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

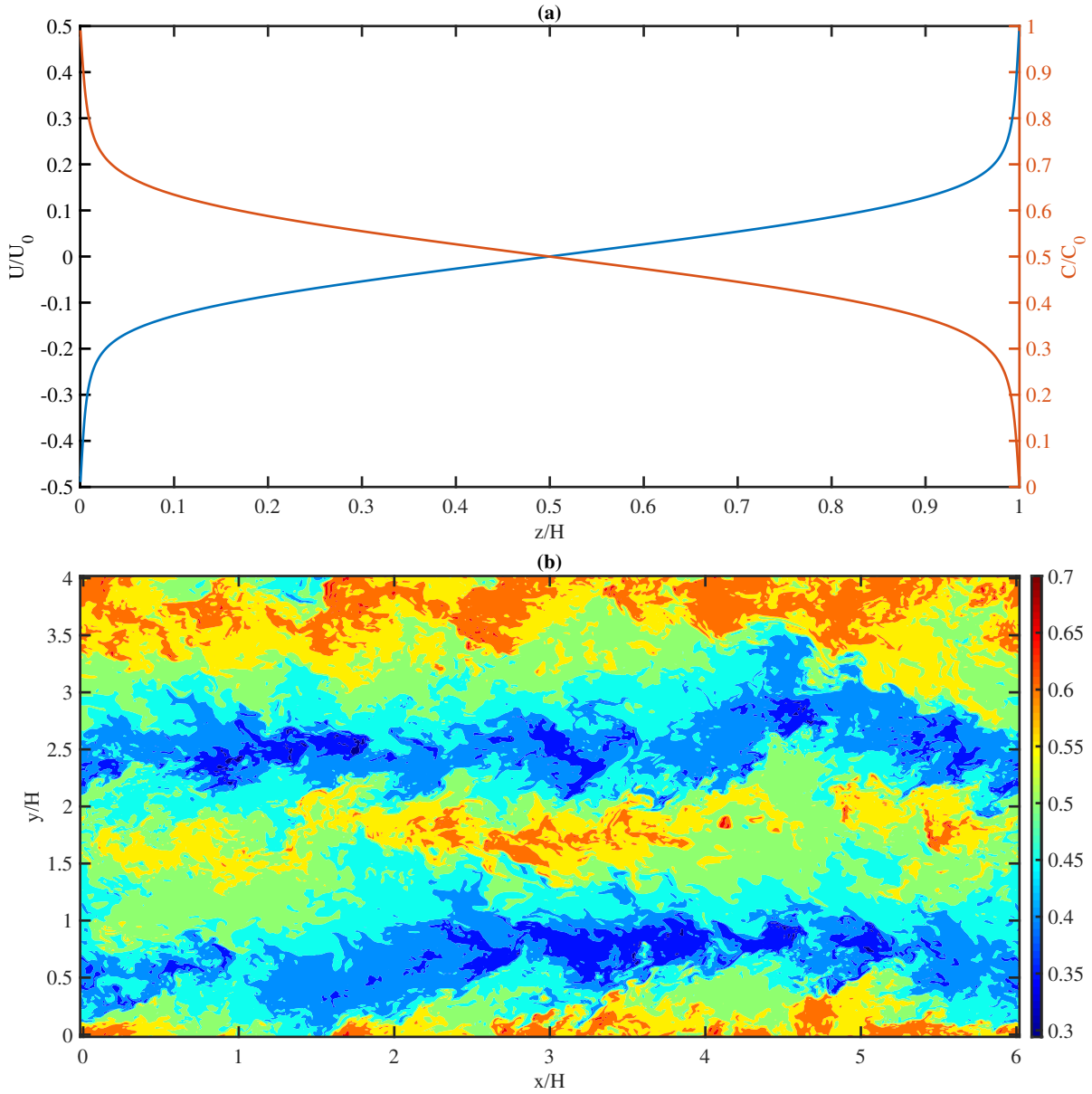

Figure 1. (a) The normalized mean streamwise velocity (blue) and normalized tracer concentration profiles (red); (b) instantaneous normalized tracer concentration in the $-xy$ plane at $z/H = 0.5$ for neutrally stratified turbulent Couette flow

