freeCappuccino - An Open Source Software Library for Computational Continuum Mechanics

Nikola Mirkov[1(✉)], Nenad Vidanović[2], and Gordana Kastratović[2]

[1] Institute of Nuclear Sciences – Vinča, Laboratory for Thermal Engineering and Energy, University of Belgrade, 11351 Belgrade, Serbia
nmirkov@vin.bg.ac.rs
[2] Faculty of Traffic Engineering, University of Belgrade, 11000 Belgrade, Serbia

Abstract. The paper describes development of an open-source library (www.github.com/nikola-m/freeCappuccino) for computational fluid dynamics and in general computational continuum mechanics. The code is based on finite volume method on arbitrary unstructured polyhedral meshes. The interfaces to highly abstract data types such as arbitrary order tensor fields on discretized finite volume domains, and scalar and vector sparse linear systems resulting from finite volume discretization of partial differential equations are provided. Explicit manipulation of tensor fields through high level, highly abstract programming syntax is explained. Also, implicit operation over tensor fields pertinent to discretization of partial differential operators is provided and explained. The library is developed in modern version of Fortran. Code parallelization is achieved through domain decomposition and implemented using MPI and OpenMP. While avoiding the usual class syntax of object-oriented programming, the code has essentially object oriented design. Comparison is made with the well-known OpenFOAM library. The purpose of the ongoing development is providing researchers with a tool for easy transfer of mathematical operations of their physical models into functional and efficient simulation software based on finite volume method. The guiding principle of development is exchange of ideas and reproducibility in computational science in general.

Keywords: Engineering software · Computational fluid dynamics
Finite volume method · Parallel computing · High-performance computing

1 Introduction

We are in the age when engineering and science rely to a large extent on computational tools. Many of these tools are developed within a single commercially oriented company and source code stays out of the reach of engineers and scientists. In efforts to simulate more complex physics researchers are often able to add additional functionality trough appropriate interface, while in most such cases, the core simulation code is inaccessible to the user. Influential voices are being heard recently [1] in scientific community, where the important question of reproducibility in science is related, when it comes to computational science, to accessibility of the source code of the program

N. Mitrovic et al. (Eds.): CNNTech 2018, LNNS 54, pp. 137–147, 2019.
https://doi.org/10.1007/978-3-319-99620-2_11

which was used to generate the simulation data. It is stressed there that the open-source software libraries constitute the backbone of reproducible scientific computing, and that not releasing original source code raises needless roadblocks to reproducibility.

Our effort in creating a platform for collaborative development of computational algorithms and testing of mathematical models results in freeCappuccino library. The name is derived to assert two sources that influenced its development, and also to suggest that we are dealing with CAFFA (Computationally Aided Fluid Flow Analysis) [2] with FOAM (Field Operation and Manipulation) [3, 4].

While the mature commercial software tools enable efficient treatment of multi-disciplinary design optimisation (e.g. [5]) due to its monolithic character, open-source tools tend to use distributed effort on modular components written by field experts on a common platform. The challenges faced when developing open-source software tools are widely discussed [6].

Solution of complex problems of continuum mechanics (fluid flow, heat transfer, elasticity, aeroacoustics) and electromagnetics relies on approximate solution of governing Partial Differential Equations. A successful library in this field should be based on a flexible numerical approach, to be able to handle complex geometric domains found in practical situations, to have data structures capable of representing basic objects of study in the studied discipline (in our case tensor fields), and to be able to scale efficiently on HPC hardware. Also it is important to easily interface other tools in the chain, whether pre-, post-processing software or other computational libraries.

In the case of freeCappuccino we have decided to base our approach on Finite Volume Method for approximation of governing equations and to use essentially object oriented approach. Other design decisions followed from these basic ones. We have also decided to write code in modern Fortran, since it enabled implementation of all the ideas that emerged along the way and has good performance record.

