SoftwareX 12 (2020) 100579

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Lethe: An open-source parallel high-order adaptative CFD solver for incompressible flows

Bruno Blais [a,*], Lucka Barbeau [a], Valérie Bibeau [a], Simon Gauvin [a], Toni El Geitani [a], Shahab Golshan [a], Rajeshwari Kamble [a], Ghazaleh Mirakhori [a,b], Jamal Chaouki [b]

[a] Research Unit for Industrial Flows Processes (URPEI), Department of Chemical Engineering, École Polytechnique de Montréal, PO Box 6079, Stn Centre-Ville, Montréal, QC, Canada H3C 3A7
[b] Process Engineering Advanced Research Lab (PEARL), Department of Chemical Engineering, École Polytechnique de Montréal, PO Box 6079, Stn Centre-Ville, Montréal, QC, Canada H3C 3A7

ARTICLE INFO

Article history:
Received 7 May 2020
Received in revised form 15 July 2020
Accepted 16 July 2020

Keywords:
Computational Fluid Dynamics
Finite Element Method (FEM)
Continuous Galerkin
High-order methods

ABSTRACT

High-order Computational Fluid Dynamics (CFD) methods have the potential to deliver higher accuracy at lower computational cost than conventional second-order methods. In this work, we introduce Lethe, an object-oriented, open-source, high-order (space and time) CFD software which leverages the well-established deal.II library. Lethe solves incompressible flow problems on 2D and 3D unstructured quad and hex meshes and allows dynamic mesh adaptation. It is parallel and it is adapted to the solution of large problems ($>10^8$ degrees of freedom) on distributed computer architectures. We illustrate some of its fundamental capacities through two benchmarks: a manufactured solution and the DNS of Taylor–Green vortices at $Re = 1600$.

Code metadata

Current code version	v0.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2020_200
Code Ocean compute capsule	Unavailable
Legal Code License	LGPL v3.0
Code versioning system used	git
Software code languages, tools, and services used	C++, MPI.
Compilation requirements, operating environments & dependencies	deal.II library v9.1.1 with Trilinos and P4Est enabled
If available Link to developer documentation/manual	https://github.com/lethe-cfd/lethe/wiki
Support email for questions	bruno.blais@polymtl.ca

Software metadata

Current software version	v0.1
Permanent link to executables of this version	https://github.com/lethe-cfd/lethe/releases/tag/v0.1
Legal Software License	LGPL
Computing platforms/Operating Systems	Linux, OS X
Installation requirements & dependencies	deal.II library with Trilinos and P4est enabled
If available, link to user manual - if formally published include a reference to the publication in the reference list	https://github.com/lethe-cfd/lethe/wiki
Support email for questions	bruno.blais@polymtl.ca

1. Motivation and significance

Computational Fluid Dynamics (CFD) has evolved considerably in the last three decades [1,2]. It is now extensively used in a large

* Corresponding author.
E-mail address: bruno.blais@polymtl.com (B. Blais).

https://doi.org/10.1016/j.softx.2020.100579

array of engineering applications ranging from more traditional mechanical and aerodynamics engineering (e.g. predicting the drag and lift forces acting on the wing of an aircraft [2]) to chemical engineering (e.g. predicting the power consumption of an agitated vessel [3]). The majority of commercial and open-source CFD software, especially those dedicated to the solution of incompressible flow problems, use second-order accurate numerical methods based on the Finite Volume Method (FVM) or the Finite Element Method (FEM) [2]. Although these methods are robust, there is no evidence that second-order accurate methods are optimal for engineering applications [2]. Alternative approaches based on high-order methods exist and they have the capacity to yield significantly higher accuracy in the same computational time.

High-order methods for CFD are defined as methods for which the order of convergence n (in the $\mathcal{L}^2$ norm) is greater than two [2]. In the case of finite element approaches (continuous or discontinuous Galerkin), this implies interpolation order p larger than one ($n = p+1$). They have demonstrated their high intrinsic potential to accurately simulate external flows [4,5], vortices [6], fluid flow in turbomachinery [7], etc. With their low numerical dissipation, high-order models are notably appropriate for Large-Eddy Simulation (LES) in which the large turbulent length and time scales of the flow are resolved while the smaller scales are modeled. Despite their high potential, their use has been thus far mostly confined to the aerospace industry. This is due to a lack of available commercial or accessible open source software that leverages the capacity of high-order schemes for the incompressible flows commonly encountered in other engineering applications (in our case chemical engineering).

