - F_u(u,v) - h r_x r_y l v + r_y h g \frac{\partial \zeta}{\partial x} = RHS_u, \\ \frac{\partial r_x r_y h v}{\partial t} + T_v(u,v) - F_v(u,v) + h r_x r_y l u + r_x h g \frac{\partial \zeta}{\partial y} = RHS_v, \\ \frac{\partial h}{\partial t} + \frac{1}{r_x r_y}\left(\frac{\partial u r_y h}{\partial x} + \frac{\partial v r_x h}{\partial y}\right) = 0,\end{gathered} \tag{1}$$

where r_x, r_y are the Lamé metric coefficients that arise while writing a system of equations in an arbitrary orthogonal coordinate system; u, v are the components of the depth-averaged horizontal velocity vector; l is the Coriolis parameter; g is the free-fall acceleration; ζ is a deviation of the sea surface height from its undisturbed state; $h = H + \zeta$ is the total depth of the ocean; H is the depth of the ocean at a rest state describing bottom topography.

The transport operators T_u, T_v are written in the divergent form:

$$\begin{aligned}T_u(u,v,h) &= \frac{\partial h r_y u u}{\partial x} + \frac{\partial h r_x v u}{\partial y} - h\left(v\frac{\partial r_y}{\partial x} - u\frac{\partial r_x}{\partial y}\right)v, \\ T_v(u,v,h) &= \frac{\partial h r_y u v}{\partial x} + \frac{\partial h r_x v v}{\partial y} + h\left(v\frac{\partial r_y}{\partial x} - u\frac{\partial r_x}{\partial y}\right)u.\end{aligned} \tag{2}$$

The viscosity operators F_u, F_v are written as the divergence of the stress tensor:

$$\begin{aligned}F_u(u,v) &= \frac{1}{r_y}\frac{\partial}{\partial x}\left(r_y^2 K D_T h\right) + \frac{1}{r_x}\frac{\partial}{\partial y}\left(r_x^2 K D_S h\right), \\ F_v(u,v) &= -\frac{1}{r_x}\frac{\partial}{\partial y}\left(r_x^2 K D_T h\right) + \frac{1}{r_y}\frac{\partial}{\partial x}\left(r_y^2 K D_S h\right),\end{aligned} \tag{3}$$

where K is the viscosity coefficient, and D_T and D_S are tension and shear components of the stress tensor:

$$D_T = \frac{r_y}{r_x}\frac{\partial}{\partial x}\left(\frac{u}{r_y}\right) - \frac{r_x}{r_y}\frac{\partial}{\partial y}\left(\frac{v}{r_x}\right),$$
$$D_S = \frac{r_x}{r_y}\frac{\partial}{\partial y}\left(\frac{u}{r_x}\right) + \frac{r_y}{r_x}\frac{\partial}{\partial x}\left(\frac{v}{r_y}\right). \quad (4)$$

The gradients of atmospheric pressure and wind friction stress are generally calculated at the right-hand side of equations RHS_u, RHS_v. At the solid boundary, no normal flow and free slip are set as velocity boundary conditions.

In the form (1)–(4), the nonlinear shallow water equations are presented in the ocean general circulation sigma model INMOM as vertically integrated momentum and continuity equations in order to resolve fast barotropic gravitational waves at a separate stage with minimum computational efforts [9]. The system of equations (1)–(4) is solved using numerical methods on the traditional Arakawa 'C' structured grid. The second-order numerical schemes on the structured grid and the explicit first-order 'leapfrog' scheme are used as spatial and temporal discretization schemes in the model, respectively. Due to the explicit time scheme, our computational method is matrix-free. More details about the form of writing a system of nonlinear shallow water equations and their numerical implementation can be found in papers [4, 9]. The shallow water model simulates extreme surges in the Azov Sea well: numerical results match ocean model results and actual observations [11]. Also, the model simulates the 2011 tsunami in Japan, which led to the Fukushima disaster, and numerical results match observations [5].

2. Software Architecture

A new software architecture for the shallow water model has been developed based on the separation of concerns. This software architecture divides program code into three layers. The lowest layer contains all subroutines required to calculate nonlinear shallow water equations, so-called computational model kernels. The highest layer is responsible for calling computational kernels and describes the time cycle of the model at a relatively high level without the knowledge of parallel data structures and parallel methods used in the model. The intermediate layer between the first two is responsible for parallel methods and approaches used in the model. That separation allows flexibly configuring the model for various computing systems without changing physics-related parts of the program.

A three-layer software architecture based on the separation of concerns has proven itself in the shallow water model of the ocean model NEMO (Nucleus for European Modeling of the Ocean) [3]. The researchers plan to implement such software architecture in the whole ocean model NEMO by 2022 [15].

As mentioned before, the shallow water model is completely written in the Fortran language introducing its specifics into the software architecture. Modules, derived types (classes), interfaces, and macros are extensively used in the code. We remind that the code for GPUs is written using native CUDA Fortran syntax. We describe each program layer of the model.

2.1. Kernel Layer

The Kernel layer contains all computational subroutines, so-called model kernels. The original non-parallelized program is a set of exactly such subroutines, which are the model's baseline. In total, there are about 15 model kernels in the shallow water model. The model kernel is a sub-

routine that consists of grid variables calculations without data synchronization inside. It means that if the subroutine has several loops over grid points and data synchronizations between them, it must be split into several subroutines (model kernels) without data synchronization inside. The Interface layer is responsible for data synchronization between model kernel calls and will be discussed later.

In the case of using CPUs for computing, the model kernel is a two-dimensional loop over grid points that updates values at grid points by numerical schemes. At this level, we work with ordinary two-dimensional arrays. Figure 1a shows a general view of the model kernel for calculation on CPUs, where nx_start, nx_end, ny_start, ny_end are boundaries of the subdomain; bnd_x1, bnd_x2, bnd_y1, bnd_y2 are boundaries of the subdomain including halo points; var is a grid variable.

```
subroutine kernel(var)
  real(wp8), intent(inout) :: var(bnd_x1:bnd_x2, bnd_y1:bnd_y2)
  [...]
  do m = nx_start, nx_end
    do n = ny_start, ny_end
      var(m, n) = [...]
    enddo
  enddo
end subroutine
```

(a) Code for CPUs

```
attributes(global) subroutine kernel(var)
  real(wp8), intent(inout) :: var(bnd_x1:bnd_x2, bnd_y1:bnd_y2)
  [...]
  m = (blockIdx%x-1)*blockDim%x + threadIdx%x + (nx_start - 1) - 1
  n = (blockIdx%y-1)*blockDim%y + threadIdx%y + (ny_start - 1) - 1
  if (m <= nx_end + 1 .and. n <= ny_end + 1) then
    var(m, n) = [...]
  endif
end subroutine
```

(b) Code for GPUs

Figure 1. The Kernel layer of the shallow water model

In the case of using GPUs for computing, the model kernel is updating values at a single grid point instead of the usual two-dimensional loop over grid points for calculation on CPUs. Each point in the grid corresponds to its thread on GPU. The model kernel launching on GPU is performed by threads that cover all grid points. Figure 1b shows a general view of the model kernel for calculation on GPUs.

In that way, 15 model kernels have been implemented for calculations on both CPUs and GPUs.

2.2. Algorithm Layer

The Algorithm layer establishes the order of calling model kernels and is the highest software architecture layer. The main time cycle of the model is described at this level. At this level, we

work with abstract data structures, such as the ocean_type class, which includes the main grid variables of shallow water equations. Figure 2 shows a code example of this layer. In this example, the sea level calculation kernel (kernel_ssh) is called first, then the velocity component calculation kernel (kernel_uv). Special envoke procedure, part of the Interface, calls model kernels and will be discussed later.

```
[...]
type(ocean_type), target :: ocean_data
procedure(empty_kernel), pointer :: kernel_ssh, kernel_uv
[...]
call envoke(ocean_data%ssh, kernel_ssh)
call envoke(ocean_data%u, ocean_data%v, kernel_uv)
[...]
```

Figure 2. The Algorithm layer of the shallow water model

2.3. Interface Layer

The intermediate layer between the model kernel and the model kernel call is the Interface layer at which parallel methods and approaches are implemented. In particular, the block-based decomposition (Section 3.1), the load-balancing method (Section 3.1), the hybrid approach using MPI and OpenMP technologies (Section 3.2), hybrid approaches using MPI, OpenMP and CUDA technologies (Section 3.3) are implemented at the Interface layer. This layer also includes code for processors synchronization, halo swaps, data transfers between CPU and GPU. The Kernel layer and the Algorithm layer contain a description of physical processes in the shallow water model. At the same time, nothing is known at these layers about features of parallel implementation hiding from them in the Interface layer. The Interface layer allows configuring the model to any target computing system flexibly. At this layer, we work with specific parallel data types, for example, the data 2D_real 8_type class, which contains distributed data across blocks and processors. In the case of using GPUs for computing, parallel data types contain symmetrical data in GPU and procedures for synchronizing this data between CPU and GPU.

Figure 3 shows the interface implementation for calculating model kernels on CPUs using a single envoke subroutine. This subroutine calls the model kernel for each data block and then synchronizes OpenMP threads and MPI processes (see Section 3.2 for more information about this approach).

Figure 3 shows the interface implementation for calculating model kernels on GPUs using a single envoke subroutine. This subroutine calls the model kernel for each data block corresponding to its GPU using a special CUDA syntax. The subroutine performs data synchronization between CPU and GPU. OpenMP threads and MPI processes are synchronized too (see Section 3.3 for more information about this approach).

3. Parallel Methods and Approaches

The following parallel methods and approaches have been implemented in the shallow water model based on the described software architecture: the block-based decomposition with load-balancing; the hybrid approach using MPI and OpenMP; hybrid approaches using MPI,

```
subroutine envoke(var, kernel)
  type(data2D_real8_type) :: var
  [...]
  !$omp do private(k) schedule(static, 1)
  do k = 1, blocks
    call kernel(var%block(k)%host)
  enddo
  !$omp end do nowait
  call sync_halo
end subroutine
```

(a) Interface for CPUs

```
subroutine envoke(var, kernel)
  type(data2D_real8_type) :: var
  [...]
  !$omp do private(k) schedule(static, 1)
  do k = 1, blocks
    cudaSetDevice(k-1)
    call kernel<<<tGrid, tBlock>>> (var%block(k)%device)
  enddo
  !$omp end do nowait
  call copy_device_to_host
  call sync_halo
  call copy_host_to_device
end subroutine
```

(b) Interface for GPUs

Figure 3. The Interface layer of the shallow water model

OpenMP and CUDA for computing on heterogeneous computing systems. Further, we describe each parallel method and approach in the model.

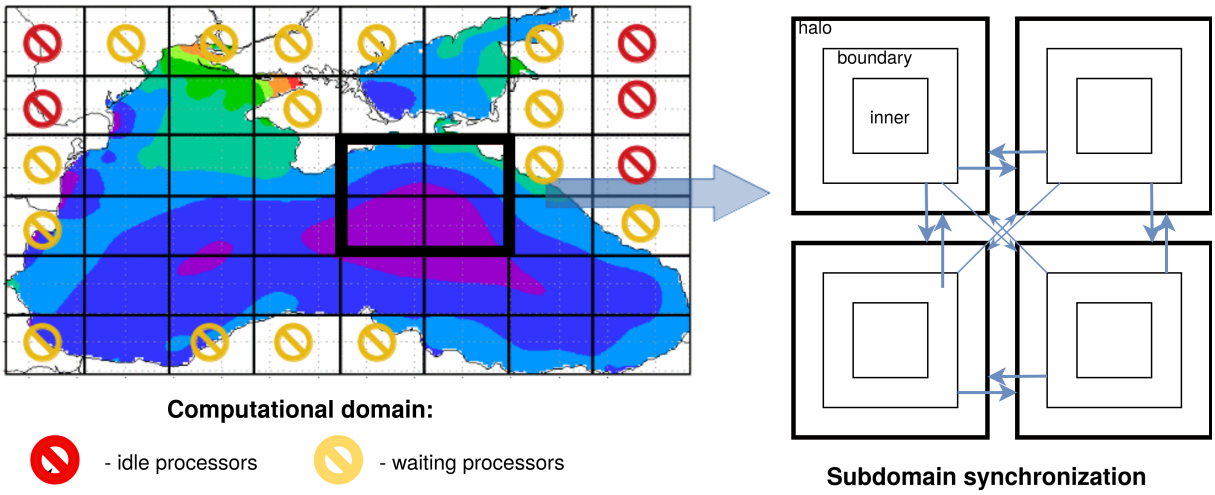


Figure 4. The uniform domain decomposition method and synchronization of processors

3.1. Block-based Decomposition and Load-balancing Method

The shallow water model uses a two-dimensional domain decomposition method as the primary parallelization method. The initial computational domain is divided into subdomains, and each processor is assigned its subdomain. Each subdomain has extra boundaries with halo points which exchanges boundary data with a neighboring subdomain. Processors are synchronized, and halo points are updated using MPI technology before each subdomain calculation. The most common and easily implemented domain decomposition method is the method of uniform subdivision into rectangular subdomains. Figure 4 demonstrates this method and the synchronization mechanism of processors with halo points updating. The figure also shows that the uniform domain decomposition method leads to workload imbalances among participating processors due to land points. Some subdomains contain only land points; some subdomains are more than half-filled with land points. Thus, some processors are idle in computing leading to performance losses. Due to islands and coasts in most ocean areas, load-balancing is an exceptionally urgent task for ocean models, particularly shallow water models [4].

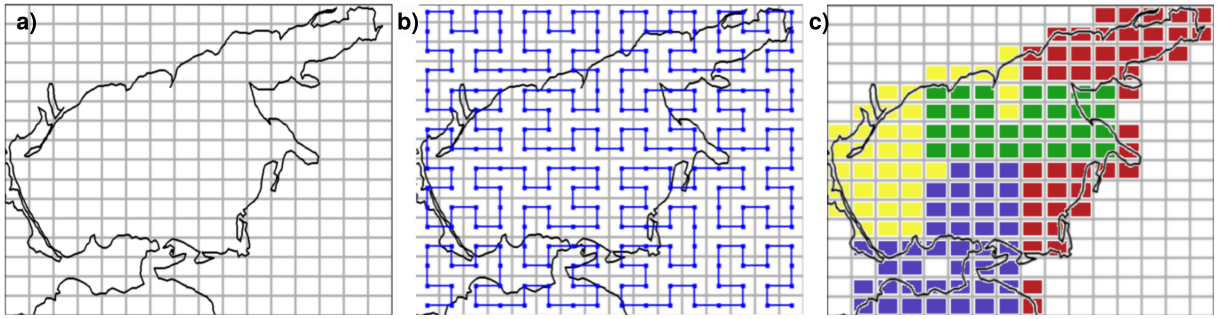


Figure 5. Load-balancing method: a) The domain is uniformly divided into blocks; b) The Hilbert curve is formed; c) Subdomains are formed along the Hilbert curve

Therefore, an improved method of subdivision into subdomains, so-called block-based decomposition, has been implemented in the model. This method uniformly divides the computational domain into rectangular blocks of small size. Then, subdomains are formed along the Hilbert space-filling curves to have approximately the same amount of workload. Land blocks are excluded from subdomains and further calculations. Each processor is assigned a certain number of blocks, which form its computational subdomain. Figure 5 clearly shows the steps of the described algorithm. All calculations in the model are organized in blocks, and each block has extra boundaries with halo points, as shown in Fig. 6. Block halo points are updated before each calculation. If a neighbor block is located on the same processor, we copy boundary values without calling the MPI library. For halo points exchanges with blocks on other processors, we use asynchronous MPI calls.

