ISSN 1995-0802, Lobachevskii Journal of Mathematics, 2018, Vol. 39, No. 4, pp. 524–532. © Pleiades Publishing, Ltd., 2018.

Parallel Algorithm of the NOISEtte Code for CFD and CAA Simulations

A. Gorobets*

(Submitted by V. V. Voevodin)

Keldysh Institute of Applied Mathematics, Russian Academy of Sciences, Moscow, 125047 Russia

Received November 15, 2017

Abstract—This paper describes the parallel algorithm of the NOISEtte code for computational fluid dynamics and aeroacoustics simulations. It is based on a family of higher-accuracy numerical schemes for unstructured hybrid meshes. The multilevel MPI + OpenMP parallelization is described in detail. Performance results are presented for various supercomputers and applications.

DOI: 10.1134/S1995080218040078

Keywords and phrases: *Parallel CFD, MPI, OpenMP, turbulent flows, aerodynamics, aeroacoustics.*

1. INTRODUCTION

The evolution of high-performance computing (HPC) systems towards exascale computing imposes challenging problems related with extreme levels of parallelism. The constantly growing numbers of cores per processor and processors in a system require highly scalable parallel algorithms.

Multicore CPUs and manycore massively-parallel accelerators, such as Intel MIC (Many Integrated Core), represent a shared-memory MIMD parallelism with SIMD extensions. The number of cores per device keeps growing and each core can handle multiple threads (Intel Xeon 8180—28 cores, 56 threads; AMD EPYC 7601—32 cores, 64 threads; Intel Xeon Phi 7290—72 cores, 288 threads). This trend makes the shared-memory parallelization indispensable.

At the same time, the number of double precision floating point (FP64) operations per tact the core can perform is growing as well, mainly due to the widening of vector registers. Therefore, the vectorization is becoming more important. However, the memory bandwidth is not growing that fast. For instance, each core of Intel Xeon 8180 can do 32 FP64 operations per tact with its two FMA (Fused Multiply-Add) AVX-512 (512-bit SIMD extension) units. Its overall peak performance (above 2 TFLOPS with 120 GB/s bandwidth) is nearly 10 times bigger compared to the previous CPU generations while the memory bandwidth has increased less than twice (e.g., Intel Xeon E5-2680v3—240 GFLOPS, 68 GB/s).

This quickly growing gap between the peak performance and the memory bandwidth is a sad situation for most of computational fluid dynamics (CFD) algorithms which are typically memory-bound. Therefore, the great majority of CFD algorithms can potentially reach only a minor fraction of the peak performance (similarly to the HPCG Benchmark [1]), except for algorithms based on very expensive high-order schemes, such as [2], that can demonstrate sustained performance near 50% of peak.

In accordance with these trends the present work is focused on the MPI + OpenMP multilevel MIMD parallelization that allows to engage efficiently a large number of cores. Additionally, the vectorization by means of compiler directives is used at the lower parallelization level. However, it currently plays a secondary role since the memory bandwidth is by far insufficient to gain a lot with SIMD extensions.

*E-mail: andrey.gorobets@gmail.com

Such a parallelization that combines distributed and shared-memory models is a widely-used approach for CFD solutions. Examples of thereof can be found, for instance, in [3] for rarified gas flows, in [4] for incompressible flows, in [5] for multi-physics simulations. In the present paper, the parallel algorithm of the NOISEtte code [6] is described. It is a code for CFD and aeroacoustics (CAA) simulations of compressible flows using high-accuracy schemes on unstructured meshes [7].

The rest of the paper is organized as follows. The mathematical model and the numerical method are briefly outlined in Sec. 2. The parallel algorithm is presented in Sec. 3. Sec. 4 illustrates the parallel performance.

2. MATHEMATICAL MODEL AND NUMERICAL METHOD

2.1. Mathematical Model

The Navier–Stokes equations for a compressible ideal gas are used for modeling of a turbulent flow. This basic system of equations is complemented with necessary source terms and additional equations for turbulence modeling within one of the following approaches: Direct Numerical Simulation (DNS), Reynolds-Averaged Navier–Stokes (RANS), including [8, 9]; Large Eddy Simulation (LES), including [10, 11]; hybrid RANS-LES approaches, such as Detached Eddy Simulation (DES) [12, 13].

2.2. Numerical Method

A family of computationally-cheap higher-accuracy schemes based on a quasi-1D reconstruction is used for the spatial discretization of the governing equations on an unstructured hybrid mesh. The set of schemes includes the edge-based vertex-centered schemes [7] and the cell-centered schemes [14].