In this work, we introduce a new open-source high-order CFD software for incompressible flow problems: Lethe. It is built upon the well-established deal.II library, which is an open source library for finite element modeling using both continuous or discontinuous Galerkin methods [8–10]. Through deal.II, Lethe uses state-of-the-art linear algebra libraries (Trilinos), allowing it to leverage high performance computing. By using the p4est library, it also allows for dynamic mesh adaptation with mesh partitioning in a distributed parallel (MPI) environment.

The core goal of this work is to introduce Lethe as an accessible open source high-performance platform for incompressible flow simulations. First, we briefly recall the problems generally associated with the solutions of the incompressible Navier–Stokes equations using continuous Galerkin formulations and discuss how these problems are tackled within Lethe. Then the capabilities of Lethe are outlined. We define the software architecture and describe the strategies used for time integration, to solve the non-linear and linear systems of equations and to handle parameter inputs.

1.1. Governing equations and solution strategies

Lethe solves the incompressible Navier–Stokes equations:

$$\nabla \cdot \boldsymbol{u} = 0 \tag{1}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\,\boldsymbol{u} = \frac{1}{\rho}\nabla p + \nabla \cdot \boldsymbol{\tau} + \boldsymbol{f} \tag{2}$$

with

$$\boldsymbol{\tau} = \nu\left((\nabla \boldsymbol{u}) + (\nabla \boldsymbol{u})^T\right) \tag{3}$$

with $\boldsymbol{u}$ the fluid velocity, p the pressure, ρ the density, $\boldsymbol{\tau}$ the deviatoric stress tensor, $\boldsymbol{f}$ a source term and $\nu = \frac{\mu}{\rho}$ the kinematic viscosity. Lethe can solve the system of Eqs. (1) and (2) for both steady-state and transient problems.

The solution of the incompressible Navier–Stokes equations using the finite element method has been the topic of a large number of reference books [11,12]. It is well known that, in the absence of stabilization, the choice of the velocity and pressure finite element spaces must be done to ensure that the Ladyzhenskaya–Babuska–Brezzi (LBB) inf–sup condition is met. It is also known that the Galerkin approximation of the Navier–Stokes equations may fail in convection dominated flows, for which there are boundary layers where the velocity solution and its gradient exhibit rapid variation over short length scales. In these regions, the classical Galerkin approach may lead to numerical oscillations which may greatly pollute the solution over the entire domain [13]. Stabilization of the elements, which is used in Lethe, can circumvent the limitations of the classical Galerkin approach and alleviate the need to use LBB stable elements. This notably allows for the use of equal order elements (such as $Q1-Q1$) [11].

We consider a domain Ω of contour Γ. Without loss of generality, we assume Dirichlet or zero stress conditions on the boundary such that the weak form becomes:

$$\int_\Omega \nabla \cdot \boldsymbol{u} q d\Omega = 0 \tag{4}$$

$$\int_\Omega \left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \boldsymbol{f}\right) \cdot \boldsymbol{v} d\Omega + \int_\Omega \boldsymbol{\tau} : \nabla \boldsymbol{v} d\Omega - \int_\Omega p \nabla \cdot \boldsymbol{v} d\Omega = 0 \tag{5}$$

where q and $\boldsymbol{v}$ are the basis function for pressure and velocity respectively.