The block-based decomposition has another advantage in addition to the load-balanced distribution. It is efficient memory management while computing on CPUs, and one can get a performance increase on CPUs due to better cache behavior of smaller blocks. For hydrodynamic models, this property is essential because most of them are strongly memory-bound, so memory management is critical to the model's efficiency and performance scaling [22].

The shallow water model employs the block-based decomposition for calculations on GPUs as well as CPUs. Calculations on GPUs are also organized in blocks, and synchronizations occur in the same way as it is done for calculations on CPUs, with the addition of halo points transfer

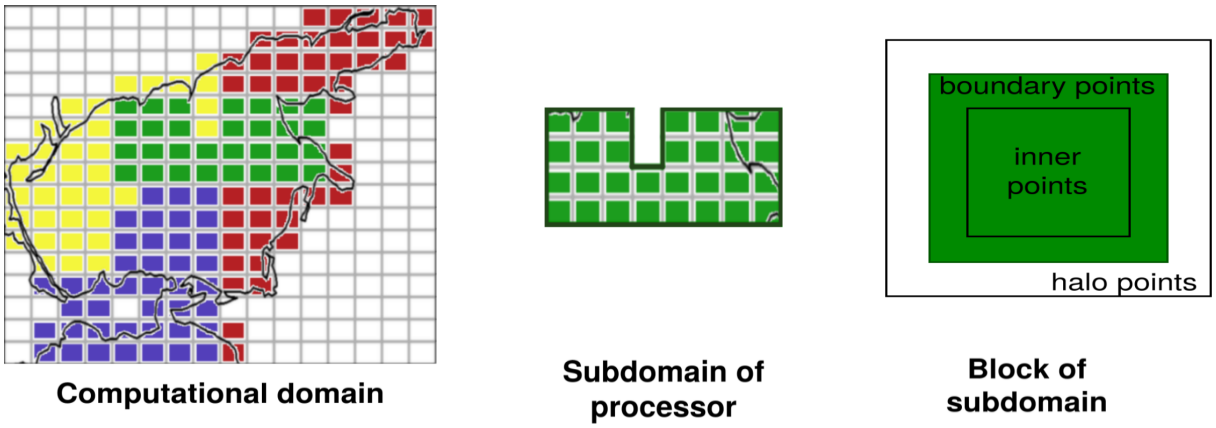


Figure 6. The block-based decomposition

between CPU and GPU. The block-based decomposition allows organizing asynchronous data transfer between CPU-GPU and overlapping calculations with memory copying and communications, leading in turn to improved performance on GPUs (see Section 3.3).

However, the block-based decomposition has a disadvantage: overheads for copying block boundary values during synchronization. Previous work [6] showed that effective work with cache memory compensates for copying overheads during synchronization for calculation on CPUs with small blocks. Therefore this disadvantage can be considered insignificant. This disadvantage becomes significant for calculations on GPUs since copying block boundary values must be carried out between CPU and GPU. Nevertheless, as it will be shown in Section 4, it is possible to improve performance using the block-based decomposition on GPUs for large computational areas due to the overlapping computations with synchronizations.

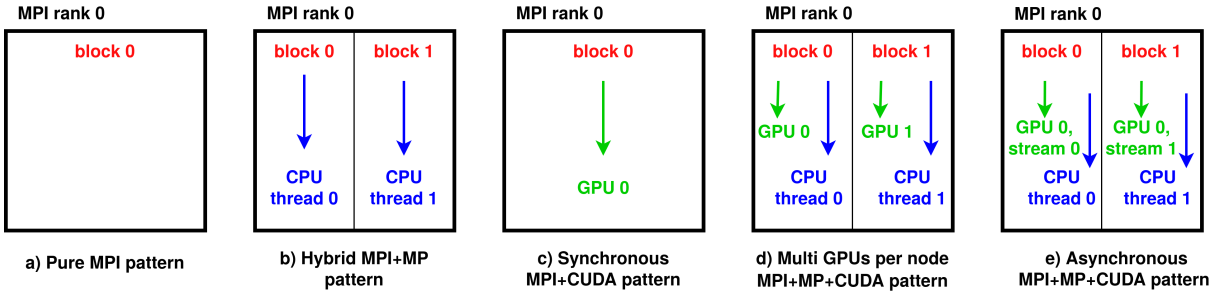


Figure 7. Parallel programming patterns implemented in the shallow water model

3.2. Hybrid Approach Using MPI and OpenMP for Use on Multiprocessor Computing Systems

Data synchronization between processors is a bottleneck in the shallow water model on multiprocessor computing systems. With an increase in the number of computing nodes, synchronization overheads increase due to the high load on the communication network. It is possible to reduce the load on the communication network using OpenMP for parallelization on shared memory inside a node. It is a so-called hybrid approach. The pure MPI approach creates a

separate MPI process for each core on a node, whereas the hybrid MPI + OpenMP approach creates only one MPI process for each node and separate threads for each core.

In the shallow water model, the task-based hybrid MPI-OpenMP approach has been implemented, which distributes subdomains (blocks from block-based decomposition) across OpenMP threads as shown in Fig. 7 compared to the pure MPI approach. Blocks are first distributed across MPI processes using the load-balancing method. Then, blocks are distributed across available threads within the MPI process, ensuring a uniform computational workload per thread. The previous work [6] has showed that this approach has the advantage compared to the widespread vector-based MPI-OpenMP hybrid approach, in which OpenMP is used only for parallelizing two-dimensional loops over subdomains. The performance of the implemented task-based hybrid approach is twice as high as the vector-based hybrid approach when calculating the model on multiple computing nodes. The previous work has also showed the advantage in performance of the task-based hybrid approach over the pure MPI approach.

3.3. Hybrid Approaches Using MPI, OpenMP and CUDA for Use on Heterogeneous Computing Systems

In the shallow water model, calculation on GPUs has been fully supported using CUDA technology. For this purpose, we have adapted 15 model kernels to calculations on GPUs. We have modified the Interface layer of the software architecture to utilize various calculation patterns on GPUs in the model. As mentioned earlier, the model kernel is a single grid point calculation on GPUs by a single thread, and it is launched by threads entirely covering the computational domain. It is necessary to note two essential details in model kernels implementation for calculation on GPUs. The first is that double precision is used everywhere in calculations, and, second is the lack of memory optimizations. The last means that no specific data placement optimizations on GPUs are implemented in model kernels; in particular, shared and texture memories are not used. All memory accesses in model kernels occur immediately to the GPU's global memory. Modern generations of GPUs are less sensitive compared to the older ones to data placement optimization, mostly due to improvements of global memory caches, as shown in [14]. The authors of this paper have considered various benchmark applications, including computational fluid dynamics solver, and showed that using different memory optimizations on modern generations of GPUs (Pascal, Volta) overall does not produce as much speedup as older ones. However, these results should be considered only as part of an overall picture. Furthermore, we have implemented model kernels for calculations on GPUs with minimal effort, and adaptation on GPUs can be further automated using macros, as done in work [3].

The implementation on GPUs has been adapted to support the block-based decomposition in the shallow water model, which has proven itself on CPUs. Due to the block-based decomposition, it is possible to balance a workload of computations on GPUs. Three parallel programming patterns have been implemented for calculations on GPUs: the synchronous MPI-CUDA pattern, the asynchronous MPI-OpenMP-CUDA pattern, the multi GPUs per node MPI-OpenMP-CUDA pattern. Note that all patterns have not been challenging to implement in the software architecture based on the separation of concerns. All code modifications have been implemented at the Interface layer without affecting the Kernel and the Algorithm layers. We describe each of the patterns in detail.

3.3.1. Synchronous MPI-CUDA calculation pattern

In this approach, each MPI process is assigned a subdomain containing only one block and a single GPU, as shown in Fig. 7. The subdomain calculation is performed entirely on GPU using CUDA. After model kernel's calculation on GPUs, processors synchronization occurs as follows:

1. Boundary points of the subdomain are transferred from GPU to CPU synchronously, meaning blocking data transfers are used.
2. MPI processes are synchronized, and halo points of each subdomain are updated. MPI synchronization is performed entirely on the CPU.
3. The updated halo points are transferred back from CPU to GPU. Data transfer is still synchronous.

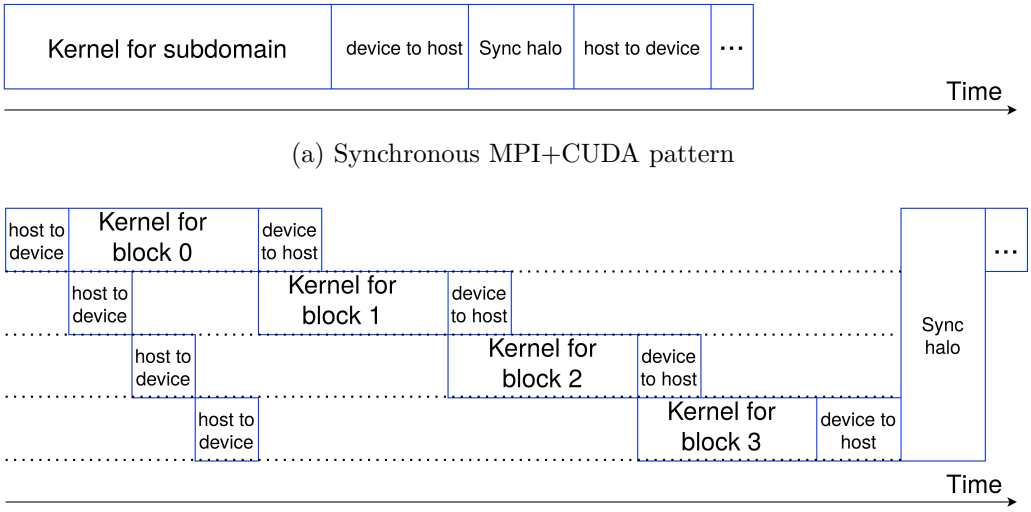
This pattern supports calculations on multiple GPUs assuming only one block per MPI process and does not support the block-based decomposition. Thus, there is no load-balancing of calculations on GPUs using this pattern.

3.3.2. Asynchronous MPI-OpenMP-CUDA calculation pattern

In this pattern, the block-based decomposition is supported for calculations on GPUs. Each MPI process is assigned a subdomain containing multiple blocks, and OpenMP threads and CUDA streams are created for each subdomain's block in the MPI process, as shown in Fig. 7. The model kernel is launched for calculation on GPU for each subdomain's block as follows:

1. Each OpenMP thread asynchronously launches the model kernel on GPU for the subdomain's block. Launching is performed in the CUDA stream corresponding to the OpenMP thread.
2. Each OpenMP thread asynchronously transfers boundary points from GPU to CPU of the subdomain's block. Data transfer is performed in the CUDA stream corresponding to the OpenMP thread.
3. All CUDA streams and OpenMP threads are synchronized.
4. MPI processes are synchronized, and halo points of each subdomain's block are updated. MPI synchronization is performed entirely on the CPU.
5. Each OpenMP thread asynchronously transfers back halo points from CPU to GPU of the subdomain's block. Data transfer is performed in the CUDA stream corresponding to the OpenMP thread.

Since modern GPUs have separate control elements for execution kernels and data transfers, this calculation pattern organizes asynchronous data transfers and overlaps kernel execution on GPU with data transfers between CPU and GPU using different CUDA streams. Figure 8 shows steps of this calculation pattern in comparison with the synchronous MPI-CUDA pattern schematically. It can be seen that data transfers overlap with kernel execution in time for the asynchronous MPI-OpenMP-CUDA pattern. This calculation pattern is completely hybrid and designed for more efficient calculation on computing nodes with one GPU per node compared to the synchronous MPI-CUDA pattern.



(a) Synchronous MPI+CUDA pattern

(b) Asynchronous MPI+OpenMP+CUDA pattern

Figure 8. The synchronous MPI-CUDA and the asynchronous MPI-OpenMP-CUDA patterns

3.3.3. Multi GPUs per node MPI-OpenMP-CUDA calculation pattern

This pattern also supports the block-based decomposition for calculations on GPUs, but differently than the asynchronous MPI-OpenMP-CUDA pattern. Each MPI process is assigned a number of blocks and OpenMP threads as many as available GPUs on a node. Accordingly, every GPU managed by a single OpenMP thread contains a single block subdomain on a node. OpenMP threads independently launch CUDA kernels, allowing efficient use of every GPU available on a node. Synchronizations are organized as in the MPI-CUDA pattern, but with synchronization of GPUs on a node and block boundaries points gathering for synchronization of MPI processes. The approach is completely hybrid and designed to calculate on computing nodes with multiple GPUs per node.

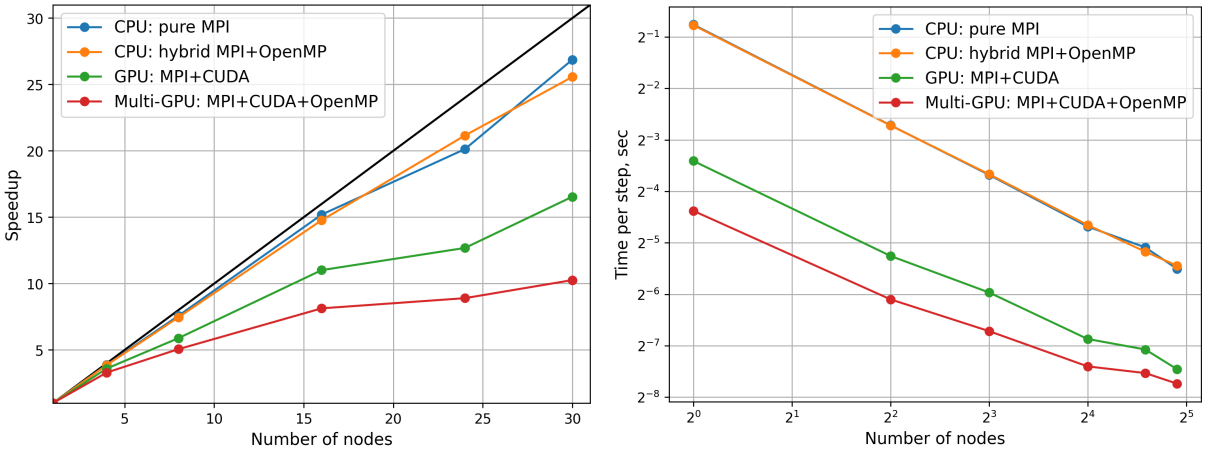
4. Results and Discussions

Our experiments were performed on the Lomonosov-2 supercomputer at Lomonosov Moscow State University [23], which is a completely heterogeneous computing system and is one of the most high-performance supercomputers in Russia today according to the TOP500 list. We performed our numerical experiments on the Pascal and Volta sections of the supercomputer, which contain modern GPUs on nodes. The Pascal section includes 160 nodes with one Intel Xeon Gold 6126 2.60GHz CPU (12 cores) and two Nvidia Tesla P100 GPUs; the Volta section includes 16 nodes with Intel Xeon Gold 6126 2.60GHz CPU (12 cores) and two Nvidia Tesla V100 GPUs. We compiled the software code using the Nvidia HPC Fortran compiler (PGI compiler), which supports native CUDA in Fortran and has recently become free. We used the optimization option -fast and libraries Open MPI v3.1.5, CUDA 11.1 for compilation.

4.1. The Box Test

The first series of experiments were performed for the Box test. This test represents the computational domain without land points and with constant sea depth. We set a gaussian water level elevation as the initial condition, and there were no wind forcing. We made this test

to demonstrate the performance scaling of the shallow water model without workload imbalances on processors. We evaluated the model's performance at different grid sizes of the computational domain. We chose computational grids corresponding to the Azov Sea grids with a resolution of 250 meters and 62.5 meters: 1525 × 1115 points and 6100 × 4460 points. We performed model simulations for one model day, with 86400 model steps in total.