The numerical fluxes through faces of control volumes are computed using one of Riemann problem solvers [15], depending on the flow parameters, including Roe for subsonic flows, HLLC for supersonic flows.

The governing equations are discretizated in time using either an explicit Runge–Kutta scheme (up to 4-th order) or an implicit Newton-based scheme (up to 2-nd order). The preconditioned Bi-CGSTAB [16] is used for the Jacobi system in case of the implicit scheme. The solver is composed of three operations: the sparse matrix-vector product ($SpMV$), the linear combination of two vectors, and the dot product. The compressed sparse row (CSR) block representation of a sparse matrix is used.

The set of boundary conditions for exterior faces includes no-slip conditions for impermeable solid walls (isothermal, adiabatic, constant heat flux), Dirichlet's conditions, characteristic inflow/outflow conditions [17].

3. PARALLEL ALGORITHM

3.1. Algorithm Outline

The computing domain given by an unstructured mesh consists of the set of *cells* (control volumes), the set of internal *faces* (that have two adjacent cells), and the set of *exterior faces* (that have only one adjacent cell). In the vertex-centered formulation the cells correspond to the mesh nodes, the faces correspond to the mesh edges (exterior faces are constructed around boundary nodes from parts of exterior faces of mesh elements). In the cell-centered case the cells coincide with the mesh elements. The *adjacency graph* represents couplings between the cells: the graph nodes correspond to the cells and the graph edges correspond to the common faces between cells. The list of faces stores for each face the indices of its adjacent cells. The *inverse topology* stores for each cell the list of its faces and its adjacent cells.

The set of primary variables (such as density, velocity components, pressure) is defined in each cell. The simulation algorithm consists in successive advancing in time by sufficiently small time integration steps. The new values of variables in cells are determined at every time step by computing the fluxes through the faces of the cells. To that end, the variables stored in cells must be reconstructed at the centers of faces. Such values from both sides of the face centers, so called left and right values, are determined with a quasi-1D reconstruction [7, 14] (see Fig. 1).

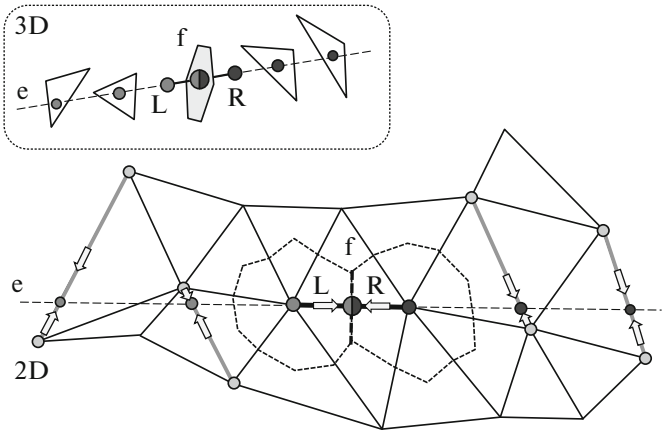

Fig. 1. Reconstruction of values at the center of the face f. The interpolation constructions around the edge line e produce the values along this line in order to reconstruct values from the sides of the left (L) and right (R) cells.

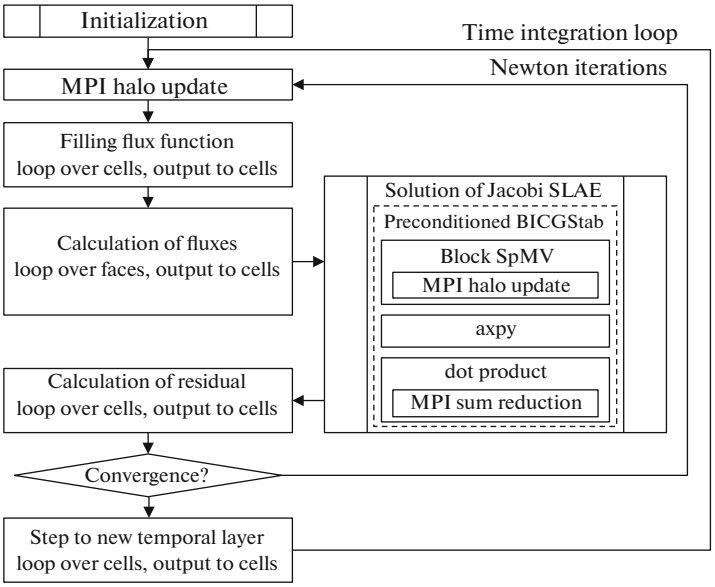

Fig. 2. Simplified diagram of the time step algorithm (an implicit Newton-based scheme).