Linearization of this system of equation (e.g using Newton's method), leads to a linear system of equations that must be solved for each non-linear solver iteration. The resulting block-system of equations takes the following generic form:

$$\mathcal{L}\begin{bmatrix}\boldsymbol{\delta u}_h \\ \boldsymbol{\delta p}_h\end{bmatrix} = -\begin{bmatrix}\boldsymbol{R}_u \\ \boldsymbol{R}_p\end{bmatrix} \tag{6}$$

where $\boldsymbol{R}_u, \boldsymbol{R}_p$ are the residuals associated to the momentum equation and mass conservation equation, respectively. In the case of Newton's method, the matrix $\mathcal{L}$ is the Jacobian matrix of the system which, for the incompressible Navier–Stokes equations, takes the form:

$$\mathcal{L} = \begin{bmatrix}A & B^T \\ B & 0\end{bmatrix} \tag{7}$$

where A and B are obtained by linearizing (through a Fréchet derivative) (5) and (4) with respect to $\boldsymbol{u}_h$ and $\boldsymbol{p}_h$. The system of Eqs. (6) can be solved using a direct solver. Regretfully, using a direct solver greatly limits the size of the simulations that can be carried out, especially as the bandwidth of the matrices increases (i.e. in 3D or at higher order). The use of iterative solvers is desired to enable the solution of large problems on distributed computer architecture (e.g. using MPI). Generally, (6) cannot be solved directly using iterative solvers because of the 0 block that relates the pressure to itself. To overcome this problem, Lethe uses two completely different strategies that can be used interchangeably to solve the incompressible Navier–Stokes equations : 1- Galerkin Least-Square (GLS) stabilization and 2- Schur complement Preconditioning with Grad-Div (GD) stabilization. These two approaches are independent from one another and are used to build the two families of CFD solvers within Lethe.

1.2. Galerkin Least-Square formulation

The Galerkin Least-Square (GLS) formulation is a Petrov–Galerkin formulation which achieves stabilization by adding two terms to the regular Galerkin formulation of the incompressible Navier–Stokes problem. The first term, the SUPG (streamline-upwind/Petrov–Galerkin), prevents oscillations in the velocity

field by adding an additional term which depends on the residual of the strong form of the momentum conservation [14]. The second term, the PSPG (pressure-stabilizing/Petrov–Galerkin), term allows the use of equal-order elements by adding an additional term to the continuity equation which depends on the residual of the strong form of momentum conservation [14]. Essentially, the aim is to relax the LBB condition.

The resulting weak form is:

$$\int_{\Omega} \nabla \cdot \boldsymbol{u} q d\Omega + \sum_{K} \int_{\Omega_k} \left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \nabla p - \nabla \cdot \boldsymbol{\tau} - \boldsymbol{f} \right) \cdot (\tau_u \nabla q)\, d\Omega_k = 0 \tag{8}$$

$$\int_{\Omega} \left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \boldsymbol{f} \right) \cdot \boldsymbol{v} d\Omega + \int_{\Omega} \boldsymbol{\tau} : \nabla \boldsymbol{v} d\Omega - \int_{\Omega} p \nabla \cdot \boldsymbol{v} d\Omega + \sum_{K} \int_{\Omega_k} \left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \nabla p - \nabla \cdot \boldsymbol{\tau} - \boldsymbol{f} \right) \cdot (\tau_u \boldsymbol{u} \cdot \nabla \boldsymbol{v})\, d\Omega_k = 0 \tag{9}$$

In the case of transient problems the stabilization parameters τ_u is defined as [15]:

$$\tau_u = \left[\left(\frac{1}{\Delta t} \right)^2 + \left(\frac{2\,|\boldsymbol{u}|}{h_{conv}} \right)^2 + \left(\frac{4\mu}{3 h_{diff}} \right) \right]^{-1/2} \tag{10}$$

whereas for stationary problems, τ_u is defined as [15]:

$$\tau_u = \left[\left(\frac{2\,|\boldsymbol{u}|}{h_{conv}} \right)^2 + \left(\frac{4\mu}{3 h_{diff}} \right) \right]^{-1/2} \tag{11}$$

where Δt is the time step, h_{conv} and h_{diff} are the size of the element related to the convection transport and diffusion mechanism, respectively [14,16]. In Lethe, both element size (h_{conv} and h_{diff}) are set to the diameter of a sphere of a volume equivalent to that of the cell [17,18]. Although we have not yet faced issues on this matter, we are aware that the stabilization might lead to poor conditioning in highly deformed cells.