(a) Speedup

(b) Time per model step (in logarithmic scale)

Figure 9. Performance scaling on the Pascal section for the Box test with 6100 × 4460 grid size

Figure 9 demonstrates the performance scaling of various calculation patterns on CPUs and GPUs at 6100 × 4460 grid size. Demonstrated results on the figure were obtained using 30 nodes of the supercomputer's Pascal section. The pure MPI and the hybrid MPI-OpenMP calculation patterns on CPUs demonstrate close to linear (black line on the figure) scaling up to 30 nodes (360 cores in total). However, the hybrid MPI-OpenMP calculation pattern failed to outperform the pure MPI pattern on this supercomputer. We assume that using 30 nodes is not enough to see the advantages of the hybrid calculation pattern over the pure MPI due to the low load on the communication network. The multi GPUs per node MPI-OpenMP-CUDA pattern allows performing calculations on 60 GPUs using 30 nodes and demonstrates the best calculation time due to the efficient use of all computing resources on a node. The multi (two exactly) GPUs per node pattern have twice better performance up to 16 nodes than one GPU per node CUDA-MPI pattern, but then the performance difference decreases due to small subdomain size per GPU. Calculation on a single GPU outperforms any calculation pattern on a single CPU by a factor of 6.3. Although the performance scaling on GPUs is worse than on CPUs, calculations on GPUs still outperform calculations on CPUs by a factor of 4.7 at 30 nodes using 60 GPUs and 360 CPU cores. We also compared the performance scaling on CPUs and GPUs at 1525 × 1115 grid size. Figure 10 shows times per a single grid point for calculations on CPUs and GPUs running 1525 × 1115 and 6100 × 4460 grid sizes. Performance per a single grid point on GPUs dramatically decreases after 2^{19} points per node, while performance on CPUs scales up to 2^{17} well. Accordingly, GPUs are much more sensitive to the grid size than CPUs in two aspects. First, communication overheads, including data transfers between CPU and GPU, can exceed the computation time and become a bottleneck in GPUs' performance. Second, small subdomains lead to better cache behavior for calculation only on CPUs but cannot saturate execution and entirely hide memory latencies on GPUs.

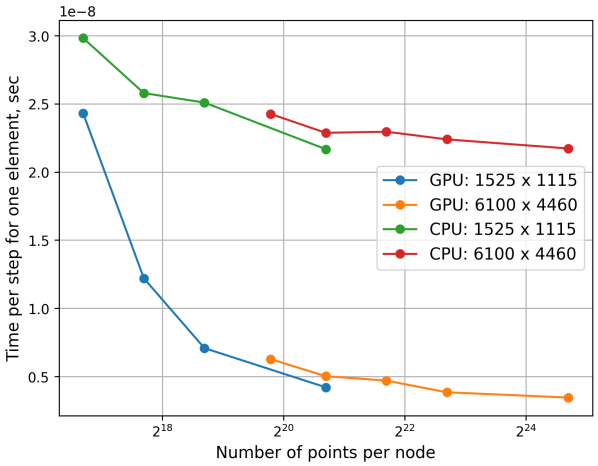


Figure 10. Time per single grid point (in logarithmic scale). The Pascal section of the supercomputer Lomonosov-2 was used

To further investigate performance scaling on GPUs, we tested the asynchronous MPI-OpenMP-CUDA calculation pattern, which overlaps data transfers with kernel execution. Experiments at 6100 × 4460 grid size were performed on the supercomputer's Volta section, including Nvidia Tesla V100 GPUs on nodes. Figure 11 demonstrates performances using a different number of blocks in the block-based decomposition on a single GPU. We see that the asynchronous calculation pattern on GPUs outperforms by 17% the synchronous calculation pattern due to kernel execution and data transfer overlapping. Figure 12 shows performance scaling of the asynchronous calculation pattern on GPUs using an optimal number of blocks (8 blocks per GPU as shown in Fig. 11) compared to the synchronous calculation pattern on GPUs. This experiment shows that the asynchronous pattern is better scaled up to 8 nodes and 28% faster on 8 GPUs than the synchronous calculation pattern. Also, this experiment clearly shows that overlapping kernel execution with data transfer compensates for overheads of copying blocks' boundary values during synchronization in the block-based decomposition on GPUs. However, this statement is valid only for sufficiently large computational domains.

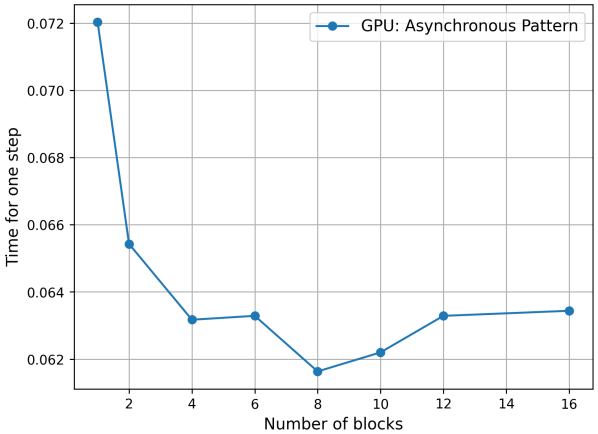
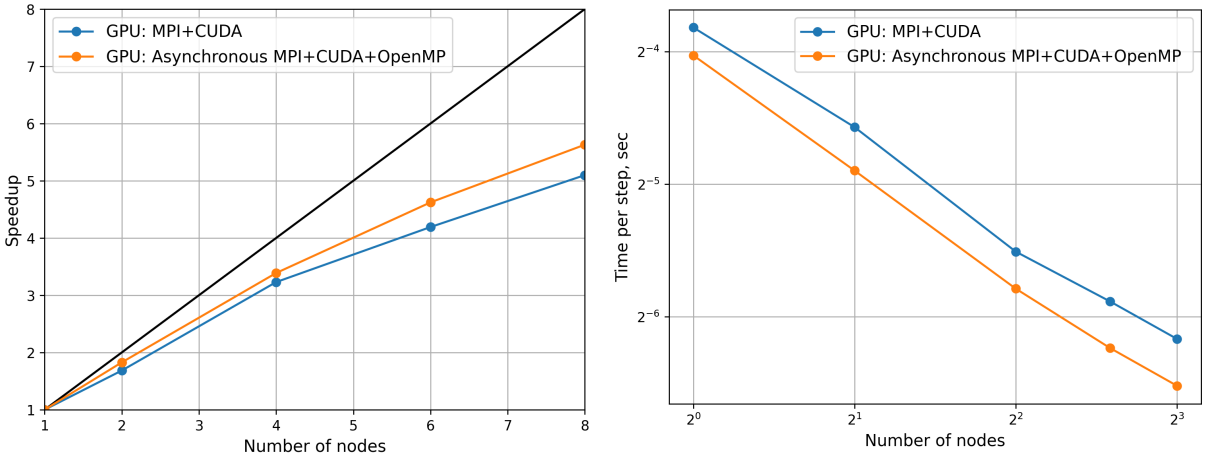


Figure 11. Time per model step (in logarithmic scale) on a single V100 GPU for the Box test with 6100 × 4460 grid size



(a) Speedup

(b) Time per model step (in logarithmic scale)

Figure 12. Performance scaling on V100 GPUs for the Box test with 6100 × 4460 grid size

4.2. The Azov Sea Test

The Azov Sea is a convenient region to test shallow water models because its dynamics and circulation can be described well by two-dimensional numerical models, thanks to small depth [19]. The computational domain of the Azov Sea has a relatively large number of land points. Figure 5 shows that more than half of the blocks is entirely land when dividing the domain into small blocks. Thus, the load-balancing method will be especially relevant here. The second series of experiments were performed for the Azov Sea to evaluate the performance scaling of the shallow water model with workload imbalances on processors and demonstrate the advantage of using the load-balancing method for calculations on CPUs and GPUs. We choose different spatial resolutions for the test: 250 meters (1525 × 1115 grid size) and 62.5 meters (6088 × 4448 grid size). It should be noted that a high resolution of the Azov Sea model is required to more correctly describe coastal flow currents and sea level changes, which are required for practical purposes, for example, for calculating level fluctuations in ports of the Azov Sea. Simulations were performed for one model day, with 86400 model steps in total.

The LB metric is responsible for balancing the partitioning in terms of computing load on processors and is defined as follows. Suppose that the partition occurs into k subdomains for p processors, then LB is defined as:

$$LB(k,p) = \frac{\max_{1\le i\le k} W_i}{\frac{1}{p}\sum_{i=1}^{k} W_i},$$

where $\max_{1\le i\le k} W_i$ – is the maximum workload of the i-th subdomain, $\sum_{i=1}^{k} W_i$ – is the full workload of the entire computational domain. The workload is computed differently for calculations on CPUs and GPUs. For CPUs, the workload of the subdomain is a sum of "wet" points in the subdomain. However, for GPUs, the workload of the subdomain is a sum of any points ("dry" and "wet"). Due to branch divergence, performance on GPUs does not drop with increasing the proportion of "wet" points, unlike on CPUs, and we take it into account. So, for calculation on GPUs, we uniformly distribute blocks from the block-based decomposition across GPUs, excluding entirely land blocks. This metric shows the ratio of the maximum subdomain's

workload to the optimal workload. The value $LB = 1$ corresponds to a perfectly load-balanced partition.

As mentioned early, the block-based decomposition divides the domain into small blocks and distributes them across available processors. On the one hand, using a small number of blocks in a partition leads to a high LB metric and unbalanced subdomains of processors. On the other hand, using a large number of blocks in a partition leads to high overheads of copying blocks' boundary values during processors synchronization, which is especially true for calculations on GPUs since we must transfer data between CPU and GPU for each block. Therefore, the number of blocks k in the partition is chosen adaptively for distribution across p processors according to the following criterion:

$$k(p) = \min_{n=0,1,\ldots} \{4^n \; s.t. \; LB(4^n, p) - LB(4^{n+1}, p) < 0.15\}. \tag{5}$$

That criterion means that optimal blocks partition is a minimum partition (4^n blocks) for which more fine partitions (4^{n+1} blocks and more) do not significantly reduce the value of the LB metric. Table 1 demonstrates optimal number of blocks for calculation on CPUs according to the described criterion; the optimal number of blocks is highlighted in the table.

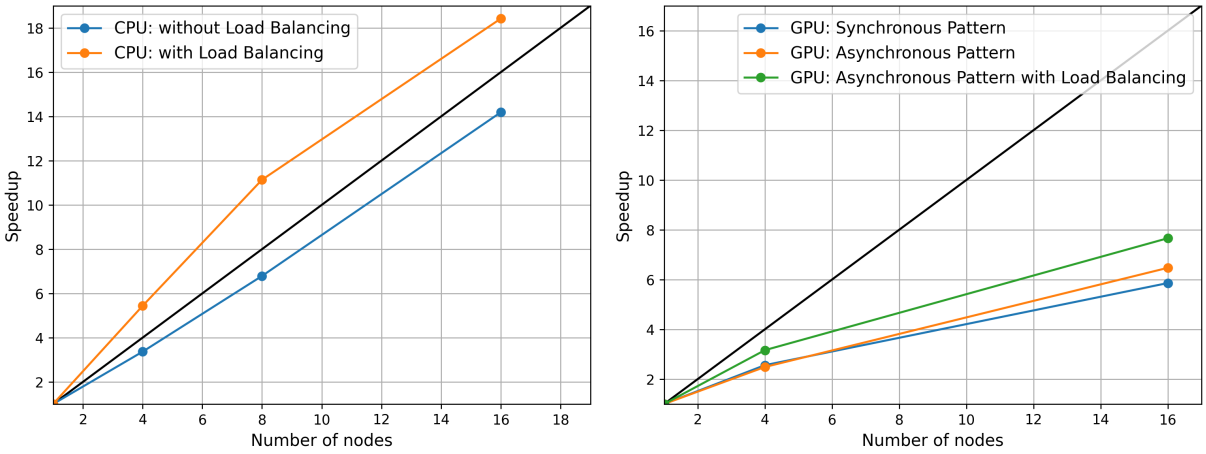
Table 1. LB metrics for the Azov Sea with 250 meters resolution

Processes	LB for 256 blocks	LB for 1024 blocks	LB for 4096 blocks
48	1.371	**1.045**	1.012
96	1.802	**1.154**	1.022
192	–	1.385	**1.070**

We tested the Azov Sea on the Pascal section of the Lomonosov-2 supercomputer. Figure 13 demonstrates performance scalings of the model with load-balancing compared to the model without load-balancing on CPUs at 1525 × 1115 grid size and on GPUs at 6088 × 4448 grid size. We considered only the pure MPI calculation pattern on CPUs and compared the asynchronous calculation pattern to the synchronous pattern on GPUs. On one node, the model was calculated without load-balancing. We used an optimal number of blocks according to criteria 5. For calculations on CPUs, Tab. 1 shows an optimal number of blocks; for GPUs, we used 64 blocks for 4 nodes and 256 blocks for 16 nodes. The figure shows that load-balancing has a significant impact on CPU computing performance: the model with load-balancing outperforms by 30% the model without load-balancing at 16 nodes and scales superlinearly due to better cache behavior of small blocks. For calculation on GPUs, it can be seen that the asynchronous pattern with load-balancing outperforms the synchronous and asynchronous patterns without load-balancing by 30% and 18% respectively at 16 nodes. Still, performance on GPUs, even with the higher resolution of the Azov Sea, scales worse than on CPUs. As mentioned before, GPUs are much more sensitive to the problem size than CPUs (see Fig. 10). That fact is especially relevant here because, with increasing nodes, the number of points per GPU dramatically decreases due to the presence of land points in the computational domain.

Conclusions

In this paper, we present the three-layer software architecture of the shallow water model based on the separation of concerns. The software architecture separates the physics-related code



(a) Speedup of CPU version for 250m resolution (b) Speedup of GPU version for 62.5m resolution

Figure 13. Performance scaling of the Azov Sea model. The Pascal section of the supercomputer Lomonosov-2 was used

from features of parallel implementation, simplifying the model's support and development. We present the blocked-based decomposition to improve the two-dimensional domain decomposition method proposing load-balanced and cache-friendly calculations on CPUs. We support the block-based decomposition for calculations on GPUs proposing overlapping kernel execution with data transfer. We present various hybrid parallel programming patterns for use on massively parallel and heterogeneous computing systems. The pure MPI and the hybrid task-based MPI-OpenMP are presented as calculation patterns on CPUs. We develop three hybrid parallel programming patterns for calculations on GPUs. The synchronous MPI-CUDA pattern supports calculations on multiple GPUs assuming only one block per MPI process and does not employ block-based decomposition. The asynchronous MPI-OpenMP-CUDA pattern overlaps kernel execution with data transfers to more effective calculation on nodes with one GPU per node than the synchronous MPI-CUDA pattern. Lastly, the multi GPUs per node MPI-OpenMP-CUDA pattern is designed to calculate on nodes with multiple GPUs per node.

We test the shallow water model on the Lomonosov-2 supercomputer at Lomonosov Moscow State University. First, we evaluate different calculation patterns' scaling performances on CPUs and GPUs at the computational domain without land points. We demonstrate that performance per a single grid point on GPUs dramatically decreases after 2^{19} points per node while performance on CPUs scales up to 2^{17} well. Although, calculations on GPUs outperform calculations on CPUs by a factor of 4.7 at 30 nodes using 360 CPU cores and 60 GPUs at 6100×4460 grid size. We demonstrate that the asynchronous MPI-OpenMP-CUDA pattern is better scaled up to 8 nodes and 28% faster on 8 GPUs than the synchronous calculation pattern. Second, we demonstrate the advantage of using the load-balancing method in the Azov Sea model. We show that the load-balancing significantly impacts computing performances: calculations with load-balancing outperform by 30% calculations without load-balancing on both CPUs and GPUs at 16 nodes.