Then, the Riemann problem is solved at each face in order to determine the flux using the left and right values at the face centers. Finally, this fluxes are applied into cells giving the values at the new temporal layer.

The simplified algorithm representation is shown in Fig. 2. The calculation of fluxes is composed of several operations, including calculation of convective fluxes, viscous fluxes, turbulence model contributions, source terms, etc. If the implicit scheme is used, the contributions to Jacobi matrix are also computed when processing the set of faces. In this case the Jacobi system is solved in order to determine the values in cells at the next Newton iteration (an iterative process inside each time step).

The algorithm of the time step is composed of several loops either over the set of cells or over the set of faces (Fig. 2). Additionally, boundary conditions are calculated in loops over exterior faces. In the MPI-parallel version halo update operations are required (explained further).

3.2. Distributed-Memory Parallelization

The workload is distributed among supercomputer nodes at the upper level using a traditional domain decomposition approach with a distributed-memory MPI parallelization. Computing domain cells are assigned to subdomains using a partitioning tool (e.g., ParMETIS [18]) that defines for each node of the

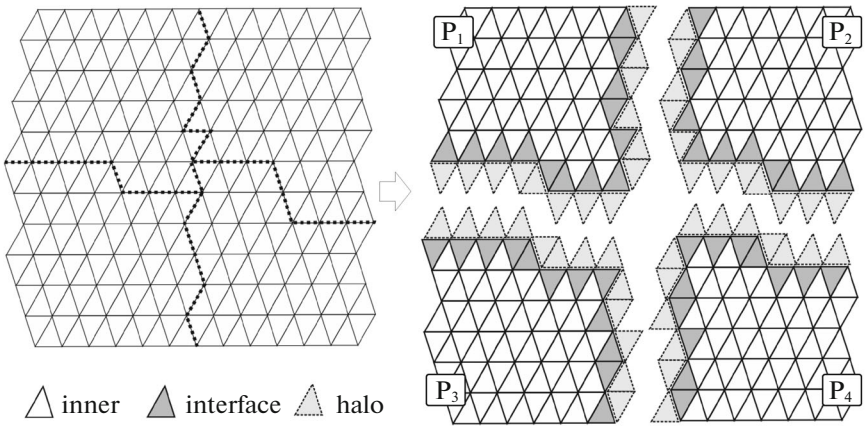

Fig. 3. Example of domain decomposition.

adjacency graph to which subdomain it belongs according to load balancing constraints and minimizing the number of edges cut. The cells inside each of these first-level subdomains are categorized into two sets: the *interface* cells (which are coupled by the spatial scheme stencil with the cells from other subdomains) and the *inner* cells (which are not). Furthermore, each sudbomain is supplemented with the *halo* set that contains the cells from other subdomains which are coupled with at least one cell from this subdmiain. An example of the first-level decomposition is shown in Fig. 3.

Communications between MPI processes are needed in order to update the data in the halo cells. The updated halo is required for operations that involve access to adjacent cells. These operations are: the flux calculation stage of the algorithm (up to 3 levels of adjacency—for a scheme with 2-level interpolation constructions shown in Fig. 1); the SpMV operation inside the Bi-CGSTAB solver (only 1 level of adjacency, since for time-accurate simulations the Jacobi matrix is filled only for the basic low-order part of the scheme). Additionally, a group reduction MPI communication is needed to determine global values, such as the global sum in the dot product used in the SLAE solver, or the global minimum in the calculation of the time step value. Further details about the parallelization technology can be found in [19].

3.3. Shared-Memory Parallelization

The OpenMP standard is used for the shared-memory parallelization. The first-level MPI subdomains are decomposed further into second-level subdomains in order to distribute the workload among parallel threads. The threads operate in a shared memory, hence no halo update is needed. This decomposition is mainly needed for elimination of the data interdependency between the threads that appears in loops over faces which involve writing data to adjacent cells. For instance, if two threads are processing simultaneously two faces of the same cell at the flux calculation stage then the race condition may appear when the threads add the fluxes into the cell (or add their contributions into the same Jacobi matrix row).