The resulting system matrix has the following structure:

$$\mathcal{L} = \begin{bmatrix} A_{\mathrm{GLS}} & B^T_{\mathrm{GLS}} \\ B_{\mathrm{GLS}} & S_{\mathrm{GLS}} \end{bmatrix} \tag{12}$$

where the matrices A_{GLS} and B_{GLS} are the Jacobian of the weak form of the Navier–Stokes equation with the GLS stabilization term. The matrix S_{GLS} is the additional PSPG pressure–pressure matrix that arises due to the GLS stabilization. This matrix introduces an additional non-zero block on the diagonal of the Jacobian matrix of the non-linear system of equation.

The resulting system of linear equation can be solved using a monolithic iterative solver. In Lethe this is achieved by using the Trilinos library for sparse linear algebra [19,20]. Using the Trilinos wrappers of deal.II, two main linear solvers strategies are implemented:

- A GMRES solver with an incomplete LU (ILU) preconditioner. This approach will be henceforth referred to as GLS-GMRES-ILU.
- A GMRES solver with an algebraic multigrid (AMG) preconditioner with an ILU based smoother. This approach will be henceforth referred to as GLS-GMRES-AMG.

For a more detailed discussion on these iterative solvers and their respective properties, we refer the reader to the Trilinos user guide [21] and to the reference book of Elman et al. [22] or Trottenberg et al. [23]. The choice of an optimal approach depends on the dimension of the problem and on the fact that the problem may be transient or not. For transient problems, the Newton non-linear solver of Lethe allows the re-use of the Jacobian matrix of the system for multiple time-steps. Consequently, the choice of preconditioner depends on the compromise between the cost of establishing the preconditioner and the time required for the solution of the linear problems. This not only affects the choice of the linear solution strategy, but also the fill-in level of the ILU preconditioner or of the ILU smoother/coarsener of the GLS-GMRES-AMG strategy.

1.3. Grad-div LBB conforming formulation

A second approach that is deployed within Lethe to solve the incompressible Navier–Stokes equation is the use of grad-div penalization with an appropriate block linear solver. This approach builds on the work of Heister et al. [24] and can be seen as a natural parallel extension of the Step-57 of deal.II [9,25]. In the context of variational multiscale method, grad-div penalization can be related to a subgrid pressure model [26]. It leads to the following weak form:

$$\int_{\Omega} \nabla \cdot \boldsymbol{u} q d\Omega = 0 \tag{13}$$

$$\int_{\Omega} \left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \boldsymbol{f} \right) \cdot \boldsymbol{v} d\Omega + \int_{\Omega} \boldsymbol{\tau} : \nabla \boldsymbol{v} d\Omega - \int_{\Omega} p \nabla \cdot \boldsymbol{v} d\Omega + \sum_{K} \gamma \int_{\Omega_k} (\nabla \cdot \boldsymbol{u})\, (\nabla \cdot \boldsymbol{v}) = 0 \tag{14}$$

where γ is an additional parameter that can be related to the augmented lagrangian formulation. Contrary to the case of the GLS formulation, the structure of the linear system of Eq. (7) is not affected by the grad-div stabilization. To solve the saddle point of (7), we follow the same procedure as [25]. The following equations are solved in a loop until global convergence is obtained:

1. The pressure correction $\boldsymbol{\delta p}_h$ is obtained by solving:

$$\bar{S} \boldsymbol{\delta p} = -\boldsymbol{R}_p \tag{15}$$

where $\bar{S}$ is the Schur complement that is approximated using :

$$\bar{S}^{-1} \approx -(\nu + \gamma) M_p^{-1} \tag{16}$$

where M_p is the pressure mass matrix.

2. The velocity residual is corrected to take into account the pressure correction:

$$-\boldsymbol{R}^*(\boldsymbol{u}) = -\boldsymbol{R}_u - B^T \boldsymbol{\delta p}_h \tag{17}$$

3. The velocity correction $\boldsymbol{\delta u}_h$ is obtained by solving:

$$\bar{A} \boldsymbol{\delta u}_h = -\boldsymbol{R}^*(\boldsymbol{u}) \tag{18}$$

where $\bar{A}$ is the A matrix with grad-div stabilization added.