2 Handling Complex Geometry and Computational Meshes

The requirement of ability to treat complex geometries which abound in engineering applications led to various numerical approaches - immersed material boundaries, castellated, and body fitted meshes. Body fitted meshes evolved to the concept of generally unstructured polyhedral meshes as the culmination of its development. In this approach, continuous domains are subdivided into discrete volume elements of polyhedral shape with arbitrary number of faces. In such case it is useful to have so-called ‘face-based’ data structure (not the only option for body-fitted meshes cf. edge-based data structure in mean-median, and Voronoi dual meshes). The face-based data structure is useful also in other approaches where, for example, we have hexahedral cells and adaptive mesh refinement based on oct-tree cell division, or in cases where block-wise structured meshes are interfaced in non-conformal way (cell faces at block interface don’t match).

While it is relatively straightforward to compute geometric properties when computational cells are well defined objects like tetrahedra, prisms, pyramids and hexahedra, it is not a trivial task in the case of face-based data structure, where optional cases include general polyhedral meshes or meshes obtained through adaptive mesh

refinement. In the present code the basic geometric data required by the Finite Volume Method, i.e. cell volumes, coordinates of cell centres and cell face centres, components of cell face normals scaled by the magnitude of the face area projection in the same coordinate direction, are all computed based on face data, with no reference to specific cell shape. For cell volume and cell centroid computation, the approach presented in [7], where, only face properties such as areas, unit normals, and face centres are employed, is adopted in present library.

Specifically important for Finite Volume Method is the way interpolation factors λ_i, i = 1, numFaces, are calculated. In the present case they represent the distance from the point where line joining cell centres intersects plane of the cell face (point j′), divided by the distance of cell centres, Fig. 1. The point of intersection is found first using the analytic expressions for the intersection of a line, and a plane, rewritten in terms of points defining the plane and the straight line. This point is not necessarily within the cell face, based on the level of cell nonorthogonality.

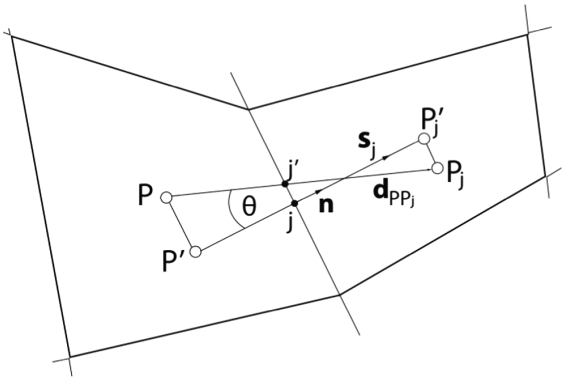

Fig. 1. A nonorthogonal grid arrangement, here represented in 2D, where the basic geometric parameters are displayed

The polyMesh format used by OpenFOAM is mesh format of choice for the library, because of its intelligent design approach which eliminates the need for cell connectivity data. The only point in simulation where cell connectivity is important is for transferring present face-based geometry data to cell-based one while writing Paraview native unstructured mesh (.vtu) files which is cell based. For that purpose cell connectivity data is generated by built in cellConectivity software utility. Other choices for input file format include Gmsh, which can be read by built-in Gmsh reader. The difference is that this format doesn't contain cell-face owner-neighbour pair data, crucial for finite volume method, and is generated before the simulation run.

It is clear that face-based approach in definition of mesh geometry gives great flexibility and requires no assumption on the type of the mesh elements, therefore such an approach is highly favourable if one insists in general application.

3 Tensor Fields Manipulation

Most important objects in the library are scalar, vector and tensor discrete fields, i.e. fields with prescribed values on discrete polyhedral cell mesh elements. There are two types of tensor fields considered, volume fields - those pertinent to cell centers, which usually denote cell averaged values in Finite Volume Method, and surface fields - where given values are defined at face centroid, or are otherwise enumerated by cell face index. In Fortran syntax they are defined using derived data types with allocatable elements. Here is the definition of three basic volume tensor field types, the third one is the symmetric tensor field type often in use in continuum mechanics (e.g. strain-rate tensor).

```
type volScalarField
  character(len=30) :: field_name
  real(dp), dimension(:), allocatable :: mag
end type

type volVectorField
  character(len=30) :: field_name
  real(dp), dimension(:), allocatable :: x, y, z
end type

type volSymmetricTensorField
  character(len=30) :: field_name
  real(dp), dimension(:), allocatable :: xx, xy, xz
  real(dp), dimension(:), allocatable ::     yy, yz
  real(dp), dimension(:), allocatable ::         zz
end type
```