To avoid that, a multistage processing of faces is used, similarly to [20]. The interface faces (that are adjacent to cells from different second-level subdomains) are excluded from parallel processing at the first stage. These interface faces and the cells adjacent to that faces undergo the third-level decomposition. After all the inner faces of the first stage are processed the second stage begins. Again, the interface faces that appear at this stage are excluded. This process goes on until no interface faces remain. Such a multilevel decomposition is shown in Fig. 4. Further details and other OpenMP parallelization options can be in [19].

3.4. Configuration of Parallel Execution

The total number of CPU cores engaged in a simulation is composed of the number of MPI processes and the number of OpenMP threads: $P = P_{MPI} \times P_T$. Usually, the distribution of one MPI process per CPU socket (or per NUMA node) is used. The cores of multicore CPUs are engaged with the OpenMP

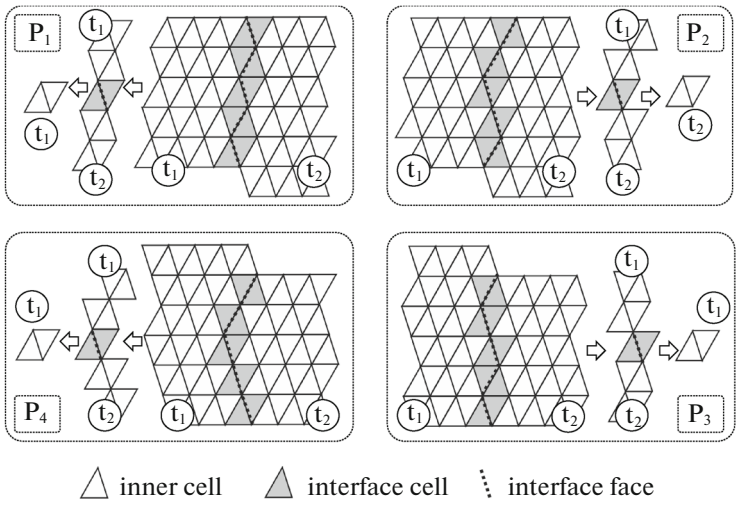

Fig. 4. Decomposition of MPI subdomains for processing with two OpenMP threads.

parallelization. This way the number of communicating processes is reduced P_T times (consequently, the volume of MPI exchanges is reduced). A single MPI process can hardly saturate the network bandwidth of a node. Furthermore, there is a notable overhead related with maintaining cache coherence (and due to NUMA-factors) when multiple CPUs are engaged by a single MPI process. Therefore, running one MPI process per CPU (instead of one per node) is optimal in most cases. More details regarding the balance between MPI and OpenMP can be found in [21].

The thread pinning is used in order to avoid migration of threads between sockets. The affinity (the list of virtual processors allowed) is automatically set for each thread according to particular execution conditions and taking Hyper-threading (HT) into account. The configuration of nodes (how many sockets, cores per socket, threads per core) is derived by parsing "/proc/cpuinfo" file (on Linux). This allows to properly set the affinity and the number of threads even in cases when the cluster nodes are configured differently (some may have HT enabled, some not, etc.). According to tests with two MPI processes per node on a cluster with two 8-core CPUs per node there is a gain in performance up to 40% compared to the default affinity, when any thread can run on any core.

4. PERFORMANCE STUDY

The scalability of the parallel implementation has been demonstrated in various tests based on real applications. The first two speedup tests evaluate the parallel efficiency for currently typical operating conditions using EBR scheme [7] with implicit time integration, MPI + OpenMP mode with one MPI process per socket.

Fig. 5 (left) shows the speedup obtained on MVS-10P supercomputer (2× 8-core Intel Xeon E5-2690 per node, FDR InfiniBand) for the simulation of the flow around a backward-facing step in a transonic flow ($M = 0.9$, $\mathrm{Re} = 7.2 \times 10^6$, mesh with 13 million nodes). The instantaneous view of the flow field kindly given by A. Duben is shown in Fig. 6. Further details about the simulation can be found in [22].

A similar plot obtained on HPC4 supercomputer (2× 12-core Intel Xeon E5-2680v3 per node, FDR InfiniBand) for the simulation of the flow around a helicopter rotor blade ($M = 0.35$ at the blade tip, $\mathrm{Re} = 1.4 \times 10^6$, mesh 22 million nodes) is shown in Fig. 5 (right). The instantaneous view of the flow field kindly given by V. Bobkov is illustrated in Fig. 7.