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

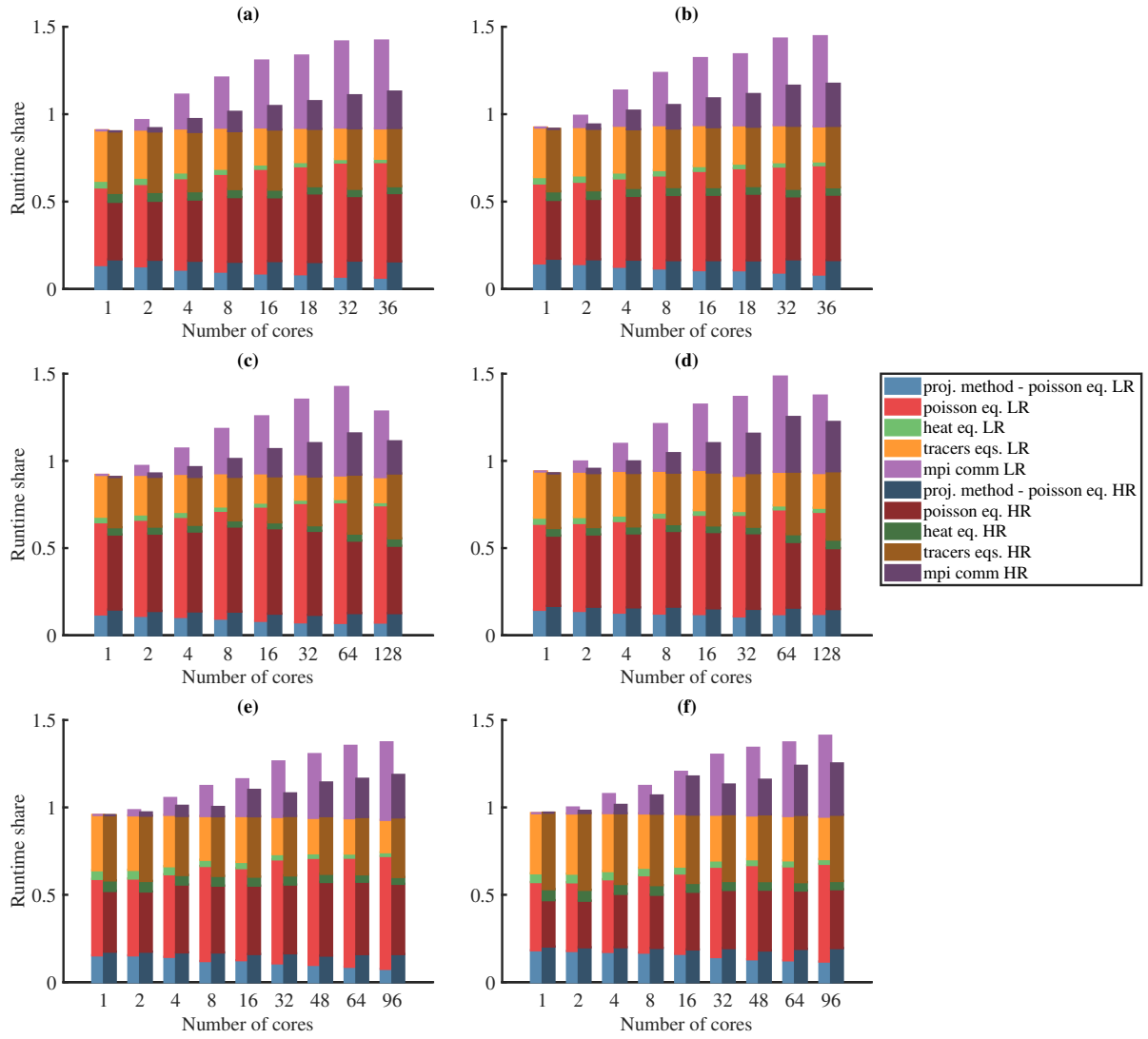

Figure 2. Run-time shares of model components for LR and HR cases on Intel Xeon Gold 1640 (a, b), AMD Rome 7H12 (c, d) and Kunpeng 920 (e, f) CPUs for 2-nd order (a, c, e) and 4-th order accurate finite-difference schemes (b, d, f)

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.