Two linear solvers are required respectively for (15) (e.g. M_p^{-1}) and (18) (e.g. $\bar{A}^{-1}$) . The solution of the (15) system is achieved using a conjugated gradient (CG) solver with an ILU or AMG preconditioner. Because of the properties of the mass matrix, the solution of this system is fast and efficient. The system of (18) is non-symmetric. It is solved with a GMRES solver with an ILU or AMG preconditioner. The global system of equation is solved using outer iterations until the global residual of the system reaches convergence. We have found that this approach was highly robust for steady-state solutions of the Navier–Stokes equations, especially since the resulting system of Eqs. (13) and (14) is a lot less non-linear than (8) and (9).

2. Software description and architecture

2.1. Overview of Lethe

Lethe is a suite of computational fluid dynamics solvers that enables the solution of incompressible flow problems in 2D and 3D systems. Lethe uses a continuous Galerkin formulation and implicit time-stepping to solve the incompressible Navier–Stokes equations (1) and (2). Lethe can be compiled under Linux, Mac and within the Linux subsystem of Windows.

Lethe supports either the use of unstructured meshes (using the GMSH [27] file format) and, through the deal.II library, also has interesting mesh generation capabilities for some geometries (ranging from a simple cube to more complex ones like airfoils). Because of its deal.II heritage, Lethe only allows for tensor elements (quadrilaterals in 2D and hexahedron in 3D). Consequently, although Lethe supports unstructured meshes, they must be made of quadrilaterals or hexahedra. This is a lesser limitation than it appears, since some mesh generation tool (such as GMSH) now support the automatic conformal transformation from tetrahedral meshes to hexahedral meshes by subdividing all tetrahedron into hexahedron. This obviously comes at the cost of additional elements, but we have not found this to be a severe limitation.

In a manner similar to some other CFD software (e.g. OpenFOAM [28]), Lethe provides an executable for each solver, as well as for each number of dimensions. Consequently, the GLS solver (Section 1.2) and the Grad-Div solver (Section 1.3) are ran from fully independent executables (e.g. the gls_navier_stokes_2d solver solves the Navier–Stokes equation using the GLS formulation in 2D domains). Lethe makes heavy use of templates, which enables generic programming at compile time. Notably, the dimension of the problem within Lethe is templated. Consequently, the same lines of codes are used to generate both the 2D and the 3D version of each solver, even though both solvers run through a different executable file.

In this section, we describe the major components of Lethe: its user input files, its implicit time integration schemes, its non-linear and physics solver architecture and finally the version control, testing and continuous integration strategy used therein.

2.2. User input and parameter files

Lethe is entirely parameterized using input files to prevent hard coding of variables and to enable users to fully control simulations. A single input file whose name is specified at run time drives Lethe. The input file contains separated sections which are combined together. Example of sections include : Simulation Control, Mesh reading/generation, Physical properties, Boundary conditions, etc. This allows maximal re-utilization of parameter sections between the solvers. For example, the mesh reading/generation section is identical for solvers that are based on a mesh and can be re-used in all solvers.

Through the use of the *muParser* library [29], the input file can parse complex expressions that depend on space and time. This allows users to specify initial conditions, source term, analytical solutions (for error calculation) and boundary conditions using expressions. This negates the use of pre-processing steps and allows dynamic mesh adaptation to maintain accuracy for space dependent source terms or boundary conditions.

Lethe supports two parameter file formats: *PRM* and *JSON* files. Listings 1 and 2 in Appendix A present examples of some of the subsections imposed through *PRM* and *JSON* files respectively. *PRM* files are inherited from deal.II. Although they are relatively modular, they suffer from a severe handicap since they do not allow the declaration of an undefined number of complex structures such as the boundary condition elements of Listing 1. Consequently, an arbitrary number of these objects must be pre-declared, which is error-prone.

To address this problematic, Lethe also allows JSON inputs which are parsed using Boost's JSON parser. This format has the same features as the *PRM* parser of deal.II, but leverages the flexibility of *JSON* files. Notably, it allows for the easy modification of the input files through python script using standard modules for *JSON*.