The considered shallow water model is formulated from a barotropic solver of the ocean general circulation sigma model INMOM. This research challenges us to extend current work to the three-dimensional ocean model INMOM on heterogeneous supercomputers.

Acknowledgements

The reported study was funded by RFBR, project number 20-31-90109. The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University.

References

1. Afzal, A., Ansari, Z., Faizabadi, A.R., Ramis, M.K.: Parallelization Strategies for Computational Fluid Dynamics Software: State of the Art Review. Archives of Computational Methods in Engineering 24, 337–363 (2017). https://doi.org/10.1007/s11831-016-9165-4

2. Shchepetkin, A.F., McWilliams, J.C.: The regional oceanic modeling system (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model. Ocean Modelling 9(4), 347–404 (2005). https://doi.org/10.1016/j.ocemod.2004.08.002

3. Porter, A.R., Appleyard, J., Ashworth, M., et al.: Portable multi- and many-core performance for finite-difference or finite-element codes – application to the free-surface component of NEMO (NEMOLite2D 1.0). Geosci. Model Dev. 11, 3447–3464 (2018). https://doi.org/10.5194/gmd-11-3447-2018

4. Chaplygin, A.V., Dianskii, N.A., Gusev, A.V.: Load balancing using Hilbert space-filling curves for parallel shallow water simulations. Num. Meth. Prog. 20:1, 75–87 (2019). https://doi.org/10.26089/NumMet.v20r108

5. Chaplygin, A.V., Diansky, N.A., Gusev, A.V.: Parallel Modeling of Nonlinear Shallow Water Equation. In: Proc. 60th All-Russia Conf. on Applied Mathematics and Informatics, Moscow Institute of Physics and Technology, Dolgoprudny, Russia, November 20-26, 2017. pp. 192–194. Moscow Inst. Phys. Technol., Dolgoprudny (2017)

6. Chaplygin, A.V., Gusev, A.V.: Shallow Water Model Using a Hybrid MPI/OpenMP Parallel Programming. Problems of Informatics 1, 65–82 (2021). https://doi.org/10.24411/2073-0667-2021-10006

7. Christidis, Z.: Performance and Scaling of WRF on Three Different Parallel Supercomputers. In: Kunkel, J., Ludwig, T. (eds.) High Performance Computing. ISC High Performance 2015. Lecture Notes in Computer Science, vol. 9137, pp. 514–528. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20119-1_37

8. Akhmetova, D., Iakymchuk, R., Ekeberg, O., Laure, E.: Performance Study of Multithreaded MPI and OpenMP Tasking in a Large Scientific Code. 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 756–765. IEEE (2017) https://doi.org/10.1109/IPDPSW.2017.128

9. Diansky, N.A.: Ocean circulation modelling and research of its response to short-term and long-term atmospheric forcing. Fizmatlit, Moscow (2013)

10. Volodin, E.M., Diansky, N.A., Gusev, A.V.: Simulation and Prediction of Climate Changes in the 19th to 21st Centuries with the Institute of Numerical Mathematics, Russian Academy of Sciences, Model of the Earths Climate System. Izv., Atmos. Ocean. Phys. 49(4), 347–366 (2013)

11. Fomin, V.V., Diansky, N.A.: Simulation of Extreme Surges in the Taganrog Bay with Atmosphere and Ocean Circulation Models. Russ. Meteorol. Hydrol. 43, 843–851 (2018). https://doi.org/10.3103/S1068373918120051

12. Fu, H., Gan, L., Yang, C., et al.: Solving global shallow water equations on heterogeneous supercomputers. PLoS ONE 12(3), e0172583 (2017). https://doi.org/10.1371/journal.pone.0172583

13. Lawrence, B.N., Rezny, M., Budich, R., et al.: Crossing the chasm: how to develop weather and climate models for next generation computers? Geosci. Model Dev. 11, 1799–1821 (2018). https://doi.org/10.5194/gmd-11-1799-2018

14. Bari, M.S., Stoltzfus, L., Lin, P., et al.: Is Data Placement Optimization Still Relevant on Newer GPUs? 2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), Dallas, TX, USA, 2018. pp. 83–96. IEEE, 2018. https://doi.org/10.1109/PMBS.2018.8641666

15. NEMO Consortium. NEMO development strategy Version 2: 2018-2022. https://www.nemo-ocean.eu/wp-content/uploads/NEMO_Development_Strategy_Version2_2018-2022.pdf (2018)

16. Tintó, O., Acosta, M., Castrillo, M., et al.: Optimizing domain decomposition in an ocean model: the case of NEMO. Procedia Computer Science 108, 776–785 (2017). https://doi.org/10.1016/j.procs.2017.05.257

17. Perezhogin, P., Chernov, I., Iakovlev, N.: Advanced parallel implementation of the coupled oceanice model FEMAO (version 2.0) with load balancing. Geosci. Model Dev. 14, 843–857 (2021). https://doi.org/10.5194/gmd-14-843-2021

18. Reguly, I.Z., Giles, D., Gopinathan, D., et al.: The VOLNA-OP2 tsunami code (version 1.5). Geosci. Model Dev. 11, 4621–4635 (2018). https://doi.org/10.5194/gmd-11-4621-2018

19. Saburin, D.S., Elizarova, T.G.: Modelling the Azov Sea circulation and extreme surges in 2013-2014 using the regularized shallow water equations. Russian Journal of Numerical Analysis and Mathematical Modelling 33(3), 173–185 (2018). https://doi.org/10.1515/rnam-2018-0015

20. Smith, R., Jones, P., Briegleb, B., et al.: The parallel ocean program (POP) reference manual: ocean component of the Community Climate System Model (CCSM) and Community Earth System Model (CESM). http://www.cesm.ucar.edu/models/cesm1.0/pop2/doc/sci/POPRefManual.pdf

21. Liu, T., Zhuang, Y., Tian, M., et al.: Parallel Implementation and Optimization of Regional Ocean Modeling System (ROMS) Based on Sunway SW26010 Many-Core Processor. IEEE Access 7, 146170–146182 (2019). https://doi.org/10.1109/ACCESS.2019.2944922

22. van Werkhoven, B., Maassen, J., Kliphuis, M., et al.: A distributed computing approach to improve the performance of the parallel ocean program. Geosci. Model Dev. 7, 267–281 (2014). https://doi.org/10.5194/gmd-7-267-2014

23. Voevodin, Vl., Antonov, A., Nikitenko, D., et al.: Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community. Supercomputing Frontiers and Innovations 6(2), 4–11 (2019). https://doi.org/10.14529/jsfi190201

24. Wilhelmsson, T.: Parallelization of the HIROMB ocean model. https://pdfs.semanticscholar.org/ee95/be1a6bb90becdc31c84f83c343ca8daf5bdc.pdf

25. Xu, S., Huang, X., Oey, L.-Y., et al.: POM.gpu-v1.0: a GPU-based Princeton Ocean Model. Geosci. Model Dev. 8, 2815–2827 (2015). https://doi.org/10.5194/gmd-8-2815-2015

DOI: 10.14529/jsfi210408

PLUMED Plugin Integration into High Performance Pmemd Program for Enhanced Molecular Dynamics Simulations

Viktor V. Drobot[1,2], *Evgeny M. Kirilin*[1,2], *Kirill E. Kopylov*[2,3], *Vytas K. Švedas*[2,3]

Metadynamics as an enhanced sampling procedure of molecular dynamics simulations is an effective tool to simulate complex molecular motions, conformations and reactivity, including enzyme plasticity and catalysis. The classic non-enhanced molecular simulation tools have reached unprecedently high performance utilizing GPU units, however their implementation for enhanced sampling are still on demand. The widespread AMBER (molecular dynamics package) + PLUMED (metadynamics plugin) still does not take advantage of GPU computing or the CPU utilization optimization included in the AMBER pmemd program. In this work we have developed PLUMED binding to pmemd program resolving performance issues within hybrid molecular dynamics/metadynamics runs. Preliminary checks and test results of the model system have validated this implementation.

Keywords: high performance pmemd program, enhanced molecular dynamics simulations, PLUMED plugin integration, metadynamics, CUDA, GPU.

Introduction

Molecular dynamics (MD) is one of the key methods for modeling complex processes in living systems, including protein folding, formation of enzyme-substrate complexes, ligand binding, etc. Classic MD approach consists in the calculation of forces acting on the atoms of the system at each time frame. Complex molecules are usually represented as rigid balls (atoms) connected with springs (chemical bonds) of different hardness. Solving multidimensional systems differential equations allows to calculate forces acting on each atom and corresponding coordinates and momenta, so one can track mechanical evolution of the system in time. However classic MD has one significant drawback: because of free-running simulation atoms of the system will have specific energy distribution which leads to the movement of the whole system to the nearest local energy minimum. If one wants to study some process with activation energy barrier (chemical reaction, protein folding, conformational dynamics) it then can be hindered for observation because it requires significant movement of the system from the local energy minimum.

Running MD simulation under elevated temperature is one of the well-known approaches to evade such problems: higher kinetic energy of the system facilitates overcoming of energy barriers [3]. However, it can lead to physically nonsense states which have no reference points in empirical studies. Adaptive bias methods are more preferred to deal with local minima problems as their selective nature allows to shift the whole system from local equilibrium and push it across barriers. The adaptive bias methods open a way for reconstruction of energy surface profile, thus making assessment of other metastable states possible. AMBER [1] is one of the most popular MD packages and provides great variety of adaptive free energy methods such as:

- MM-PBSA – for calculation of protein–ligand binding energy;
- thermodynamic integration and calculational alchemistry;
- umbrella sampling;

[1]Lomonosov Moscow State University, Belozersky Institute of Physicochemical Biology, Moscow, Russia
[2]Lomonosov Moscow State University, Research Computing Center, Moscow, Russia
[3]Lomonosov Moscow State University, Faculty of Bioengineering and Bioinformatics, Moscow, Russia

- steering dynamics (e. g. APR);
- non-equilibrium free energy (NFE) methods, including simple metadynamics.

Most methods listed above are available for use in pmemd module of AMBER which is capable of using GPU for calculations (specifically, pmemd.cuda and pmemd.cuda.mpi executables), thus leading to the significant acceleration of the whole simulation. However, current list of available collective variables for NFE methods is pretty much limited while config system is not quite flexible for complex setups.

Metadynamics (MetaD) as one of the adaptive bias methods is used for potential energy surface exploration in selected coordinates (collective variables, CV) [2]. The usage of collective variables (distances between atoms, attack angles, coordination numbers etc) allows one to reduce full-dimensional phase space of generalized coordinates and momenta to a much smaller phase space of selected CVs, thus facilitating its exploration. Moreover, physically-based CVs give more clear representation of the process under study. The essence of the method is to add small biasing energy potential to the full energy of the system at specific time, thus pushing it from current local equilibrium. Having history of such additions it becomes possible to reconstruct free energy surface of the system by summing all added biases and inverting the sign.

Since collective variables are functionally dependent on generalized coordinates and momenta communication between MD and MetaD code is required. One of the most popular packages for performing MetaD runs is PLUMED [5] which acts both as a plugin for existing MD engines and as a molecular dynamics trajectory analysis tool. PLUMED provides much broader variety of CVs and adaptive bias methods (classic MetaD, well-tempered MetaD, steering dynamics, etc) and existing code has been used for 12 years. However, previously there was only one PLUMED extension for AMBER engine, which allowed to use it only with sander module and CPUs.

1. Results and Discussion

Weve implemented [4] another PLUMED extension for AMBER with the help of the original authors of the former: it allowed to use PLUMED with pmemd module of AMBER as well as its variants (MPI, CUDA, CUDA+MPI) including both CPU and GPU calculations. Original source code and idea was taken from existing extension for sander, however, the usage of GPU required special actions Fig. 1. Right after MD run all necessary information about units used (time, length, mass, charge, etc) is passed to the PLUMED. Next on each step of molecular dynamics run all current coordinates, velocities and charges are passed to the MetaD engine. After that PLUMED calculates required biasing potential and passes it back to the MD engine, which in turn sums it with full energy. At the same time the rest of the energy is calculated in the MD kernel on the CPU (sander, pmemd, sander.MPI, pmemd.MPI executables) or on the GPU (pmemd.cuda, pmemd.cuda.MPI). Key difference between our approach and the existing PLUMED extension for sander is the synchronization of coordinates and velocities of atoms with master process for the subsequent transfer to the PLUMED.

Such synchronization can be a potential bottleneck and can slow down the whole calculation. NFE code analysis has shown that currently there is no way to implement adaptive bias methods in AMBER fully on GPU. After basic implementation of PLUMED extension for pmemd all required checks were held [6] to confirm the correctness of MD and MetaD interaction:

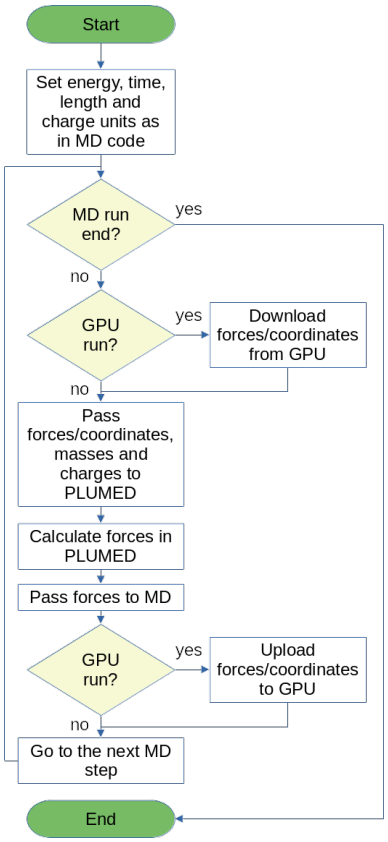


Figure 1. Classic MD and MetaD interaction algorithm. By exchanging coordinates, forces and charges between MD and MetaD engines one can calculate biasing potential in the PLUMED kernel followed by passing calculated forces back to the MD engine

- coordinates passing: to verify atomic coordinates are being passed in specific order and right units;
- integrator timestep passing: to verify that time-dependent properties will be calculated correctly in MetaD engine;
- energy passing: to verify energies are being calculated in MetaD engine;
- masses and charges passing: to verify that inertial (gyration radii, centers of mass, etc) and dipole values are being calculated right;
- forces passing: to verify calculated biases;
- virials passing: to verify pressure-dependent values;
- forces on energy passing: to verify energy-dependent biases.

All checks listed above were successfully passed. Then test system consisted of alanine dipeptide in water box (10234 atoms, TIP3P water), that is widely used as for demonstrational purposes, was prepared for metadynamics run with the new extension. The Lomonosov-2 [7] supercomputer cluster was used for executing test runs in the following configurations:

1. CPU-only MD simulations: pmemd.MPI and sander.MPI executables from AmberTools 21/Amber 20 package were compiled with GCC 9.1.0 toolchain using OpenMPI 4.1.0 and Intel MKL 2015.3.187 (including FFTW) libraries. Plumed 2.7.1 was also compiled with GCC 9.1.0 using OpenMPI 4.1.0 and BLAS/LAPACK libraries from Intel MKL 2015.3.187. PLUMED was triggered via usual AmberTools interface by setting PLUMED_KERNEL environment variable. OpenMP support was turned on during compilation. Executables were run in parallel: 36 MPI processes on single “volta2” partition node (2x Intel Xeon

Gold 6240 2.60GHz, 36 CPU cores in total; 1.5 TiB RAM; 2x NVIDIA Tesla V100, not used for this kind of run). Infiniband FDR node interconnect was not involved in the calculation. HILLS, COLVAR files and MD trajectories were stored on distributed Lustre filesystem included in Lomonosov-2 cluster.