The OpenMP parallel efficiency is shown in Fig. 8 (left). This test corresponds to the simulation of a round subsonic jet on a coarsened mesh of 1 million nodes. The visualization of the flow kindly given by A. Duben is shown in Fig. 9. These jet simulations are presented in [23]. The speedup in case of the explicit scheme appears to be notably higher. This is because the implicit scheme involves a sparse linear solver which consists of especially memory-bound operations, such as the SpMV (that process a large amount of data with a low computing cost). Therefore, despite these operations are perfectly parallel, the

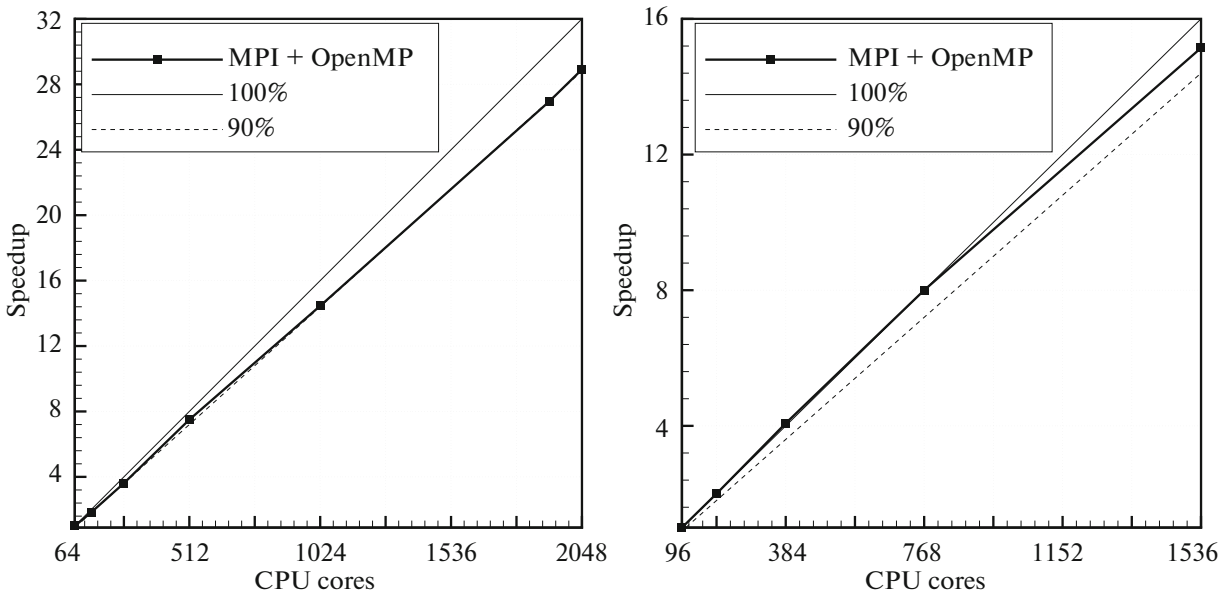

Fig. 5. Speedups: MVS-10P, implicit scheme, IDDES [13], mesh 13 million nodes (left); HPC4, implicit scheme, IDDES, 22 million nodes (right).

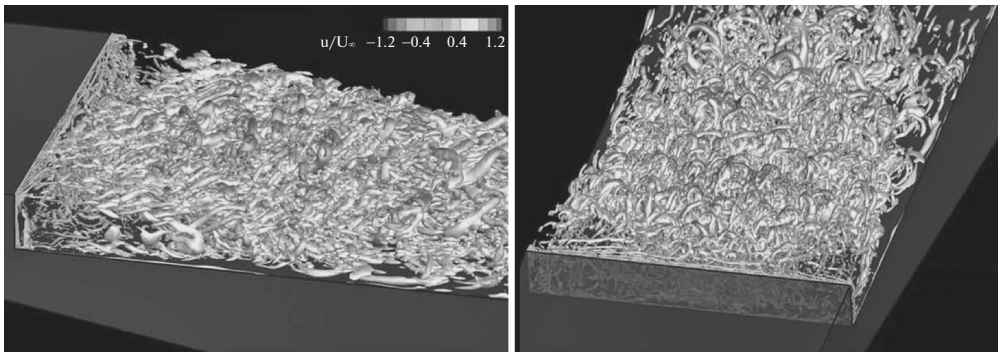

Fig. 6. Instantaneous snapshot (Q-criterion) of the transonic flow around a wedge with a backward-facing step.