2.3. Physics solver architecture

The solvers for specific physics (i.e. sets of equations) are the core of Lethe. They consist of large classes that instantiate Lethe specialized classes (e.g. for simulation control, non-linear solution, etc.) and deal.II classes (e.g. for degree of freedom and mesh handling, etc.). The solver classes are responsible for enacting all of the steps required for a solution of a given problem. The solvers within Lethe use the Template Method Pattern [30]. All solvers are derived from a PhysicsSolver class, which is a pure virtual class. It provides a number of protected functions which are used by the children class, many of which are virtual and must be overriden.

The purpose of the Template Method pattern is to separate the solution algorithm into different functions that an algorithm can use to achieve a desired result. This is exactly what we do with the non-linear-solvers (which is the algorithm), and the PhysicsSolver (which implements parts of the algorithm). Consequently, any non-linear solvers can work with any physics solver. This ensures a high degree of modularity as classes that implement the physics have no knowledge on which non-linear solver is used to solve it.

When solvers share additional common ground, as is the case for the GLS and the Grad-Div Navier–Stokes solvers, an additional Subclass Sandbox design pattern is used to reduce code duplication. This allows the solvers to have full flexibility in what they implement. Fig. 1 illustrates the relationship between the PhysicsSolver class, the NavierStokesBase class and the concrete implementation of the GLS and Grad-Div solvers for the incompressible Navier–Stokes equations.

2.4. Non-linear solvers

Since Lethe uses implicit time-stepping, a non-linear problem must be solved for every time iteration. To achieve this, Lethe uses the Newton method with an adaptative correction.

It allows the non-linear solver to maintain convergence even for most highly non-linear problems. Lethe implements two variants of the Newton method with adaptive correction. The first variants (*Newton*) recalculates the Jacobian matrix for every non-linear iteration. The second variant (*skip-Newton*) recalculates the Jacobian matrix after a fixed number of non-linear solutions, which is controlled by the *skip* variable. In cases of non-convergence, it automatically forces the recalculation of the Jacobian matrix. Through Lethe flexible PhysicsSolver base class, additional non-linear solvers can be implemented without altering the Physics solver.

2.5. Time integration

Both the Grad-Div and the GLS formulation within Lethe are fully implicit. Lethe has two family of implicit time-stepping schemes: backward difference (BDF) [31] and Single Diagonal Implicit Runge–Kutta (SDIRK) [32]. For the BDF methods, orders 1 to 3 are implemented whereas only order 2 and 3 are implemented for SDIRK.