2. GPU-involved MD simulations: pmemd.cuda_SPFP executable from AmberTools 21/Amber 20 package was compiled with GCC 9.1.0 and NVCC 11.1.74 toolchains using Intel MKL 2015.3.187 and NVIDIA CUDA Toolkit 11.1 libraries (including FFTW from MKL and CUFFT from CUDA Toolkit). Plumed 2.7.1 was also compiled with GCC 9.1.0 using OpenMPI 4.1.0 and BLAS/LAPACK libraries from Intel MKL 2015.3.187. PLUMED was triggered via usual AmberTools interface by setting PLUMED_KERNEL environment variable. MPI support was explicitly disabled during compilation, while OpenMP support was enabled. pmemd.cuda_SPFP executable was run as a single process on single node on the following Lomonosov-2 partitions:
   - "compute": 1x Intel Xeon E5-2697 v3 2.60GHz, 14 CPU cores; 64 GiB RAM; 1x NVIDIA Tesla K40s;
   - "pascal": 1x Intel Xeon Gold 6126 2.60GHz, 12 CPU cores; 92 GiB RAM; 2x NVIDIA Tesla P100;
   - "volta2": 2x Intel Xeon Gold 6240 2.60GHz, 36 CPU cores in total; 1.5 TiB RAM; 2x NVIDIA Tesla V100.

In cases when node had 2 GPUs the executable was run on the first of them. Infiniband FDR node interconnect was not involved in the calculation. HILLS, COLVAR files and MD trajectories were stored on distributed Lustre filesystem included in the Lomonosov-2 cluster. Resulting MD simulation performance is shown in Fig. 2.

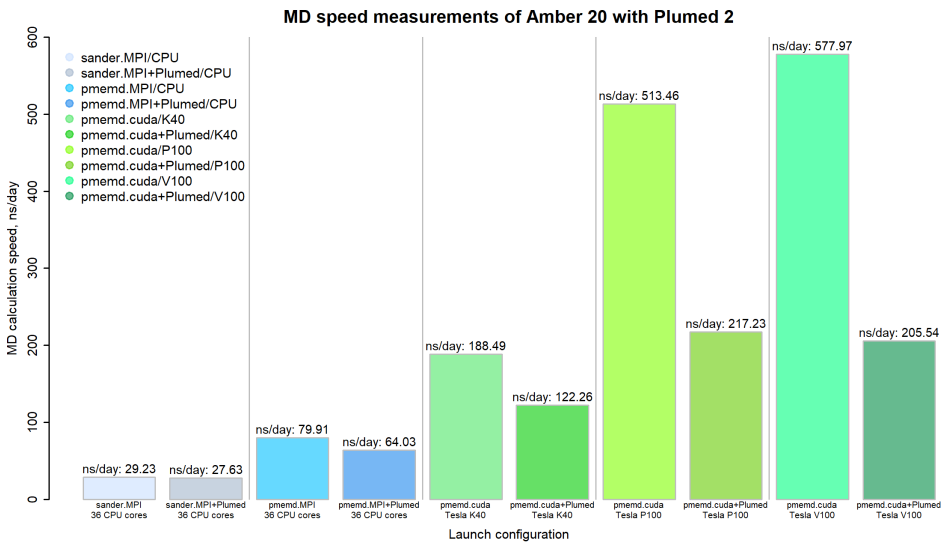


Figure 2. Dependence of MD performance of AmberTools 21/Amber 20 + Plumed 2 binding (Lomonosov-2 cluster) on the run configuration (on CPUs for single node in MPI mode; on NVIDIA GPUs for single mode on different GPU generations: K40s "Kepler", P100 "Pascal", V100 "Volta")

Performance is determined as a number of finished MD steps per specific time period, which is then converted to the usual unit of ns/day. Sander executable is included in AmberTools 21 package and have MetaD support implemented by PLUMED authors. Our version of PLUMED

extension for pmemd increases performance by a factor of 2.3. Usage of GPU for calculations gives 4.4x boost in case of NVIDIA Tesla K40s GPUs, 7.9x boost for NVIDIA Tesla P100 GPUs and 7.4x boost for NVIDIA Tesla V100 GPUs. In the latter case performance limit is observed for PLUMED + pmemd.cuda case despite of boost for classic MD run without MetaD in comparison with NVIDIA Tesla P100. Its due to the specific algorithm implementation and GPU-CPU synchronization being the most important performance-limiting step. Impact of such a synchronization is clearly demonstrated by usage of NVIDIA Nsight Systems profiler. Analysis of pmemd.cuda_SPFP executable performance on GPU run was held with NVIDIA Nsight Systems profiler (CUDA Toolkit 11.1). For this analysis, debug versions of AmberTools 21/Amber 20 and Plumed 2.7.1 were built having the configuration described above with extra enabledebug configuration option and -g compilation option. Analysis was done in terminal mode with nsys tool on single "volta2" node by using DWARF traceback algorithm.

Resulting *.qdrep file was analyzed in NVIDIA Nsight Systems GUI (Fig. 3, Fig. 4).

Figure 3. Results of profiling the original pmemd.cuda_SPFP executable version (CUDA 11.1) with NVIDIA Nsight Systems. 1 ms range is shown. Most of the calculation time is spent during "computational kernels" execution

Figure 4. Results of profiling the Plumed-enabled pmemd.cuda_SPFP executable version (CUDA 11.1) with NVIDIA Nsight Systems. 1 ms is shown. Data exchange between CPU and GPU takes about 42% of all calculation time while "computational kernels" execution takes about 58% of time

Profiling has shown that for the original version of pmemd.cuda_SPFP executable most of the time is spent during "computational kernels" execution on GPU while the data exchange between CPU and GPU have almost negligible impact on the overall calculation time: all required data is already placed in the GPU memory.

For the runs with PLUMED plugin enabled profiling has shown that very intensive transmission of data between CPU and GPU occurs on each MD run step, thus taking about 42% of all calculation time. Unfortunately, current PLUMED calculations can be done only on CPU and such transmission is absolutely required for synchronization. It leads to the bottleneck and limits MD run performance. In general, the use of PLUMED in combination with the high performance pmemd executable of the AmberTools software suite can successfully accelerate the calculations of enhanced sampling methods based on molecular dynamics. The next fundamental step in acceleration would be the transfer of the existing additional metadynamic potential to the GPU memory to lower CPU and GPU synchronization while updating metadynamic potential is not rate limiting task up to date.

Acknowledgements

This work was supported by the Russian Foundation for Basic Research (grant 20-04-01119). The research was carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University [7].

References

1. Case, D.A., Aktulga, H.M., Belfon, K., et al.: Amber 2021. University of California Press (2021), http://ambermd.org/doc12/Amber21.pdf

2. Laio, A., Parrinello, M.: Escaping free-energy minima. Proceedings of the National Academy of Sciences 99(20), 12562–12566 (2002). https://doi.org/10.1073/pnas.202427399

3. Qi, R., Wei, G., Ma, B., Nussinov, R.: Replica Exchange Molecular Dynamics: A Practical Application Protocol with Solutions to Common Problems and a Peptide Aggregation and Self-Assembly Example, pp. 101–119. Springer, New York, NY (2018). https://doi.org/10.1007/978-1-4939-7811-3_5

4. The PLUMED Consortium: Plumed 2 source code repository. Pull request #486 (2019), https://github.com/plumed/plumed2/pull/486

5. The PLUMED Consortium: Promoting transparency and reproducibility in enhanced molecular simulations. Nature Methods 16(8), 670–673 (2019). https://doi.org/10.1038/s41592-019-0506-8

6. The PLUMED Consortium: The PLUMED manual, version 2.7 (2020), https://www.plumed.org/doc-v2.7/developer-doc/html/_how_to_plumed_your_m_d.html

7. Voevodin, V.V., Antonov, A.S., Nikitenko, D.A., et al.: Supercomputer Lomonosov-2: Large scale, deep monitoring and fine analytics for the user community. Supercomputing Frontiers and Innovations 6(2), 4–11 (2019). https://doi.org/10.14529/jsfi190201

DOI: 10.14529/jsfi210409

Turbulent Length Scale for Multilayer RANS Model of Urban Canopy and Its Evaluation Based on Large-Eddy Simulations

Andrey V. Glazunov[1,2,3], ***Andrey V. Debolskiy***[3,4], ***Evgeny V. Mortikov***[1,3]

Large-Eddy Simulation (LES) numerical experiments of neutrally-stratified turbulent flow over an urban-type surface and passive scalar transport by this flow are performed. A simple parameterization of the turbulent length scale containing only one empirical constant is proposed. Multilayer Reynolds-Averaged Navier-Stokes (RANS) model of turbulent flow and turbulent scalar diffusion is constructed. The results of the RANS model are compared with the LES experiments. It is shown that the proposed approach allows predicting the average flow velocity and the scalar concentration inside and above the urban canopy.

Keywords: atmospheric boundary layer, numerical simulation of turbulence, urban canopy, scalar turbulent transport.

Introduction

Increasing performance of supercomputers makes it possible to make detailed weather forecasts, in which the horizontal scale of large metropolitan areas is resolved explicitly on grids of General Circulation Models (GCMs). One of the important blocks of large-scale models are local one-dimensional RANS models of the atmospheric boundary layer, which describe the near surface turbulent transport of momentum and scalar quantities vertically. In most contemporary GCMs, any surface, including the urban canopy, is considered a rough surface coinciding with the smoothed terrain, and its small-scale geometric features (e.g., high-rise buildings) are taken into account only in terms of their integral impact on the turbulent exchange processes between the surface as a whole and the atmosphere above. At the same time, along with grid refinement of GCMs horizontally, their vertical resolution also increases so that for several grid nodes with a step of the order of several meters close to the surface there are city buildings which induce form drag that needs to be accounted for. So, it is necessary to develop multilayer one-dimensional turbulence models that take into account the dynamic and thermal effects of buildings and vegetation on turbulence in the form of vertically distributed forcing and incorporate special turbulent closures for them. In such models, the surface layer is considered as a porous medium, which creates volumetric resistance to the mean wind and is capable of generating turbulent kinetic energy in the flow around buildings.

The first such models appeared quite a long time ago (see, e.g., [17]) and now incorporate parameterizations of various physical processes, such as multilayer parameterizations of radiation sources and heat sinks, the dynamic and thermal interactions of turbulence with vegetation (see [5, 14–16, 20]). These models will eventually make it possible to refine the meteorological characteristics within the urban canopy near surface. In addition, an important stimulus for the development of multilayer RANS models is the need to predict the concentrations of pollutants represented by gaseous impurities and particulate matter.

[1]G.I. Marchuk Institute of Numerical Mathematics, Russian Academy of Science, Moscow, Russian Federation
[2]Moscow Center for Fundamental and Applied Mathematics, Moscow, Russia
[3]Research Computing Center, Lomonosov Moscow State University, Moscow, Russian Federation
[4]A.M. Obukhov Institute of Atmospheric Physics,Russian Academy of Science, Moscow, Russian Federation

One of the most important problems arising in the development of multilayer one-dimensional models of the urban canopy is the difficulty of independent evaluation of their individual components. Measurements in real cities are not always feasible, since they are subject to the combined influence of numerous external meteorological factors and have significant spatial heterogeneity. There is a need for evaluation of RANS models in idealized conditions simulating urban turbulence in a state of statistical equilibrium and spatial homogeneity on horizontal scales exceeding the size of large elements of urban surface roughness. It is possible to create such conditions in two ways. The first one is traditional and consists of setting up specialized laboratory experiments to measure the statistical characteristics of turbulence of a flow around objects similar to buildings, but with simple shape and regular structure of their location on the surface (see, for example, [3]).

For laboratory experiments performing high Reynolds number flow experiments, comparable to urban canopy in atmospheric boundary layer, remains a challenge. Researchers resort to some combined way of organizing measurements, placing large arrays of simple in shape large objects (e.g., cubes of a 1.5 m size) in the open air, rather than in the laboratory setup (see, experiment COSMO (Comprehensive Outdoor Scale Model) http://www.ide.titech.ac.jp/ kandalab/COSMO/COSMO.html [13]). This approach provided reliable data for developing parameterizations of the dynamic and thermal interaction of turbulence with urban-type surfaces (see, e.g., [12]).

Another way to obtain low noise, detailed and sufficiently reliable data on turbulent dynamics and turbulent scalar transport in urban-type environments is direct numerical simulation (DNS) and large-eddy simulation (LES). The use of numerical models (both LES and DNS) to establish the aerodynamic characteristics of urban-type surfaces is not new. The first such numerical simulations were performed more than 15 years ago [4, 21] and showed good agreement with laboratory data [3]. Contemporary HPC systems, not only allow us to significantly extend the range of surface configurations considered, but also to accurately resolve the characteristics of turbulent flow within the $0 < z < h$ layer filled with roughness objects. This, in particular, makes it possible to use LES data to construct and verify multilayer local one-dimensional RANS models.

In [19] the LES data obtained from flow simulations over an array of cubes (see Fig. 3 of [19]), a second order (K-l) multilayer model based on prognostic equation for turbulent kinetic energy (TKE) and prescribed turbulent length scales within the canopy layer was calibrated and validated: l_m to calculate the turbulent viscosity and TKE diffusion coefficients and l_ε to calculate TKE dissipation rate. In that study universal functions $\tilde{l}_m(\tilde{z})$ and $\tilde{l}_\varepsilon(\tilde{z})$ of these scales are proposed, which depend on the geometric parameters of urban canopy (here: $\tilde{l}_m = l_m/h$, $\tilde{l}_\varepsilon = l_\varepsilon/h$, $\tilde{z} = z/h$ are normalized by the thickness of the urban canopy layer). The one-dimensional model is constructed by introducing an additional drag force into the equation for the average velocity U:

$$F_D = -C_D U \left|U\right|, \tag{1}$$

where C_D is positive inside the layer $0 < z < h$ and, generally speaking, depends on z volume drag coefficient with dimensions $[C_D] = \mathrm{m}^{-1}$. In addition, a new term is added to the TKE balance equation, which describes its production through the interaction with the buildings:

$$P_D = C_D \left|U\right|^3. \tag{2}$$

The authors of [19] have shown with their tests that the parameterization calibrated with LES data improves the reproduction of the average velocity U both inside and above the $0 < z < h$ layer. In addition, that paper has provided comparisons with other independent simulations and measurements.

Our study has two research objectives. First, we will try to test the performance of the proposed parameterizations for surfaces with relatively small object densities (on the grounds that when including them in GCMs, we will first have to consider the impact of the highest buildings, which are typically represented with a small fraction). Second, we will test how the proposed parameterizations model turbulent scalar transport within canopy. These goals seem to us even more relevant than modelling of the mean wind and its dispersion profiles in the urban environment, because it is directly related to the possibility of forecast of pollutants concentrations.

In addition, we propose the simplest way to construct a turbulent length scale within the roughness layer. This scale is derived from dimensional considerations and includes only one empirical parameter. In addition, we propose to calculate this scale without using data of the geometry of the urban surface, which is extremely difficult to generalize and express by a single number. We suggest that the nature of the turbulence itself can do this generalization for us. Indeed, above the roughness layer at large Reynolds numbers and for neutral stratification, the profile of the dimensionless mean wind velocity $\tilde{U}$ can be approximated by a logarithmic relationship:

$$\tilde{U} = \frac{U}{U_*} = \frac{1}{\kappa} \ln\left(\frac{z - D_u}{z_{0u}}\right), \tag{3}$$

where U_* is the friction velocity ($U_* = |\tau_s|^{1/2}$; τ_s is the mean tangential frictional stress on the surface as a whole normalized by air density); $\kappa \approx 0.4$ is the von Karman constant; z_{0u} is the roughness length parameter, D_u is the displacement height. The parameter $z_{0u} \ll h$ has the meaning of a height independent additive constant to the entire dimensionless velocity profile and characterizes the surface as a whole in terms of the efficiency of momentum exchange with the atmosphere, and hence it is, to a greater extent, related to the value of the drag coefficient C_D. At the same time, the validity of using the height $z' = (z - D_u)$ in the approximation (3) suggests that the characteristic turbulent length scales in the flow over the complex surface due to the influence of surface geometry are proportional to the scale $z' = z - D_u$ (instead of their proportionality to the scale z over the flat wall) with small corrections. Our null hypothesis is that the dimensionless parameter D_u/h characterizes the turbulent length scales both above and inside the canopy layer, since at least the largest turbulent fluctuations obviously cannot be independent outside and inside the urban environment. The specific form of our proposed approximation is given below. Note that the parameters z_{0u} and D_u are measurable and can be obtained from meteorological observations and eddy covariance momentum flux measurements over the urban surface.