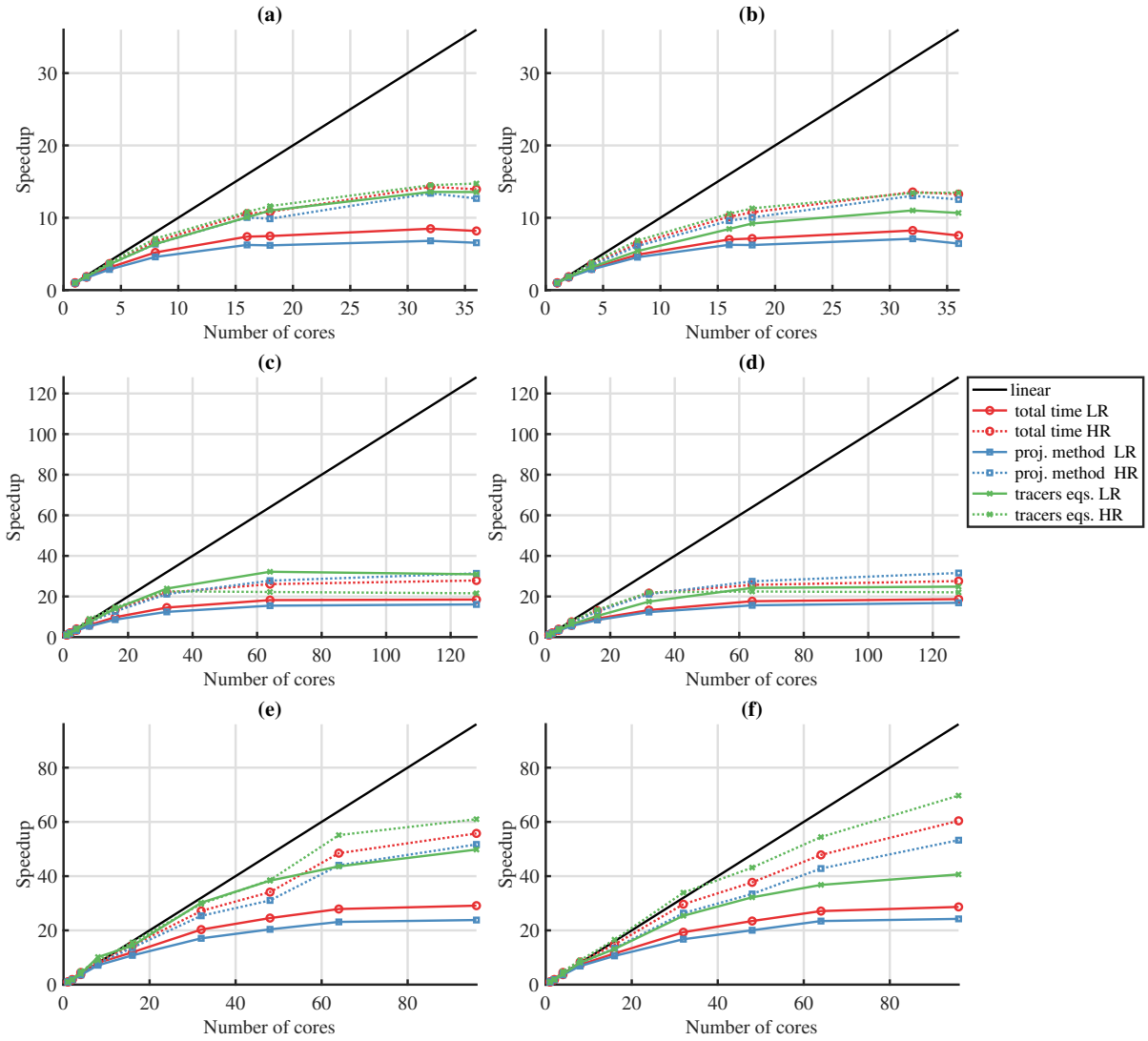

Figure 3. Speedup of DNS model and its components for LR and HR cases on Intel Xeon Gold 1640 (a, b), AMD Rome 7H12 (c, d) and Kunpeng 920 (e, f) CPUs for 2-nd order (a, c, e) and 4-th order accurate finite-difference schemes (b, d, f)