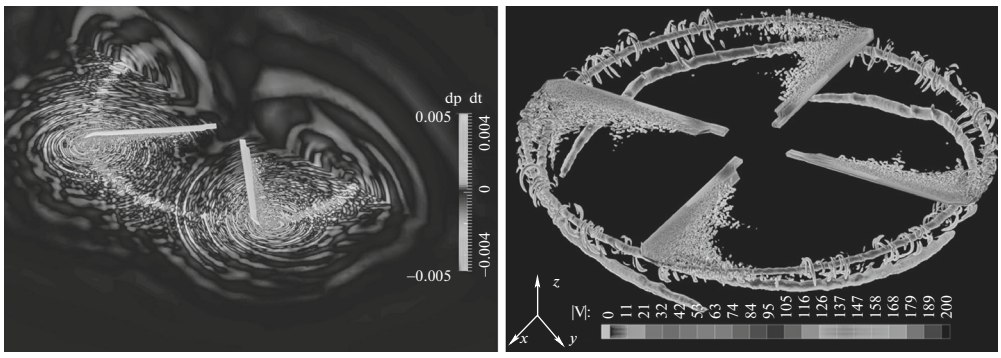

Fig. 7. Flow around a helicopter rotor: acoustic field (left) and turbulence (Q-criterion., right).

solver saturates the memory bandwidth and slows down the acceleration. The weight of this operation grows from 25% in the sequential mode to 44% on 24 cores.

Finally, the plot in Fig. 8 (right) shows the speedup on a bigger mesh (160 million nodes, 1 billion of tetrahedrons) using up to 10240 cores of Lomonosov supercomputer (2×4-core Intel Xeon X5570 perc node, QDR InfiniBand). This amount of resources is far beyond currently typical usage conditions.

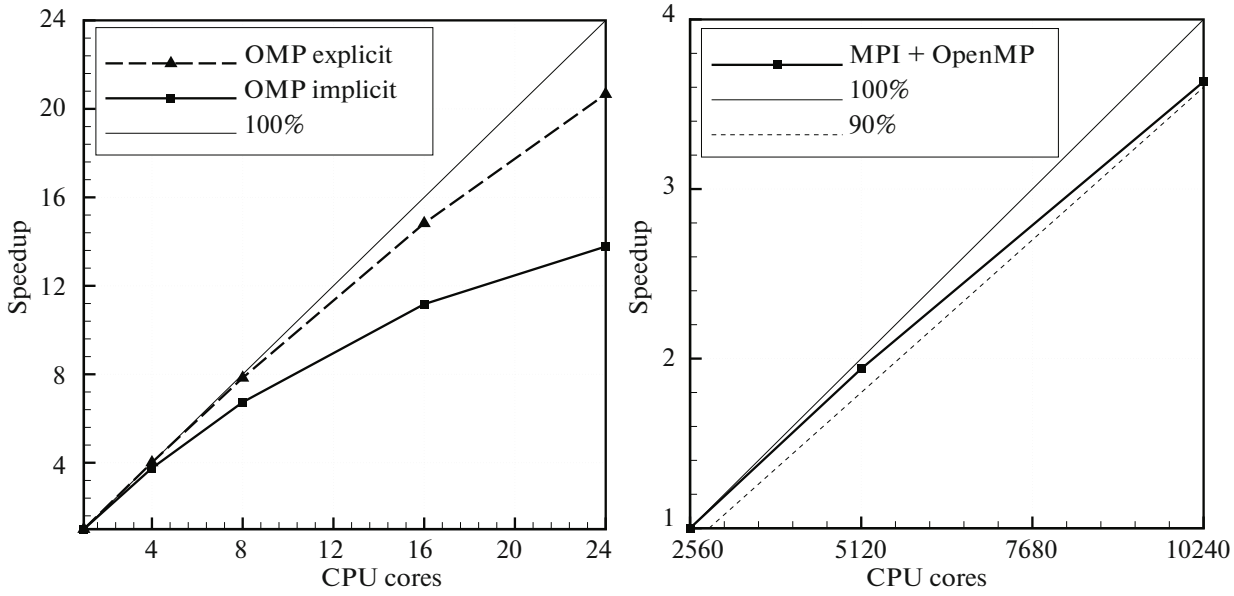

Fig. 8. Speedups: OpenMP mode, 24-core Intel Xeon 8160 CPU, mesh 1.6 million nodes (left); Lomonosov, implicit scheme, DES [12], mesh 160 million nodes (right).