A number of the basic arithmetical operators are overloaded so one can add, subtract and multiply given fields. Besides overloading, operators are polymorphic in character, so multiplication sign '*' can mean scalar field - vector field multiplication, or scalar - rank two tensor multiplication, depending on type of the fields on two sides of the '*' sign. Also a number of basic vector and tensor field algebra operations are defined. These are defined as binary and unary operators, see Table 1. Polymorphism is used with these operators to seamlessly treat distinct cases when different ranks of tensors are given, or different tensor type (volume, surface, symmetric, etc.) are provided.

As an example we present what a call to unary operator for computation of a deviatoric part of a rank 2 tensor field -.dev. incorporates.

Table 1. Basic tensor field operators in freeCappuccino library.

Operator	Type	Operates on
.dot.	Binary	Vol/surface rank 1/rank 2 tensor fields
.cross.	Binary	Rank 1 tensor fields
.tensor.	Binary	Vol/surface rank 1/rank 2 tensor fields
.colon.	Binary	Vol/surface rank 2 tensor fields
.transposed.	Unary	Rank 2 tensor fields
.trace.	Unary	Rank 2 tensor fields
.det.	Unary	Rank 2 tensor fields
.diagonal.	Unary	Rank 2 tensor fields
.hodge.	Unary	Rank 2 tensor fields
.curl.	Unary	Rank 2 tensor fields
.symm.	Unary	Rank 2 tensor fields
.skew.	Unary	Rank 2 tensor fields
.magSq.	Unary	Rank 2 tensor fields
.mag.	Unary	Rank 2 tensor fields
.dev.	Unary	Rank 2 tensor fields
.devTwo.	Unary	Rank 2 tensor fields
.hyd.	Unary	Rank 2 tensor fields

```
function deviatoric_part_rank2_tensor(T)  result(devT)
  implicit none
  type(volTensorField), intent(in) :: T
  type(volTensorField)             :: devT
  type(volTensorField)             :: I

  devT = new_volTensorField(numCells)
  I  = eye(numCells)
! overloaded operator - here '-' substracts two tensor fields
!          | '*' multiplies tensor field by a constant scalar
!          |             | '*' multiplies tensor fields by a scalar array
!          |             |          |
  devT = T - ( 1./3.0_dp * ( .trace.T * I) )

end function deviatoric_part_rank2_tensor
```

In the calling routine there are other examples of calls to unary and binary operators and polymorphism.

The main appeal for operator overloading and of shorthand operator definitions is to make it easy for users to remember how to call tensor algebra operations and to make the written code easier to read and interpret [8]. In this way the overloaded and shorthand operators have their conventional meaning and expressions of manipulations on tensor fields get their compact, comprehensive form.

4 Finite Volume Discretisation

The finite volume method for approximation of PDEs has established itself as a method of choice for engineering applications of computational fluid mechanics due to its flexibility, and most of the general purpose CFD codes employ this method for equation approximation.

To systematize the approximation of the governing equations of continuum mechanics, we cast them into integral form of the general conservation law, which is one of the central general objects of approximation,

$$\frac{d}{dt}\int_V \rho\phi dV + \int_S \rho\phi\mathbf{u}\cdot\mathbf{n}dS = \int_S \Gamma_\phi \nabla_\phi \cdot \mathbf{n}dS + \int_V q_\phi^V dV + \int_S \mathbf{q}_\phi^S \cdot \mathbf{n}dS \quad (1)$$

where $\boldsymbol{\varphi}$ is either a scalar, or a vector or tensor component to which conservation law is applied. The two terms on the left are the transient and convection term, on the right are diffusion term, and volume and surface source terms. Variables figuring in this equation are: fluid density $\boldsymbol{\rho}$, $\mathbf{t}$ is time, u fluid velocity vector, $\boldsymbol{\Gamma}$ is diffusion coefficient, and finally volume and surface source terms q^v and $\mathbf{q}^{\mathbf{S}}$, where surface term is a vector quantity.