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

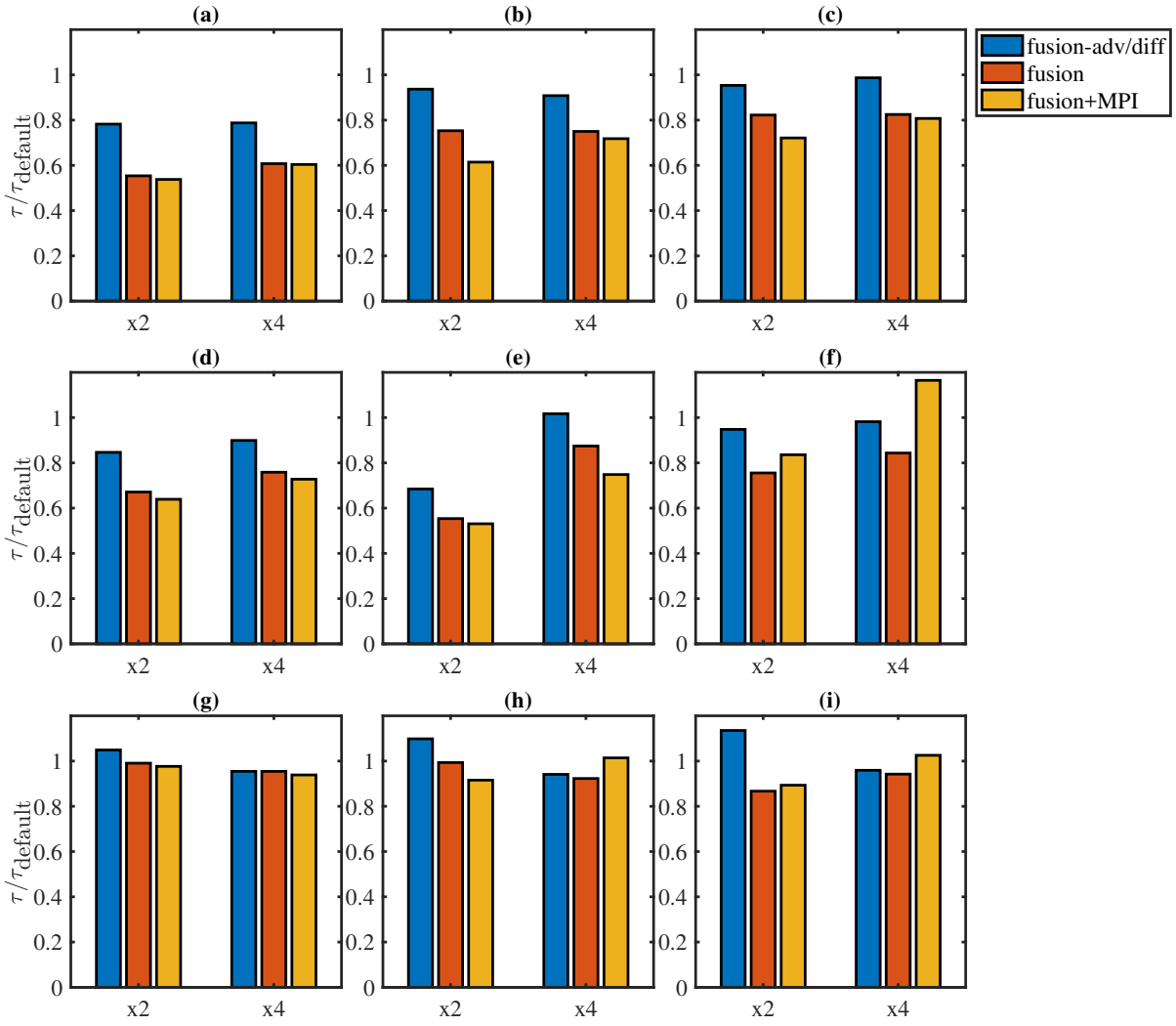

Figure 4. Normalized run-time (relative to default implementation) of the optimized tracer transport model component for LR case on Intel Xeon Gold 1640 (a, b, c), AMD Rome 7H12 (d, e, f) and Kunpeng 920 (g, h, i) CPUs for 2-nd order (`'x2'`) and 4-th order accurate finite-difference schemes (`'x4'`).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