Below we present the results of numerical experiments with the INM RAS LES model [7], which has been previously repeatedly tested for modeling turbulence over urban surfaces [8–11]. The experiments have been performed in the traditional formulation, which coincides, except for minor details, with the numerical experiments design carried out in [4, 21] and [19]. In addition to modeling the flow itself, we calculated the turbulent scalar transport with the source at the surface.

A one-equation $K - l$ turbulence model similar to the [19] model was used as a local one-dimensional RANS model and, in the same way, augmented with parameterization of the bulk

form drag (1) and TKE production (2). The main difference is in the choice of the turbulent length scale. For comparison, we also present the results of calculations with our RANS model with the length scales proposed in [19]. Here we do not investigate the parameterization of the C_D coefficient, so we will use the coefficient calculated from LES data for its assignment in both RANS models.

The results of the local-one-dimensional models are compared with the LES data and with each other. We will show that our proposed simple parameterization of the turbulent scale on the considered surface geometries is not inferior, and in some cases superior, to the parameterizations built on the basis of generalization of geometric parameters of the urban environment.

The article is organized as follows. The sections 1 and 2 provide a brief descriptions of the models (LES and RANS, correspondingly). Section 3 describes the setup of numerical experiments with the models and the parameters of numerical calculations. The proposed turbulent length scale is presented in section 5. Section 6 shows the results of RANS calculations and their comparison with LES data.

1. LES Model Description

The model explicitly reproduces the filtered velocity $\overline{\mathbf{u}} \equiv F_{\overline{\Delta}}(\mathbf{u})$, except for small-scale fluctuations $\mathbf{u}'' = \mathbf{u} - \overline{\mathbf{u}}$ (here: $F_{\overline{\Delta}}$ is a given spatial filter commuting with differentiation operators). The differential equations for conservation of momentum and mass of incompressible fluid in tensor notation have the following form:

$$\frac{\partial \overline{u}_i}{\partial t} + \frac{\partial \overline{u}_i \overline{u}_j}{\partial x_j} = -\frac{\partial \tau_{ij}}{\partial x_j} - \frac{\partial \overline{p}}{\partial x_i} + \overline{F_i^e}, \qquad \frac{\partial \overline{u}_i}{\partial x_i} = 0, \tag{4}$$

where F_i^e corresponds to the external force acting on the flow, as well as the Coriolis acceleration and buoyancy forces (equal to zero in the case under consideration); $\overline{p}$ is the normalized pressure; $\tau_{ij} = \overline{u_i u_j} - \overline{u_i}\ \overline{u_j}$ is the "sub-grid/ sub-filter" stress tensor, subject to parameterization. The term involving the kinematic viscosity of air ν is usually neglected in LES models of atmospheric boundary layer flows.

The system of equations (4) is supplemented by the filtered scalar transport equation $\overline{s}$:

$$\frac{\partial \overline{s}}{\partial t} + \overline{u}_i \frac{\partial \overline{s}}{\partial x_i} = -\frac{\partial \vartheta_i^s}{\partial x_i} + \overline{F_s}, \tag{5}$$

where $\overline{F_s}$ are bulk sources; $\vartheta_i^s = \overline{s u_i} - \overline{u_i}\ \overline{s}$ are parameterizable subgrid fluxes.

The system of equations (4), (5) is solved by an explicit method. The conservative fourth-order accurate spatial approximation [18] and the second-order Adams-Bashforth scheme in time are applied. The Poisson equation is solved iteratively by the preconditioned conjugate gradient method. The equations are discretized on a regular staggered grid.

1.1. Turbulent Closure

A mixed subgrid/subfilter model [1] is used to compute the tensor τ_{ij}:

$$\tau_{ij}^{mix} = \tau_{ij}^{smag} + \tau_{ij}^{ssm} = -2(C_s\overline{\Delta})^2 |\overline{S}| \overline{S}_{ij} + (\overline{\overline{u}_i\ \overline{u}_j} - \overline{\overline{u}_i}\ \overline{\overline{u}_j}), \tag{6}$$

where $\overline{S}_{ij}$ is the filtered strain-rate tensor; C_s is the dimensionless coefficient variable in space and time dependent on the local flow characteristics and determined dynamically [6]. The details of the localized dynamic model and features of its numerical implementation are described in [7].

The turbulent diffusion model is used as a closure for scalar fluxes:

$$\vartheta_i^s = -{K_h}^{subgr} \frac{\partial \overline{s}}{\partial x_i}, \tag{7}$$

where ${K_h}^{subgr} = (1/Sc^{subgr})(C_s\overline{\Delta})^2|\overline{S}|$ is subgrid diffusivity coefficient. The turbulent subgrid Schmidt number Sc^{subgr} has a fixed value $Sc^{subgr} = 0.8$.

1.2. Boundary Conditions

In the numerical experiments presented below, a simple configuration of roughness elements is considered, which does not require accurate reproduction of the positions of flow separation and the formation of internal boundary layers. It is assumed that within the grid cell closest to the surface, the flow has a plane-parallel structure, which obeys the general laws of turbulent near-wall flow over the surface with small roughness elements.

The natural boundary conditions for the resolved velocity components at the solid boundaries of the region for staggered grids are the free slip and non-penetration:

$$\partial \overline{\mathbf{u}}_s/\partial n|_\Gamma = 0; \qquad \overline{u}_n|_\Gamma = 0, \tag{8}$$

where $\overline{\mathbf{u}}_s$ is the tangential velocity to the boundary Γ, and $\overline{u}_n$ is the normal velocity component. This conditions allows us to obtain an approximation of the convective terms of the equation of motion (4) , which preserves the kinetic energy in the absence of viscosity.

The terms associated with the subgrid turbulent stresses are approximated in flux form, so that the wall friction is taken into account by setting the surface tangential stress vector $\tau_s = (\tau_{in}\tau_{jn})|_\Gamma$, depending on the velocity $\overline{\mathbf{u}}_s = (u_i, u_j)$ at grid nodes distant from the boundary by a distance $\Delta_{gn}/2$ (Δ_{gn} is the grid step in the direction of the normal n to the surface):

$$\tau_{in}|_\Gamma = -C_u^2|\mathbf{u}_s|\overline{u}_i. \tag{9}$$

The momentum exchange coefficient C_u in these calculations was set the same for all surfaces of urban canopy (ground, roofs and walls) and corresponds to the value of the roughness parameter $z_0/h = 6.25 \cdot 10^{-4}$, where h is the height of objects.

Periodic boundary conditions are used at the lateral boundaries of the computational domain, and the condition (8) is used at the upper boundary. The boundary conditions for scalar quantities are similar to those for the velocity components tangential to the surface (8). For the scalar transport, the source at the surface is specified as an additional constant flux Q_s.

1.3. Parallel Implementation

Algorithms for solving system of equations (4), (5) of the LES model are parallelized using MPI library. A three-dimensional decomposition of the computational domain is applied. At each model step and at each iteration in procedures for Poisson and dynamic Smagorinsky closure, the MPI-processes exchange with each other by data related to the boundary nodes of the decomposition subdomains. Collective communication calls are used to calculate the norms of vectors used in iterative methods and for spatial averaging in statistical data processing procedures. Depending on the spatial approximation of a particular differential operator, the data for exchanges are shifted by one or two grid nodes from the subdomain boundaries. Most of the exchanges between the processes are implemented using MPI non-blocking subroutines.

2. RANS Model Description

For a horizontally homogeneous, unidirectional, neutrally stratified flow, the one-dimensional (in the vertical direction) equations of the RANS model are as follows:

$$\begin{aligned} \frac{\partial U}{\partial t} + \frac{\partial \tau}{\partial z} &= -C_d \left|U\right| U + F_e, \\ \frac{\partial S}{\partial t} + \frac{\partial Q_s}{\partial z} &= F_s. \end{aligned} \tag{10}$$

Here U, and S are mean wind speed and scalar concentration, respectively. $F_D = -C_d \left|U\right| U$ is the quadratic form drag.

The turbulent momentum flux τ and turbulent kinematic scalar concentration flux Q_s are calculated using the gradient approximation:

$$\begin{aligned} \tau &= -K_m \frac{\partial U}{\partial z}, \\ Q_s &= -K_h \frac{\partial S}{\partial z}, \end{aligned} \tag{11}$$

where K_m and K_h are the turbulent viscosity and diffusivity coefficients, which, in contrast to the LES formulation, do not depend on the length scale related to the filter width or the grid resolution. The system of equations (10), (11) corresponds to the simplified boundary layer model used in large-scale atmospheric models. We will consider a closure in which the coefficients K_m and K_h are determined from the similarity relations:

$$\begin{aligned} K_m &= S_m \frac{E_k^2}{\varepsilon}, \\ K_h &= S_h \frac{E_k^2}{\varepsilon} \equiv K_m / Sc_t. \end{aligned} \tag{12}$$

The TKE is described by a prognostic equation:

$$\frac{\partial E_k}{\partial t} - \frac{\partial}{\partial z} \frac{K_m}{\sigma_k} \frac{\partial E_k}{\partial z} = P + P_D - \varepsilon, \tag{13}$$

where P_D is the TKE production through interactions with buildings, given by the formula (2), and P is the TKE production by mean shear:

$$P = K_m \left(\frac{\partial U}{\partial z} \right)^2. \tag{14}$$

The TKE dissipation rate ε is defined with a diagnostic turbulent length scale l_ε:

$$\varepsilon = \frac{E_k^{3/2}}{l_\varepsilon}. \tag{15}$$

In the equations (12–15) S_m and S_h are universal functions determining, in particular, the turbulent Schmidt number $Sc_t = S_m / S_h$, σ_k is the closure parameter responsible for the TKE diffusion. Here, we will follow the traditional approach and assume $\sigma_k = 1$. In stratified turbulent flows, the functions S_m and S_h depend on the Richardson number Ri. Since in these calculations buoyancy was not considered, we set constant values: $S_m = 0.09$, $Sc_t = 0.8$, $S_h = S_m / Sc_t$.

Obviously, the interactions between flow and buildings significantly changes the scale of turbulent vortices both inside and above the urban canopy. In the standard model described above, a single length scale l_ε is introduced, and all other length scales are proportional to it, since the model coefficients are fixed and do not change values depending on the solution and position in space. The standard choice for the turbulent scale over a flat wall is:

$$l_\varepsilon = l_T = \kappa z, \tag{16}$$

In the external flow over the urban medium (at $z > h$) we can use approximation:

$$l_T = \kappa(z - D_u). \tag{17}$$

It remains to choose the length scale in the layer $0 < z < h$. A separate section 5 of this paper is devoted to this issue, where we will discuss construction of the length scale using methods reminiscent of those of similarity theory. Thus, we retain the standard and well-known K-l model of turbulence (see, for example, [2]) without changing the relations between length scales and without introducing any other corrections responsible for time scales.

An alternative approach is to introduce unknown, different functions ψ_i of arbitrary form within layer $0 < z < h$ for length scales of dissipation l_ε, velocity l_m, scalar fluctuations, and so on, which are assumed universal: $l_\varepsilon = z\psi_\varepsilon(z/h, \varsigma_1, \varsigma_2, \varsigma_3, ...)$, $l_m = z\psi_m(z/h, \varsigma_1, \varsigma_2, \varsigma_3, ...)$,.... Here, $\varsigma_1, \varsigma_2, \varsigma_3, ...$ is a set of dimensionless parameters generated from all geometric dimensions of the underlying surface. In fact, this is the approach proposed in [19] to construct l_ε and l_m, where the functions ψ_ε and ψ_m are determined and calibrated from LES data. The main defect of this method is the difficulty of determining a limited set of leading dimensionless parameters ς_i, $i < N \sim 1$. Moreover, this choice may not be universal with respect to the underlying surface geometry.

We do not cite formulas from [19] in this paper due to their unwieldiness and some differences in writing in terms of the system of equations (12)–(15). However, we have adopted their turbulence model within our RANS code and will compare its results with those obtained using the l_T scale described below.

3. Numerical Experiments Design

3.1. LES Model

Here we present the results of three simulations performed on an equilateral grid of 512×256×128 nodes. The experiments $EXP1$, $EXP2$, and $EXP3$ (see, Fig. 1a) differ in the geometry and location of the objects on the lower boundary of the area (cubes and rectangular parallelepipeds simulating buildings). In this figure: h and $h/2$ are the heights of the objects (there are 32 grid steps Δ per height h). The height h will further be used as a length unit to normalize the results. The size of the entire computational domain was: $L_x \times L_y \times L_z = 16h \times 8h \times 4h$.

The flow is accelerated by a constant external pressure gradient $F_i^e = -dP/dx = U_{*fix}^2/(L_z - h)$, where U_{*fix} is the given value of friction velocity at height h in equilibrium. The initial profiles of the mean streamwise velocity at $z > D_u$ were calculated using equation (3) and from preliminary estimates of the parameters z_{0u} and D_u. The average scalar concentration was set equal to zero in the entire computational domain. An additional term is added to the right-hand side of the prognostic equation of the scalar

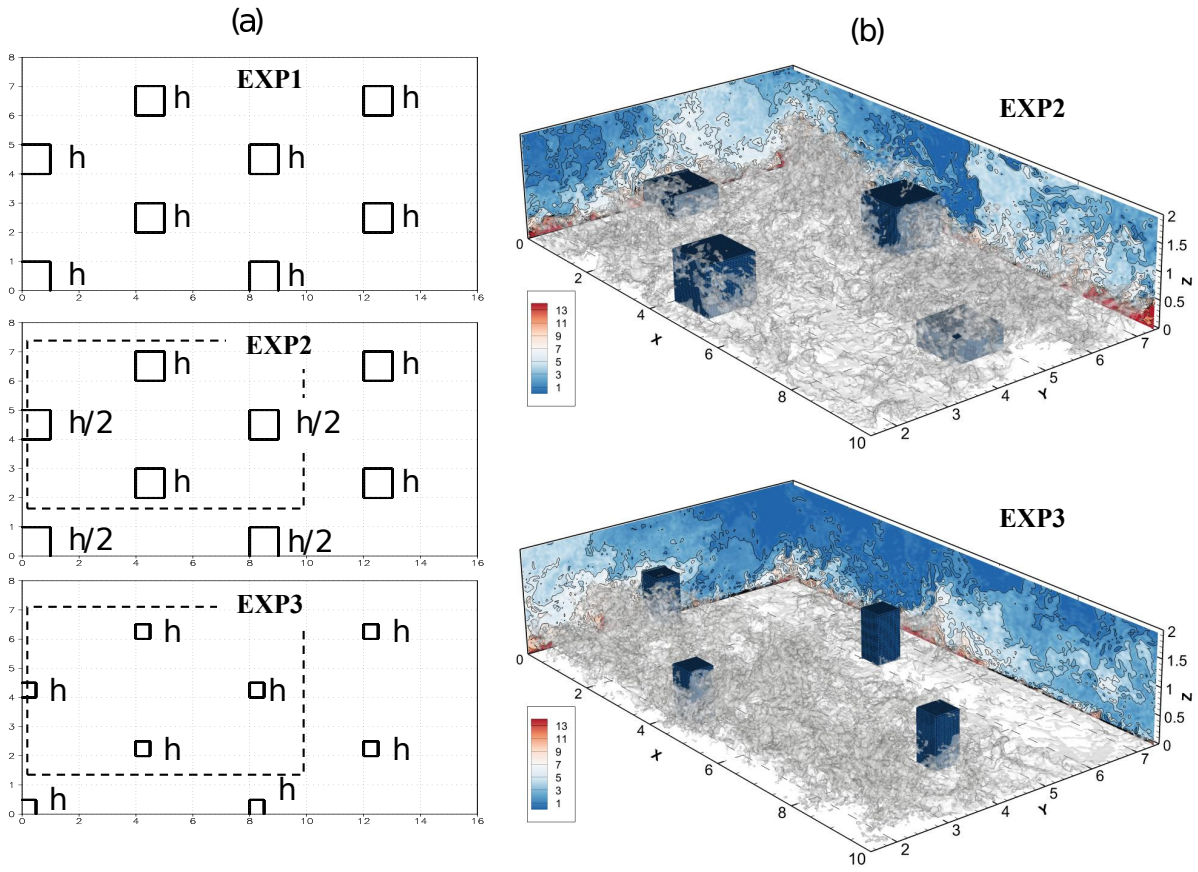


Figure 1. (a) configuration of streamlined objects; (b) visualization of the instantaneous flow fields in EXP2 and EXP3 (inside the area denoted by the dotted line on the left); the color on the section shows the normalized scalar concentration transported by the turbulent flow $(S - S_{top})/S_*$; isosurface – $(S - S_{top})/S_* = 7$

concentration $\overline{s}$ to compensate for the trend of the mean concentration, which is independent of the coordinates:

$$\overline{F_s} = \frac{1}{V_{air}} \int_\Gamma -Q_s d\Gamma, \tag{18}$$

where V_{air} is the volume of the part of the model domain filled with air, and Γ is the surface on which the concentration flux Q_s is set (in this case, the "ground" surface).