The development of the numerical algorithm for freeCappuccino library has been described in detail in [9], where most of the algorithmic improvements are covered and code verification and validation details are presented. Code was specifically targeted for application on highly complex nonorthogonal meshes, therefore proving the second order accuracy, even in highly distorted mesh situations, was specially considered in the paper. The verification tests ranged from manufactured solutions to analytical solutions.

5 Linear Algebra

The process of discretisation of governing PDEs leads to linear systems of equations with sparse matrices. To represent such a mathematical object, an appropriate storage format should be chosen, leading to efficient storage (memory) requirements, in correspondence to numerical method and an easy interface with external libraries. The Compressed Sparse Row (CSR) format [10] as probably most popular among general sparse matrix storage formats is chosen for freeCappuccino library. Such choice is based, in the first place, on considerations of underlying finite volume algorithm. Additional advantage is based on the fact that CSR format is used by many numerical linear algebra libraries.

The finite volume discretisation involves calculating fluxes between neighbouring cells trough a sharing cell face. Pairs of indices in both permutations define nonzero matrix entries and matrix sparsity pattern. This format enables fast identifications of neighbouring cells for each cell from matrix sparsity pattern, namely, it is the list of indices of all non-zero matrix entries in a specific row, defined by cell number, except, of course, the diagonal cell, since it represents the specific cell in question. This often comes in handy when making local calculations pertinent only to cells' immediate surrounding.

In freeCappuccino, the csrMatrix object is defined from mesh data, to store sparsity pattern information and discretisation coefficients, using Fortran derived types:

```
type csrMatrix
  integer, dimension(:), allocatable :: ioffset
  integer, dimension(:), allocatable :: ja
  real(dp), dimension(:), allocatable :: coef
end type
```

By extending the csrMatrix type, we derive fvEquation derived data type. Finite volume equation is a datatype that besides sparse matrix, holds RHS vector, defined below as source vector and past values from two consecutive time-steps, o, and oo. This is done so the implicit Backward Differentiation of second order (BDF2 algorithm) can seamlessly shift time values during the simulation, and this information is pertinent to each field that is implicitly solved.

```
type, extends(csrMatrix) :: fvEquation
  real(dp), dimension(:), allocatable :: source
  real(dp), dimension(:), allocatable :: o
  real(dp), dimension(:), allocatable :: oo
end type fvEquation
```

Further extension by inheritance is done defining fvVectorEquation, where same data as above is hold for each vector component.

The fvEquation and fvVectorEquation have overloaded three basic arithmetic operations. The plus '+' sign is overloaded so one can add fvEquation and volScalarField, and do artihmetics with implicitly discretised tensor fields and volume field that define source terms in approximated governing equations. Overloading summation also applies when two objects of type e.g. fvEquation need to be added together. The same applies for minus sign, which is overloaded in the same way. What is interesting is the == sign, where basic idea is based on the one in Open-FOAM, i.e. this sign defines the same as the plus sign but is important for emulating the mathematical form of equations. The == sign is used to add volume source to linear equation systems' source if we have fvEquation and volScalarField objects respectively, on two sides of the sign, and it means adding two objects of type fvEquation if these are on different sides of the sign. That is principle of polymorphism in action. The solve function call accepts fvEquation object, solves linear system by an iterative algorithm and returns unknown volScalarField.

In freeCappuccino there are several built-in iterative solvers for sparse linear systems. The built-in solvers belong to two categories, stationary iterative methods such as Jacobi, and Gauss-Seidel, and preconditioned methods from Krylov family of algorithms - Diagonally Preconditioned Conjugate Gradient, Incomplete Cholesky Conjugate Gradient, ILU(0) preconditioned Bi-CGStab, and ILU(0) preconditioned GMRES [10]. All of these have also their parallel versions. Finally for wider choice of algorithms including Smooth Agglomeration-Algebraic Multigrid, freeCappuccino interfaces external library for sparse numerical linear algebra LIS [11].