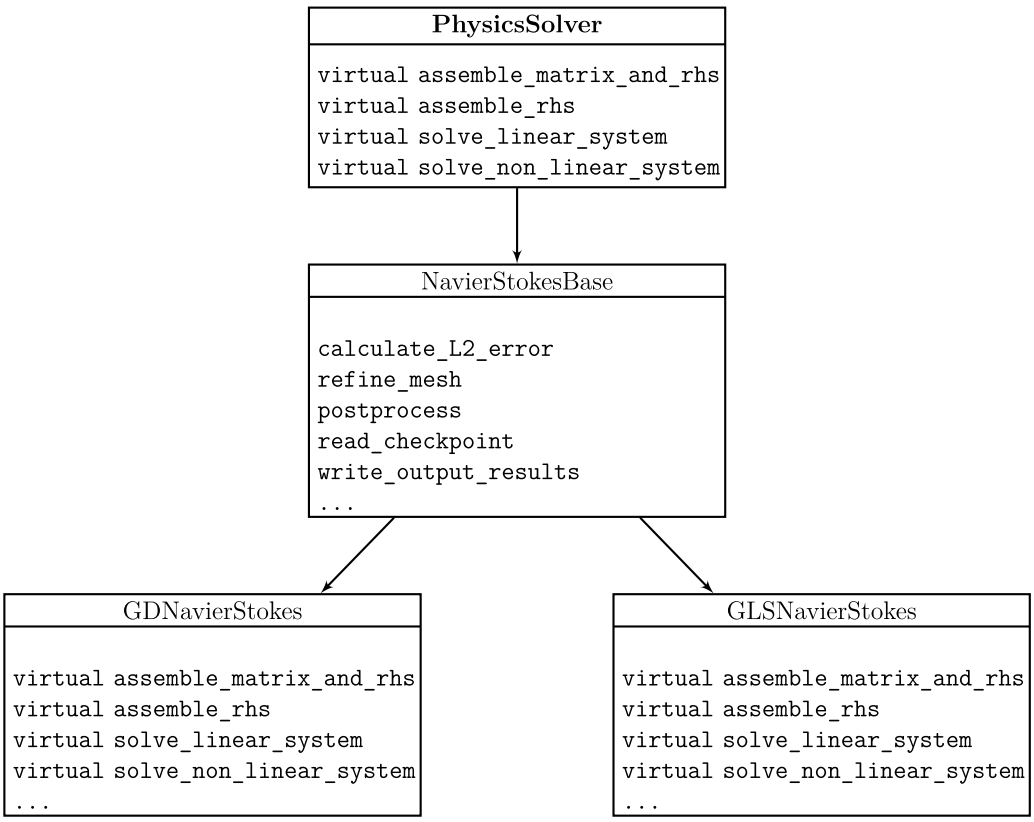


Fig. 1. Block diagram for the relation between the core PhysicsSolver class, the derived NavierStokesBase and the final GLS and Grad-Div Navier–Stokes solvers.

As is well known, the BDF methods are not self-starting. In Lethe, they are started using the SDIRK scheme of the appropriate order (BDF2 is started using a single step of SDIRK22 and BDF3 is started using two steps of SDIRK33). The SDIRK methods require multiple solutions of the non-linear problems per time-step (2 for SDIRK22 and 3 for SDIRK33). However, the same Jacobian matrix can be re-used for multiple solutions which reduces the computational cost significantly.

2.6. Version control, testing and continuous integration

Lethe is built using CMake [33] and is kept under version control in a public github repository https://github.com/lethe-cfd/lethe which is also home to the Wiki of the project. Lethe is distributed with a test suite that consists of unit tests for individual classes and application tests. The application tests are used to test the complete solvers.

Using Travis-CI and Docker, the unit tests, the application tests and the indentation are verified for each commit on a virtual machine [34]. This ensures quality control across all branches as well as stability of the master branch.

3. Illustrative examples

In this section, we consider two representative flow problems to illustrate the capacities of Lethe. The first example is a manufactured solution, which allows us to verify that our implementations are capable of reaching an arbitrary prescribed order of convergence [35]. The second problem is a full benchmark of the Direct Numerical Simulation (DNS) capacities of Lethe by simulating a Taylor–Green vortex at Re=1600. The results for the temporal energy dissipation and the volume-integrated enstrophy are used to evaluate the accuracy of Lethe and to quantify the numerical dissipation.

3.1. Manufactured solution

We consider the following 2D manufactured solution in a square domain $\Omega = [-L, L]\times[-L, L]$ for the velocity ($\boldsymbol{u} = [u, v]^T$) and the pressure p:

$$u = \sin\left(\frac{\pi}{L}x\right)^2 \cos\left(\frac{\pi}{L}y\right) \sin\left(\frac{\pi}{L}y\right) \tag{19}$$

$$v = -\cos\left(\frac{\pi}{L}x\right) \sin\left(\frac{\pi}{L}x\right) \sin\left(\frac{\pi}{L}y\right)^2 \tag{20}$$

$$p = \sin\left(\frac{\pi}{L}x\right) + \sin\left(\frac{\pi}{L}y\right) \tag{21}$$

This 2D manufactured solution is $\mathcal{C}^\infty$ for $\boldsymbol{u}$ and p and has zero no-slip boundary condition on all sides ($x = -L, x = L, y = -L, y = L$). Without loss of generality, this problem is solved for $L = 1$. To ensure that all terms have an equal contribution, the problem is solved at $Re = 1$ [35]. Fig. 2 shows the $\mathcal{L}^2$ norm of the error for the velocity obtained using the GLS formulation. It can be readily seen that the right order of convergence is recovered from Q1 to Q4 elements. Identical results, not shown here for the sake of brevity, are obtained for the GD formulation when using LBB stable elements (Q2-Q1, Q3-Q2 and Q4-Q3).