To initialize the simulations, random noise of small amplitude was imposed on the initial conditions. The calculations were run for 40 units of dimensionless time $\tilde{t} = L_z/U_{*fix}$, the last 10 units of dimensionless time were used to average the results. To normalize the scalar concentration profiles, the presented results use the turbulent concentration scale $S_* = \langle w's' \rangle_h / U_*$, where $\langle w's' \rangle_h$ is the total time- and space-averaged steady-state scalar flux (including subgrid diffusion) at height $z = h + 1.5\Delta$ (Δ is the model grid step). The velocity is normalized by the friction velocity U_* at the same height. Since we set the scalar flux from the surface rather than calculating it, we compare the LES results with the RANS results by the scalar concentration defect $(S - S_{top})/S_*$, where S_{top} is the calculated concentration near the top of the model domain. From the physical point of view, this means that we compare the concentrations calculated in LES and RANS under the condition of equality of their fluxes from the ground surface and that at the height L_z is scalar concentration is equal to zero. Figure 1 shows the concentration

$(S - S_{top})/S_*$ (one of the instantaneous states) in experiments EXP2 and EXP3 to demonstrate the turbulent nature of the flow.

3.2. RANS-models

The conditions of numerical experiments with one-dimensional RANS models are identical to those in LES. The same height of the computational region $L_z = 4h$ and the same roughness parameter of the ground surface $z_0/h = 6.25E - 4$ were chosen. The grid in all RANS-model experiments matched the vertical grid of the LES model. For correctness of comparison here we will not apply any parameterizations of the drag coefficient $C_D(z)$ in both models, but will take its "true" value obtained from LES results (see section 4). Additional checks have shown that, at qualitative level, the conclusions presented below do not change when the parameterized coefficient C_D is substituted.

4. LES Results

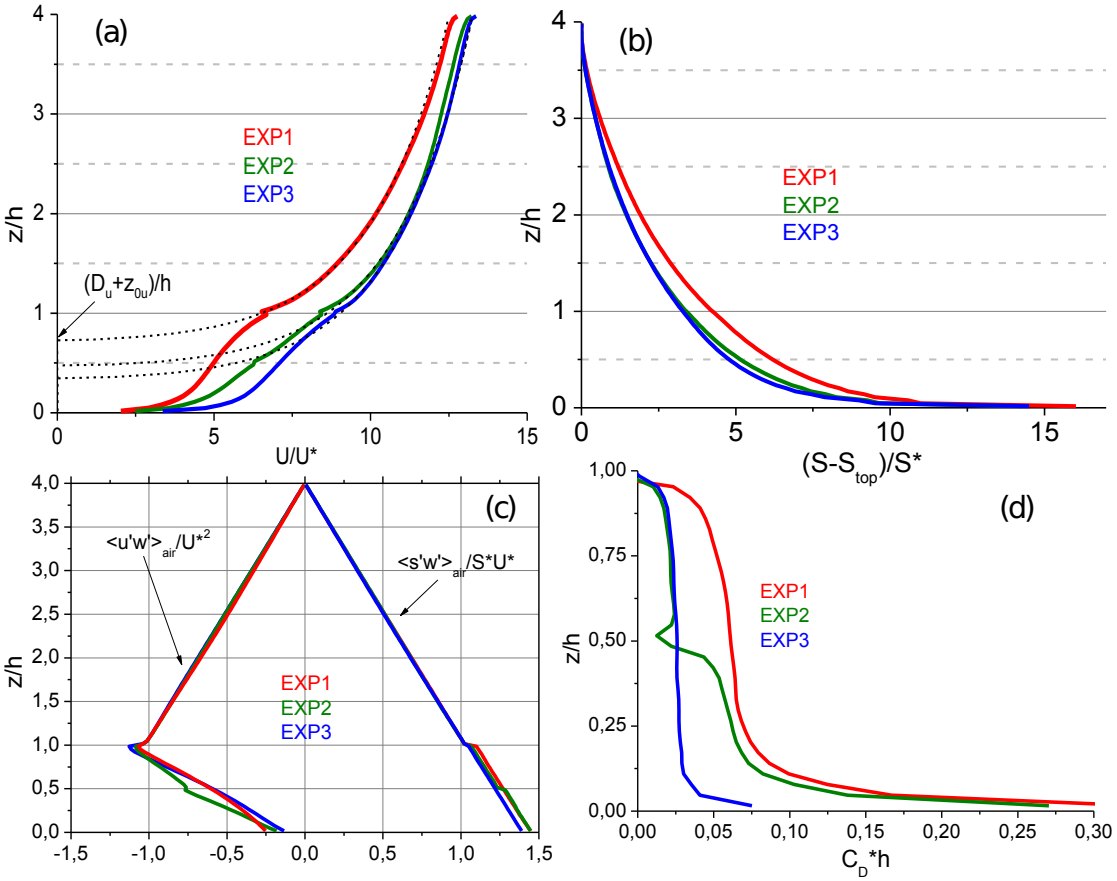


Figure 2. Results of LES-model experiments with configurations EXP1, EXP2 and EXP3: (a) normalized velocity $U/U*$, (b) scalar concentration defect $(S - S_{top})/S_*$, (c) normalized momentum and concentration fluxes (averaging over part of the region occupied by air), (d) normalized form drag coefficient hC_D

Some results of the LES numerical experiments are shown in Fig. 2. Here Fig. 2a shows the average flow velocity in all three experiments. The dotted curves approximate the velocity

profiles by logarithmic law (3). Displacement heights D_u and roughness parameters z_{0u} were determined by minimizing the standard deviation of the approximations from the LES profiles at the model grid cell located in the height interval $1.1 < z/h < 2.6$. Figure 2a shows that the dependence (3) sufficiently well approximate the simulated mean velocity. The largest deviations of the approximation from the values obtained in LES are observed for the surface in EXP2 with variable roughness elements height. In addition, Fig. 2a shows that the surfaces EXP2 and EXP3 produce approximately the same aerodynamic drag for the external turbulent flow above them, since the average flow velocities in these calculations are close to each other. The surface EXP1 has greater aerodynamic drag and provides a lower flow velocity for the same external force accelerating the flow. The data on the values of parameters D_u and z_{0u} are presented in Tab. 1. The dimensionless scalar concentration defects are shown in Fig. 2b. This figure shows that the

Table 1. Roughness length z_{0u} and displacement height D_u determined from the mean average velocity profiles for different surface configurations

	z_{0u}/h	D_u/h
$EXP1$	$2.2 \cdot 10^{-2}$	0.7
$EXP2$	$1.76 \cdot 10^{-2}$	0.45
$EXP3$	$1.83 \cdot 10^{-2}$	0.32

surfaces EXP2 and EXP3 provide approximately the same ventilation of the layer $0 < z/h < 0$, and for the surface EXP1 the concentration of the impurity in the lower part is higher.

Figure 2c shows the full (including the explicitly resolved and subgrid part in LES) scalar and momentum fluxes, normalized to the corresponding turbulent scales. It can be seen from Fig. 2c that in all three experiments an equilibrium state is reached, in which the divergence of the mean fluxes balances a constant external force: for both concentration and momentum, the fluxes have a linear form at $z/h > 1$. Here, the averaging was performed only over the part of the computational domain occupied by air, so the concentration fluxes in Fig. 2c have discontinuities at heights where the area occupied by the objects changes. When averaged over the entire computational domain and considering zero fluxes inside the objects, the heat fluxes are continuous and piecewise linear. The momentum flux profiles in this averaging will have discontinuities due to subgrid friction against the top boundary of the objects and due to the abrupt change in form drag.

The normalized drag coefficient $C_D h$ obtained for the different LES experiments is shown in Fig. 2b. This coefficient is calculated from the balance of mean forces in the equilibrium state:

$$C_D U_{air}^2 = -\frac{\partial P}{\partial x} - \frac{\partial \langle u'w' \rangle_{air}}{\partial z}. \tag{19}$$

Here, the averaging is also performed over the part of the region filled with air. The corresponding correction factor S_{air}/S_{tot} differing slightly from unity is taken into account when substituting the factor $C_D(z)$ in the RANS model, where no real objects are present. Note that EXP2 and EXP3 produce the same coefficient C_D at the top of the canopy layer. This is apparently due to the fact that the frontal area of objects per unit volume in these calculations at $0.5 < z/h < 1$ is the same, and the interposition of objects does not have much influence on the value of C_D

at their rather sparse distribution. Note that although EXP2 and EXP3 configurations differ significantly in the value of C_D at small values of z/h, they set approximately the same resistance for the external flow (the mean velocities of the external flows in EXP2 and EXP3 are close to each other), that is: for the value of z_{0u} the upper part of the urban geometry has a determining influence. As it will be seen from the results presented in section 6, none of the multilayer RANS models tested reproduce this effect.

5. Turbulent Length Scale for RANS

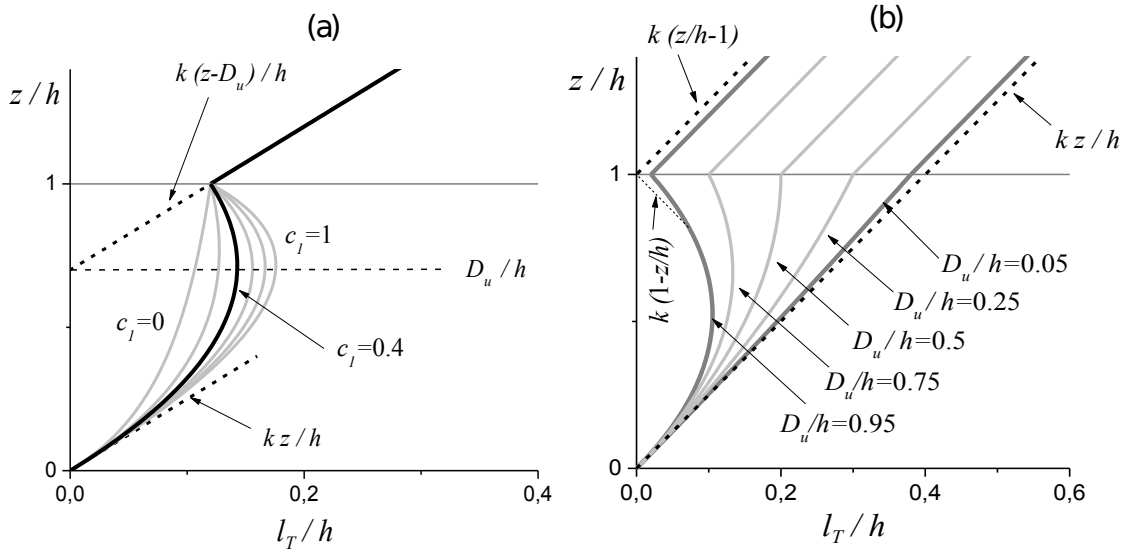


Figure 3. (a) variations of the turbulent scale l_T (23) with changes in c_1 (grey solid lines). The curve at $c_1 = 0.4$ is highlighted in black. All simulations with the RANS model shown in Fig. 4 are performed with this value of c_1. (b) variations of l_T with changes in D_u (here, $c_1 = 0.4$)

As noted in the introduction, the length scale D_u (along with some average building height scale $\langle h \rangle$ and ground surface distance z) is one of the defining turbulent length scales, as indicated by the calculated velocity profiles (see Fig. 2) that are close to logarithmic dependence (3). Indeed, when the velocity gradient is normalized by length scale $z' = z - D_u$ we obtain a value close to the constant:

$$\frac{dU}{dz}\frac{z - D_u}{U_*} \approx \frac{1}{\kappa},$$

from which the expression (3) follows. In turn, this means that the scale $z' = z - D_u$ characterizes the spatial spectra and co-spectra of turbulent fluctuations above urban canopy (see spectral analysis of such turbulent flows in papers [9] and [10]). Since there is no clear boundary between the urban canopy and the atmosphere above it, we are entitled to assume that the spectral characteristics of turbulence in these two layers under the influence of disturbances from streamlined objects change in a such way, that one can establish the similarity between them using length scales D_u, $\langle h \rangle$ and z. In this case, D_u acts as a length scale that has already absorbed all the geometric parameters of the urban environment. The simplest, and in some cases successful, technique to obtain one generalized length scale from several is inverse interpolation of scales with some weighting coefficients. This is equivalent to averaging with the weights of the corresponding wave numbers and it approximates, for example, the co-spectrum-weighted

average wave numbers proportional to the inverse scale of the Prandtl mixing length, which, in fact, we need to calculate the turbulent viscosity and diffusion coefficients (see [9]). Since the RANS model with only TKE prognostic equation in its most simplified form needs only one scale of length l_T, and all others are considered proportional to it, we will use the following approximation:

$$\frac{1}{l_T} = \frac{1}{c_z z} + \frac{1}{c_D D_u}, \tag{20}$$

where c_z and c_D are undefined functions depending on two dimensionless quantities $z/\langle h\rangle$ and $\langle h\rangle/D_u$. We can impose the following constraints these functions need to satisfy:

$$l_T \to \kappa z, \qquad \text{for} \quad z \to 0, \tag{21}$$

and

$$l_T = \kappa(\langle h\rangle - D_u), \qquad \text{for} \quad z = \langle h\rangle\,. \tag{22}$$

The constraint (21) means that the length scale we obtain should transition into the usual near-wall turbulence scale κz when approaching the ground surface, and the constraint (22) ensures continuity of the length scale as we transition from the canopy layer to the external flow above it.