6 Parallelisation

Modern problems in computational science and engineering often require resolution of many spatial and temporal scales which require highly advanced parallel application specific algorithms and high-performance computational (HPC) resources. State of the art HPC architectures available to researchers over the world offer combined power of multicore CPUs and GPUs with inherent parallelization approaches. Many core CPU architectures are arranged in multi node clusters offering greater computational power to the user. Fast growth in computational power available necessitates development of novel simulation approaches and tools which could use advantages of these resources in their full potential and still are able to maintain qualities such as portability and easy maintenance. State of the art codes should demonstrate capability of scaling with increasing the number of distributed computational nodes and computational cores. The problem they face and performance bottlenecks are result of the algorithmic and implementation choices. One typical example is that of numerical simulation of turbulent single/multiphase fluid flows. Common technical applications involve such flows in complex geometries where vortices at different scales are present. These vortices need high spatial numerical resolution in certain regions. Additionally to high spatial resolution, complex turbulent flows present multiscale features in time domain which lead to high demand in computational resources for long time integration over small discrete time-steps. Additional effects such as heat transfer, particulate transport and chemical reactions, or need for interphase interfaces capturing or resolving lead to stiff problems which further burden the simulation algorithm. Such problems require novel computational approaches which can follow the raise of physical complexity.

The approach adopted in freeCappuccino is based on domain decomposition and Single Program Multiple Data (SPMD) parallelisation paradigm implemented using Message Passing Interface (MPI). The required exchange of the data at certain points is achieved by a high level call to exchange routine in which data is moved to/from buffers and a call to MPI MPI_SENDRECV_REPLACE function as the main routine driving exchange of data is done. Other MPI function calls include MPI_ALLREDUCE global communication often employed in linear solvers to perform e.g. global residual norm calculation. At some places threading is used trough calls to OpenMP library.

7 Example Simulation Results

Here we illustrate use of freeCappuccino in most important context, as a fully capable computational fluid dynamics solver.

The first example is well known lid driven cavity case. The side of the cavity is $d = 0.1$, lid velocity is $U_{lid} = 1$ m/s, density is set to unit value and dynamic viscosity to $\mu = 0.01$ Pas, which give Reynolds number $Re = 10$. The mesh is uniform and quite coarse, consisting of only 400 cells. The case setup details and computational mesh is provided as a basic tutorial example in OpenFOAM library, therefore this example also serves to test the ability of freeCappuccino to use meshes written in polyMesh format. Simulation is steady state, run in laminar mode, and SIMPLE algorithm is used to solve

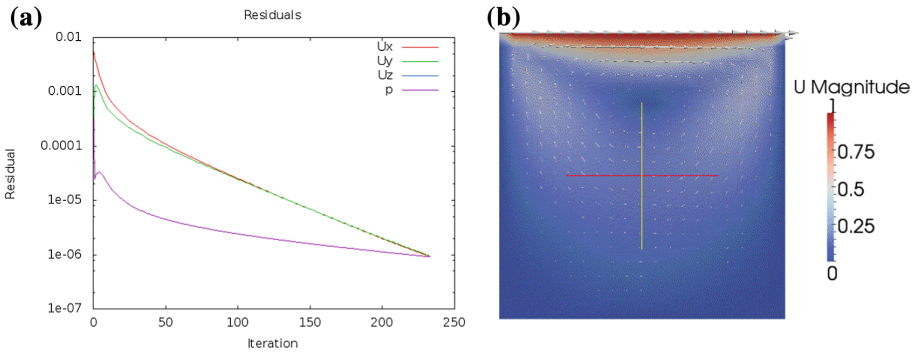

Fig. 2. Lid-driven cavity case. (a) Convergence of the SIMPLE algorithm, and (b) Velocity magnitude and flow velocity vectors at steady state.

coupling of velocity and pressure fields. Figure 2 shows simulation convergence and simulation result for velocity magnitude.