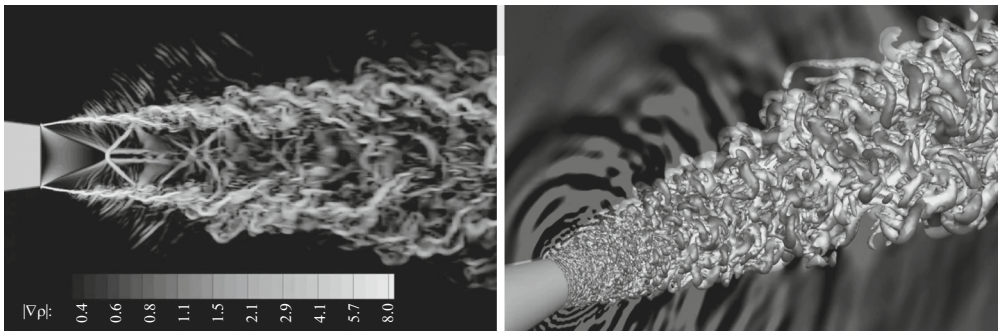

Fig. 9. Round jets: supersonic jet flow in mid-span section (left) and subsonic jet turbulence (Q-crit., right).

The case corresponds to the flow around a 3D cavity studied in [24] but with a mesh that was uniformly refined for the test.

5. CONCLUSIONS

The parallel structure of the NOISEtte CFD/CAA simulation code [6] has been described. The main attention was focused on the MIMD parallelization for both the distributed-memory and the shared-memory models. The multilevel MPI + OpenMP parallelization fits well modern csystems with multicore CPUs and manycore accelerators. The parallel efficiency of doubling the number of cores at typical operational conditions in real applications appears to be above 95%.

The operations that form the algorithm are fully compatible with cSIMD and stream processing. This potentially allows us to use hybrid GPU-based systems. However, the research nature of the code requires fast and easy ways to implement new schemes and methods, to modify existing implementations. Therefore, it appears unduly difficult to maintain both versions of code for CPUs and GPUs. For this reason porting the code to GPUs is postponed until the numerical methods and implementations that are being developed become mature enough and, consequently, the intensity of the research work related with significant code modifications goes down. Meanwhile, a fully-portable heterogeneous computing technology with the MPI + OpenMP + OpenCL parallelization has been developed on a base of a simplified CFD code with a particular cell-centered scheme. This CFD solution for simulation of compressible turbulent flows on a wide range of computing architectures, including

CPUs, MICs, GPUs, is presented in [25]. This approach is to be applied in future to the NOISEtte's computing algorithm.

ACKNOWLEDGMENTS

This work has been funded by the RSF, project 15-11-30039. This work has been carried out using computing resources of the following organizations: the shared research facilities of HPC computing resources at Lomonosov Moscow State University; the federal collective usage center Complex for Simulation and Data Processing for Mega-science Facilities at NRC "Kurchatov Institute", http://ckp.nrcki.ru/; the Joint Supercomputer Center of the Russian Academy of Sciences. The authors thankfully acknowledge these institutions.