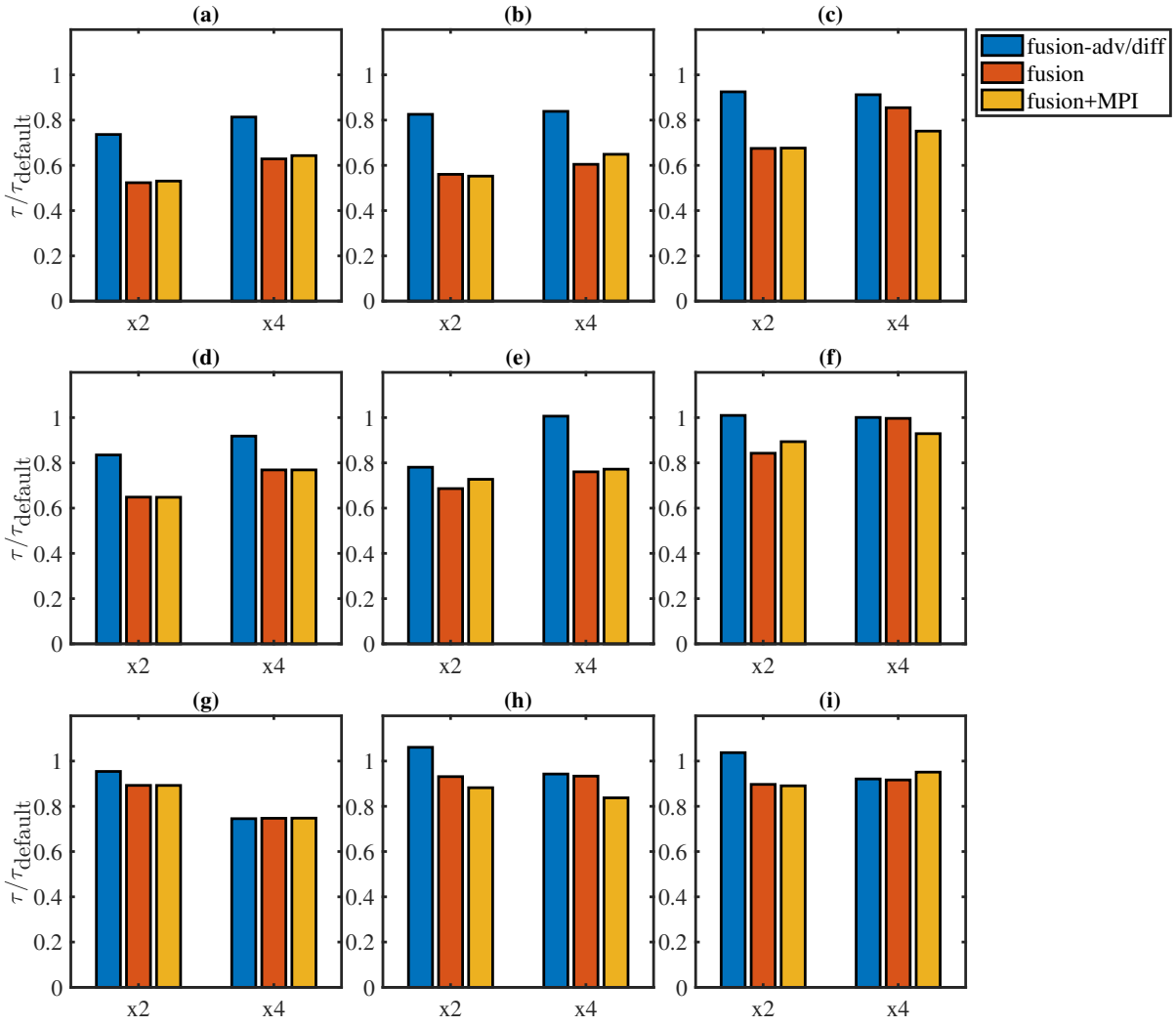

Figure 5. Same as in Fig. 4, but for HR case

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