Next example is transient turbulent flow over backward facing step, also taken from OpenFOAM example set, known there as pitzDaily case. It can be used to validate several solver capabilities and as such is used as an example case for a few different OpenFOAM solvers. Here we are interested to show results of using SIMPLE in non-stationary mode. The mesh consists of 12225 cells. Turbulence is modelled using standard k-ε turbulence model. Convection scheme is second order upwind and Venkatakrishnan gradient limiter is employed. Time-step is set to $\Delta t = 0.001$ s, and number of SIMPLE iterations per time-step is limited to 30.

The flow develops to steady state after showing initial transients, which are shown in Fig. 3 and Fig. 4 for time t = 0.02 s. Other options also exist for non-stationary run besides running SIMPLE in transient mode, but this ability is useful because of the stability when larger time-steps are used. Time-stepping is based on backward differentiation formula of second order.

The code is extensively tested for turbulent flows in the atmospheric boundary layer of complex terrains. Most of the algorithmic developments were done to enable efficient treatment of highly distorted nonorthogonal meshes, which are often present

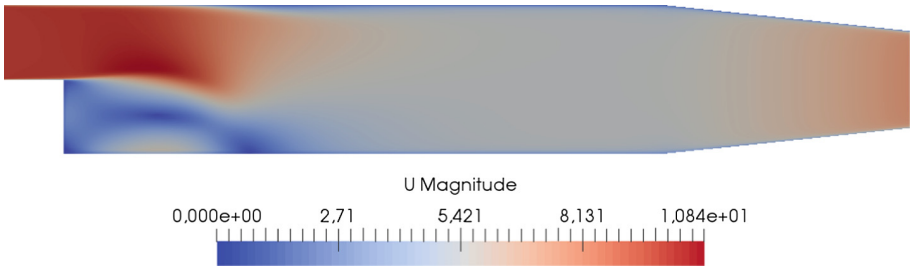

Fig. 3. Snapshot of initial transient flow development in backward facing step case of Pitz and Daily - Velocity magnitude

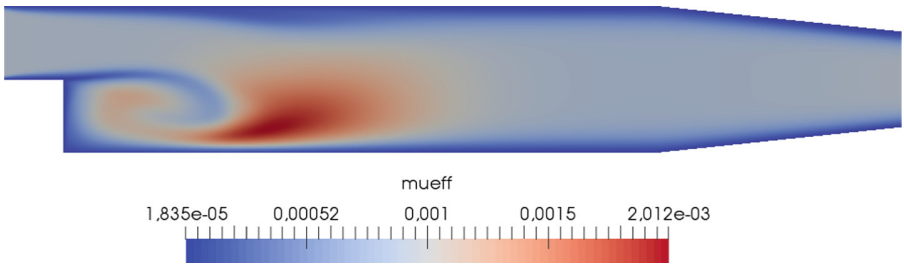

Fig. 4. Snapshot of initial transient flow development in backward facing step case of Pitz and Daily - Effective viscosity, after 0.02 s.

when real terrains are modelled. Major results are reported elsewhere e.g. [9, 12], and readers are encouraged to consult these references for more detail.

Here we show a comparison of vertical wind-velocity profiles over complex terrain, in wind-farm sitting study, Fig. 5. The differences between results of two codes with the same model are greater than between two different models in the same code. We believe that numerous algorithmic improvements aimed at flow over complex terrain cases had led to improved predictions in this case.