REFERENCES

1. J. Dongarra, M. A. Heroux, and P. Luszczek, "HPCG Benchmark: a New Metric for Ranking High Performance Computing Systems," Univ. of Tennessee Computer Science Technical Report UT-EECS-15-736 (Univ. of Tennessee, 2015).
2. P .Vincent, F. Witherden, B. Vermeire, J. S. Park, and A. Iyer, "Towards green aviation with Python at petascale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC'16*. doi 10.1109/SC.2016.1
3. V. A. Titarev, "Application of model kinetic equations to hypersonic rarefied gas flows," Comput. Fluids (2017, in press). doi 10.1016/j.compfluid.2017.06.019
4. A. Gorobets, F. X. Trias, R. Borrell, O. Lehmkuhl, and A. Oliva, "Hybrid MPI + OpenMP parallelization of an FFT-based 3D Poisson solver with one periodic direction," Comput. Fluids **49**, 101–109 (2011). doi 10.1016/j.compfluid.2011.05.003
5. M. Vazquez, G. Houzeaux, S. Koric, A. Artigues, J. Aguado-Sierra, R. Aris, D. Mira, H. Calmet, F. Cucchietti, H. Owen, A. Taha, and J. M. Cela, "Alya: towards exascale for engineering simulation codes," arXiv:1404.4881 (2014).
6. I. V. Abalakin, P. A. Bakhvalov, A. V. Gorobets, A. P. Duben, and T. K. Kozubskaya, "Parallel research code NOISEtte for large-scale CFD and CAA simulations", Numer. Methods Programm. **13**, 110–125 (2012).
7. P. A. Bakhvalov and T. K. Kozubskaya, "Construction of edge-based 1-exact schemes for solving the Euler equations on hybrid unstructured meshes," Comput. Math. Math. Phys. **57**, 680–697 (2017). doi 10.1134/S0965542517040030
8. P. R. Spalart and S. R Allmaras, "A one-equation turbulence model for aerodynamic flows," in *Proceedings 30th Aerospace Science Meeting, Reno, Nevada, May 20–22, 1992,* AIAA Paper 92-0439.
9. F. R. Menter, "Two-equation eddy-viscosity turbulence models for engineering applications," AIAA J. **32**, 1598–1605 (1994).
10. F. Nicoud and F Ducros, "Subgrid-scale stress modelling based on the square of the velocity gradient tensor," Flow, Turbulence Combust. **62**, 183–200 (1999).
11. F. X. Trias, D. Folch, A. Gorobets, and A. Oliva, "Building proper invariants for eddy-viscosity subgrid-scale models," Phys. Fluids **27**, 065103 (2015). doi 10.1063/1.4921817
12. M. L. Shur, P. R. Spalart, M. Kh. Strelets, and A. K. Travin, "A hybrid RANS-LES approach with delayed-DES and wall-modeled LES capabilities," Int. J. Heat Fluid Flow **29**, 1638–1649 (2008). doi 10.1016/j.ijheatfluidflow.2008.07.001
13. M. L. Shur, P. R. Spalart, M. Kh. Strelets, and A. K. Travin, "An enhanced version of DES with rapid transition from RANS to LES in separated flows," Flow, Turbulence Combust. **95**, 709–737 (2015). doi 10.1007/s10494-015-9618-0
14. P. A. Bakhvalov and T. K. Kozubskaya, "Cell-centered quasi-one-dimensional reconstruction scheme on 3D hybrid meshes," Math. Models Comput. Simul. **8**, 625–637 (2016). doi 10.1134/S2070048216060053
15. E. F. Toro, *Riemann Solvers and Numerical Methods for Fluid Dynamics* (Springer, Berlin, Heidelberg, 2009). doi 10.1007/b79761
16. H. A. van der Vorst, "Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems," SIAM J. Sci. Stat. Comput. **13**, 631–644 (1992). doi 10.1137/0913035
17. Ch. Hirsch, *Numerical Computation of Internal and External Flows,* Vol. 2: *Computational Methods for Inviscid and Viscous Flows* (Wiley, New York, 1990).
18. D. LaSalle and G. Karypis, "Multi-threaded graph partitioning," in *Proceedings of IEEE 27th International Parallel and Distributed Processing Symposium IPDPS, 2013,* pp. 225–236.
19. A. V. Gorobets, "Parallel technologies for solving CFD problems using high-accuracy algorithms," Comput. Math. Math. Phys. **55**, 638–649 (2015). doi 10.1134/S0965542515040065

20. R. Aubry, G. Houzeaux, M. Vazquez, and J. M. Cela, "Some useful strategies for unstructured edge-based solvers on shared memory machines," Int. J. Numer. Methods Eng. **85**, 537–561 (2010). doi 10.1002/nme.2973
21. A. V. Gorobets, "The technology of large-scale CFD simulations," Math. Models Comput. Simul. **8**, 660–670 (2016). doi 10.1134/S2070048216060089
22. B. Dankov, A. Duben, and T. Kozubskaya, "Numerical modeling of the self-oscillation onset near a three-dimensional backward-facing step in a transonic flow," Fluid Dyn. **51**, 534–543 (2016). doi 10.1134/S001546281604013X
23. A. P. Duben and T. K. Kozubskaya, "Jet noise simulations using quasi-1D schemes on unstructured meshes," AIAA Paper No. 2017-3856 (2017). doi 10.2514/6.2017-3856
24. A. P. Duben, N. S. Zhdanova, and T. K. Kozubskaya, "Numerical investigation of the deflector effect on the aerodynamic and acoustic characteristics of turbulent cavity flow," Fluid Dyn. **52**, 561–571 (2017). doi 10.1134/S001546281704010X
25. S. Soukov, A. Gorobets, and P. Bogdanov, "Portable solution for modeling of compressible turbulent flows on whatever hybrid systems," Mat. Model. **29** (8), 3–16 (2017).