Conditions (21) and (22) allow us to set $c_z = \kappa$, and also impose a restriction on the type of function $c_D = c_D(z/\langle h\rangle, \langle h\rangle/D_u)$. To calculate l_T we use the following linear with respect to $z' = (\langle h\rangle - z)/D_u$ expression for c_D:

$$c_D = \kappa\left[\left(\frac{\langle h\rangle}{D_u}\right)^2 - \frac{\langle h\rangle}{D_u}\right] + c_1 z' f_1\left(\frac{\langle h\rangle}{D_u}\right), \tag{23}$$

where c_1 is a constant, f_1 is a function, which depends on only one variable $\langle h\rangle/D_u$. Note that in the expansion (23) the first term is determined by the constraint (22). Let us consider the case $f_1 \equiv 1$, which allows us to obtain an expression for l_T containing only one constant c_1. The dependence of the turbulent scale on the height $z/\langle h\rangle$ at different values of c_1 is shown in Fig. 3a. Figure 3b shows the changes of turbulent scale l_T with varying displacement height D_u value. It can be seen that the scale l_T approaches κz in the entire layer $0 < z < h$ when D_u goes to zero, i.e. in the case when the influence of the canopy is negligible.

The function (23) allows us to set a maximum in the value of l_T at some height from the surface inside the canopy, $z < h$, and at that corresponds to a polynomial of small degree from the arguments $z/\langle h\rangle$ and $D_u/\langle h\rangle$.

The determination of additional parameters c_1 and, in the general case, of the function f_1 requires large set of LES experiments for different configurations of the roughness elements within urban canopy. Further, when setting l_T and evaluating RANS model with LES data, for simplicity we will assume that $c_1 = 0.4$ (the curve is highlighted in black in Fig. 3).

6. RANS Results and Their Comparison with LES Data

Here we compare only the velocity and scalar concentration profiles in the equilibrium state, since they are what the simplified RANS-models are supposed to reproduce. In addition, in authors opinion, comparing the TKE in one-dimensional RANS and the TKE in LES is incorrect, especially within the roughness layer, for a number of reasons, such as: the presence of passive large-scale structures in LES that do not carry momentum and concentration vertically;

the presence of a two-dimensional dispersion component of the time-averaged horizontal velocity associated with the flow's envelopment of obstacles; and the substantial anisotropy of turbulent fluctuations, which is only implicitly accounted for this type of the flow in one dimensional RANS model. A comparison of the TKE in LES and the TKE in RANS would require an interpretation which deserves a separate study.

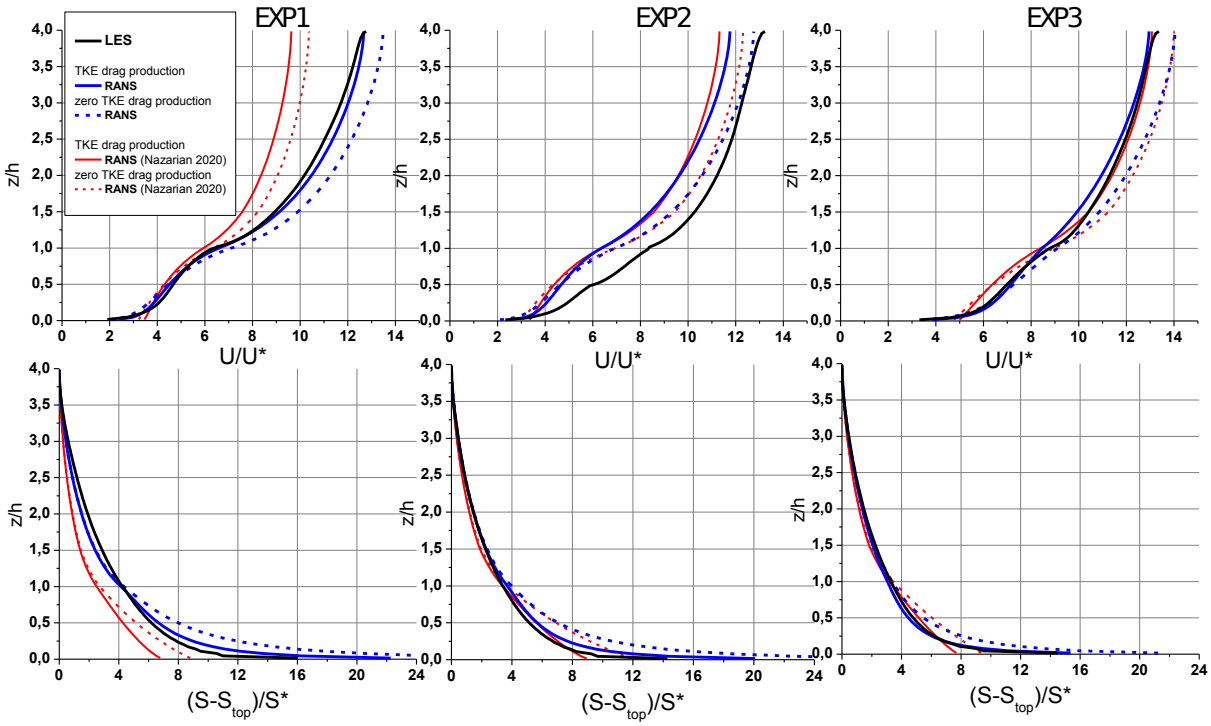


Figure 4. Comparison of LES and RANS results. The black curves are LES results. Blue curves – profiles calculated using the turbulent length scale shown in Fig. 3; red curves – according to the turbulent length scale model from [19]. The solid lines are RANS with TKE production term due to form drag (2); the dashed lines correspond to the ones without this additional production term. Left, center and right columns correspond to EXP1, EXP2 and EXP3 configurations respectively

Figure 4 shows the results of calculations using the RANS models in comparison with the LES data (black curves). Profiles of the normalized velocity U/U_* are plotted in the upper row, and the lower row corresponds to the normalized scalar concentration defect $(S - S_{top})/S_*$. The blue curves show profiles calculated using the turbulent length scale described in section 5, and the red curves show profiles obtained using the turbulent length scale model proposed in [19]. The solid lines show the results obtained using the full TKE balance model, which includes TKE production due to form drag (2), and the dashed curves show the results RANS simulations in which this term in the TKE balance equation (13) is omitted. The plots are arranged by columns, each representing different geometry configurations of urban canopy.

Let us highlight the most characteristic differences between the results obtained with the model proposed in this study and the model [19]. Near the ground surface, the velocity and scalar

gradients reproduced by the proposed model increase towards the surface, which is a reflection of the correct asymptotic of the near-wall flow observed also in LES results. Inside the $0 < z < h$ layer, the profiles calculated using the proposed L_T scale have a regular curved shape, reflecting the absence of excess turbulent viscosity and diffusion. In most cases (except for the velocity profile in EXP2), adding extra TKE production associated with turbulent flow interactions with buildings improves simulation results. In the [19] model, all profiles appeared excessively mixed within the $0 < z < h$ layer, and no near-ground asymptotic are observed. These exact features can also be seen in the authors original paper (see mean velocity in Figs. 12 and 14 from [19]). Note that here we have made more rigid requirements to RANS models by using a low density of the buildings configurations (plane density for the EXP1 is equal to $6.25 \cdot 10^{-2}$ – lower limit of what was modeled in [19], and EXP2 and EXP3 has even lower plane density), so that the defects of turbulent closure in reproducing velocity profiles are more appreciable than the influence of form drag. Second, we have considered transport of scalars for which this volume force is absent. The manifestation of asymptotic of the logarithmic layer near the surface in RANS simulations allows us to hope that with such a model it is possible to refine modelling of heat and moisture exchange with the ground surface in urban canopies, for which Monin-Obukhov similarity theory is of most importance.

Both RANS models equally incorrectly reproduced the velocity profile in EXP2. The profiles obtained in RANS are shifted relative to the LES profile to the left, that is, the canopy modelled by RANS creates a greater aerodynamic resistance to external flow than the explicitly resolved canopy in LES experiments. In terms of approximations (3), this means that the roughness parameter z_{0u} of the model surface was overestimated. In section 4, we noted that, judging from the comparison of EXP2 and EXP3 results, the upper part of the objects is more responsible for the efficiency of momentum exchange between the external flow and the surface as a whole. We do not have a ready recipe for taking this effect into account in the RANS models, but we hope that this problem can also be solved by involving dimensional considerations.

Conclusions

In this paper, we performed numerical experiments with LES model of neutrally stratified flow over an urban-type surface with a low density of roughness elements. The LES was used as a benchmark for evaluation of multilayer local one-dimensional RANS models of urban canopy. Due to the development of supercomputer technology, such models may, in the near future, become units of weather prediction and climate atmospheric models and will allow detailing weather forecasts and assessing possible risks during climatic changes in urban environments of specific megapolises.

In order to construct a one-dimensional RANS model of the urban canopy, we deliberately chose the simplest possible approach, being limited to the standard K-l turbulence closure and did not adjust its constants for specific type of the flow. However, the choice of simple turbulent length scale, which logically follows from the dimensional analysis and trivial considerations about the spectral structure of urban canopy turbulence, gives reassuring results. The main "null hypothesis" was the assumption that the wide set of key length scales necessary to generalize the geometric characteristics of the urban environment is already contained in the spectral structure of the turbulent flow over the urban canopy. The main integral and "measurable" parameter, which reflects the impact of the surface geometry as a whole on the spatial spectra, is the displacement height D_u. Based on this hypothesis, we proposed a parameterization of

the turbulent length scale within the urban canopy, which includes D_u as one of the main dimensional parameters, and does not incorporate the analysis of the geometry of objects as such.

The proposed turbulent length scale parameterization allows simple tuning of RANS models with just one constant without losing physically valid asymptotics near the ground surface and near the upper boundary of the urban canopy. More rigorous validation and tuning of the proposed model requires consideration of a wide range of surface configurations, which will require additional supercomputer computational experiments with eddy-resolving turbulent flow models (LES and DNS).

Identified defects of the proposed closure are indicated in section 6. In particular, our model, as well as the model [19], was unable to correctly reproduce the velocity of the external flow for canopy with variable heights of roughness objects and overestimates the aerodynamic drag of such a surface as a whole. Additional LES calculations with surfaces of this type are needed to elucidate the reasons for this defect and to correct it. Here we have considered only neutrally stratified flows, however there is a greater interest for flows which are affected by buoyancy forces, characteristic for the atmospheric boundary layer, as well as – processes of heat and moisture exchange with the urban surface. Such flows over the urban canopy can also be studied with LES-models (see, for example, [10]). Comparison of the results of multilayer RANS models with the data of eddy-resolving simulations is the most straightforward way to evaluate and improve such turbulence closures.

Acknowledgements

Numerical experiments, their analysis and the development of new parameterization for RANS of urban layer (sections 3, 4 and 5) was performed with financial support of the Russian Science Foundation, grant 21-71-30023.

LES and RANS model code development and implementation of numerical algorithms for solving of LES and RANS equations (sections 1 and 2) was supported by INM RAS Division of Moscow Center for Fundamental and Applied Mathematics (agreement No. 075-15-2019-1624).

References

1. Bardina, J., Ferziger, J., Reynolds, W.: Improved subgrid-scale models for large-eddy simulation. In: 13th fluid and plasmadynamics conference. p. 1357 (1980). https://doi.org/10.2514/6.1980-1357

2. Burchard, H.: Applied turbulence modelling in marine waters, vol. 100. Springer Science & Business Media (2002)

3. Cheng, H., Castro, I.P.: Near wall flow over urban-like roughness. Boundary-Layer Meteorology 104(2), 229–259 (2002). https://doi.org/10.1023/A:1016060103448

4. Coceal, O., Thomas, T., Castro, I., Belcher, S.: Mean flow and turbulence statistics over groups of urban-like cubical obstacles. Boundary-Layer Meteorology 121(3), 491–519 (2006).

https://doi.org/10.1007/s10546-006-9076-2

5. Dupont, S., Otte, T.L., Ching, J.K.: Simulation of meteorological fields within and above urban and rural canopies with a mesoscale model. Boundary-Layer Meteorology 113(1), 111–158 (2004). https://doi.org/10.1023/B:BOUN.0000037327.19159.ac

6. Germano, M., Piomelli, U., Moin, P., Cabot, W.H.: A dynamic subgrid-scale eddy viscosity model. Physics of Fluids A: Fluid Dynamics 3(7), 1760–1765 (1991). https://doi.org/10.1063/1.857955

7. Glazunov, A., Rannik, Ü., Stepanenko, V., et al.: Large-eddy simulation and stochastic modeling of lagrangian particles for footprint determination in the stable boundary layer. Geoscientific Model Development 9(9), 2925–2949 (2016). https://doi.org/10.5194/gmd-9-2925-2016

8. Glazunov, A.V.: Numerical simulation of turbulence and transport of fine particulate impurities in street canyons. Numerical methods and programming 19, 17–37 (2018)

9. Glazunov, A.: Numerical modeling of turbulent flows over an urban-type surface: Computations for neutral stratification. Izvestiya, Atmospheric and Oceanic Physics 50(2), 134–142 (2014). https://doi.org/10.1134/S0001433814010034

10. Glazunov, A.: Numerical simulation of stably stratified turbulent flows over an urban surface: Spectra and scales and parameterization of temperature and wind-velocity profiles. Izvestiya, Atmospheric and Oceanic Physics 50(4), 356–368 (2014). https://doi.org/10.1134/S0001433814040148

11. Glazunov, A.: Numerical simulation of stably stratified turbulent flows over flat and urban surfaces. Izvestiya, Atmospheric and Oceanic Physics 50(3), 236–245 (2014). https://doi.org/10.1134/S0001433814030037

12. Kanda, M., Kanega, M., Kawai, T., et al.: Roughness lengths for momentum and heat derived from outdoor urban scale models. Journal of Applied Meteorology and Climatology 46(7), 1067–1079 (2007). https://doi.org/10.1175/JAM2500.1

13. Kanda, M., Kawai, T., Moriwaki, R., et al.: Comprehensive outdoor scale model experiments for urban climate (COSMO). In: Proc., 6th Int. Conf. on Urban Climate. pp. 270–273 (2006). https://doi.org/10.1023/A:1016060103448

14. Krayenhoff, E.S., Jiang, T., Christen, A., et al.: A multi-layer urban canopy meteorological model with trees (BEP-Tree): Street tree impacts on pedestrian-level climate. Urban Climate 32, 100590 (2020). https://doi.org/10.1016/j.uclim.2020.100590

15. Krayenhoff, E., Christen, A., Martilli, A., Oke, T.: A multi-layer radiation model for urban neighbourhoods with trees. Boundary-layer meteorology 151(1), 139–178 (2014). https://doi.org/10.1007/s10546-013-9883-1

16. Krayenhoff, E., Santiago, J.L., Martilli, A., et al.: Parametrization of drag and turbulence for urban neighbourhoods with trees. Boundary-Layer Meteorology 156(2), 157–189 (2015). https://doi.org/10.1007/s10546-015-0028-6

17. Martilli, A., Clappier, A., Rotach, M.W.: An urban surface exchange parameterisation for mesoscale models. Boundary-layer meteorology 104(2), 261–304 (2002). https://doi.org/10.1023/A:1016099921195

18. Morinishi, Y., Lund, T., Vasilyev, O., Moin, P.: Fully conservative higher order finite difference schemes for incompressible flows. J. Comp. Phys. 143, 90–124 (1998). https://doi.org/10.1006/jcph.1998.5962

19. Nazarian, N., Krayenhoff, E.S., Martilli, A.: A one-dimensional model of turbulent flow through "urban" canopies (MLUCM v2.0): updates based on large-eddy simulation. Geoscientific Model Development 13(3), 937–953 (2020). https://doi.org/10.5194/gmd-13-937-2020

20. Santiago, J., Martilli, A.: A dynamic urban canopy parameterization for mesoscale models based on computational fluid dynamics Reynolds-averaged Navier–Stokes microscale simulations. Boundary-layer meteorology 137(3), 417–439 (2010). https://doi.org/10.1007/s10546-010-9538-4

21. Xie, Z., Castro, I.P.: LES and RANS for turbulent flow over arrays of wall-mounted obstacles. Flow, Turbulence and Combustion 76(3), 291–312 (2006). https://doi.org/10.1007/s10494-006-9018-6