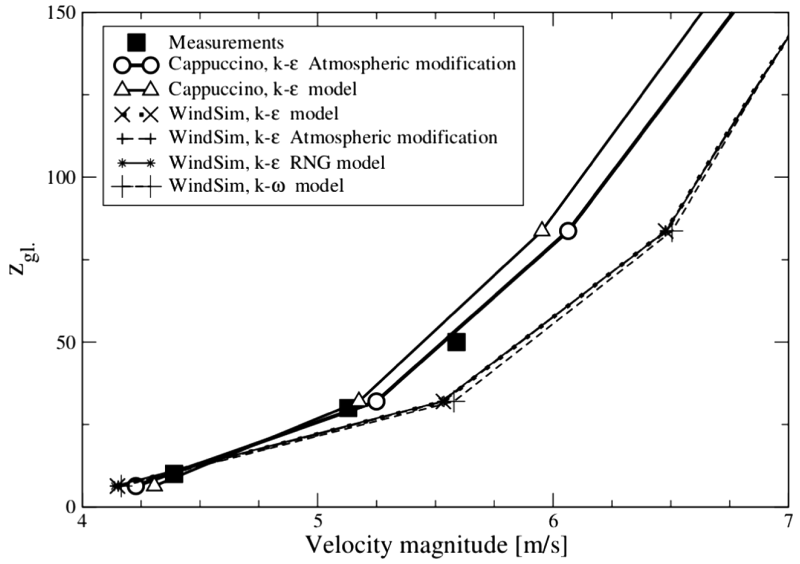

Fig. 5. Vertical wind velocity profiles predictions, see [12].

8 Conclusion

In this paper we have described a design approach for a computational science and engineering software library aimed at computational fluid dynamics and in general to computational continuum mechanics. Design decisions were such that any new solver should easily be implemented with provided tools for general unstructured mesh manipulation, operation over tensor fields and approximation of differential operators needed to write governing equations of the problem in continuum mechanics. Code parallelisation enables higher numerical efficiency for demanding problems. The code is provided in public repository with Git version control (www.github.com/nikola-m/freeCappuccino) with the hope to promote scientific collaboration and to serve as a basis for reproducibility of scientific results.

Acknowledgements. Support of the Ministry of Education, Science and Technological Development of republic of Serbia, trough project TR-33036 is greatly acknowledged.

References

1. Ince, D.C., Hatton, L., Graham-Cumming, J.: The case for open computer programs. Nature **482**, 485–488 (2012)
2. Ferziger, J.H., Perić, M.: Computational Methods for Fluid Dynamics, 2nd edn. Springer, Berlin (1999)
3. Weller, H.G., Tabor, G., Jasak, H., Fureby, C.: A tensorial approach to computational continuum mechanics using object oriented techniques. Comput. Phys. **12**(6), 620–631 (1998)
4. Jasak, H., Jemcov, A., Tuković, Ž.: OpenFOAM: a C++ library for complex physics simulations. In: International Workshop on Coupled Methods in Numerical Dynamics IUC, Dubrovnik, Croatia, 19th–21th September 2007
5. Vidanović, N., Rašuo, B., Kastratović, G., Maksimović, S., Ćurčić, D., Samardžić, M.: Aerodynamic-structural missile fin optimization. Aerosp. Sci. Technol. **65**, 26–45 (2017)
6. Bangerth, W., Heister, T.: What makes computational open source software libraries successful? Comput. Sci. Discov. **6**(1), 015010 (2013). http://stacks.iop.org/1749-4699/6/i=1/a=015010
7. Wang, Z.J.: Improved formulation for geometric properties of arbitrary polyhedra. AIAA J. **37**(10), 1326–1327 (1999)
8. Bartlett, R.E.: A simple convention for the specification of linear algebra function prototypes in C++. ACM Trans. Math. Softw. 1–6 (2015). https://trilinos.org/docs/dev/packages/thyra/doc/html/LinearAlgebraFunctionConvention.pdf
9. Mirkov, N., Rašuo, B., Kenjereš, S.: On the improved finite volume procedure for simulation of turbulent flows over real complex terrains. J. Comput. Phys. **287**, 18–45 (2015)
10. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia (2003)
11. Library of Iterative Solvers for linear systems (LIS). http://www.ssisc.org/lisl
12. Mirkov, N., Stevanović, Ž.: New non-orthogonality treatment for atmospheric boundary layer flow simulation above highly non-uniform terrains. Therm. Sci. **20**(Suppl. 1), 223–233 (2016)