Fig. 3 shows the time required to reach a residual of 10^{-12} for the non-linear system of Eqs. (8) and (9) using the GLS-GMRES-AMG solver with an ILU(0) coarsener and smoother at various orders. This figure demonstrates that the same accuracy can be reached for laminar flow problems in 2D in much shorter time (sometimes 100x less or more). On the other hand, it also demonstrates that for the same computational time, significantly higher accuracy can be obtained (>1000x for some cases). We would like to emphasize that these conclusions are for a laminar flow problem in 2D. Although similar trends were observed in 3D and for higher Reynolds number, the impact of the higher-order might be less pronounced for other cases.

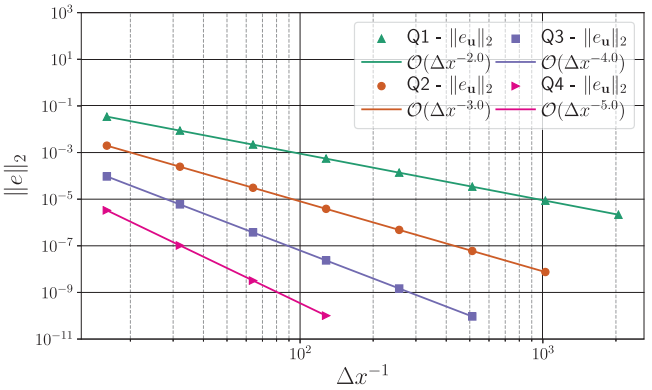


Fig. 2. Evolution of the $\mathcal{L}^2$ norm of the error on the velocity vector for different scheme order and mesh refinement.

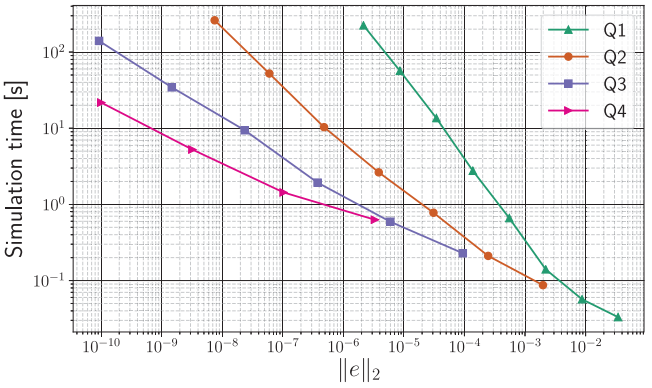


Fig. 3. Simulation time on 8 cores (Intel i9-9900k) as a function of the accuracy of the simulation for the various interpolation order considered using the GLS-GMRES-AMG solver configuration.

3.2. DNS of a Taylor Green Vortex

The Taylor–Green vortex (TGV) is a canonical problem in fluid dynamics [36]. This test case is regularly used and constitutes a canonical DNS benchmark for high order methods [2]. It evaluates various aspects of real turbulent flows such as vortex stretching and turbulence decay. Small scales turbulent structures such as eddies are generated upon breaking of the large vortices. In the absence of any external force, the energy is dissipated by the turbulent motion of the small scale eddies and eventually the fluid comes to rest. We simulate this problem in a periodic cubic box Ω of dimensions : $0 \leq (x, y, z) \leq 2\pi$ at a Reynolds number $Re = 1600$. Periodic boundary conditions are applied on the entire contour of the domain.

The initial velocity $\boldsymbol{u} = [u, v, w]^T$ and pressure p are given by:

$$u = \sin x \cos y \cos z \tag{22}$$

$$v = -\cos x \sin y \cos z \tag{23}$$

$$w = 0 \tag{24}$$

$$p = \frac{1}{16}(\cos 2x + \cos 2y)(\cos 2z + 2) \tag{25}$$

We monitor the kinetic energy E_k and its decay $\varepsilon(E_k)$, which can be calculated respectively as :

$$E_k = \frac{\rho}{\Omega}\int \frac{\boldsymbol{u}\cdot\boldsymbol{u}}{2} d\Omega \tag{26}$$

and

$$\varepsilon(E_k) = -\frac{dE_k}{dt} \tag{27}$$

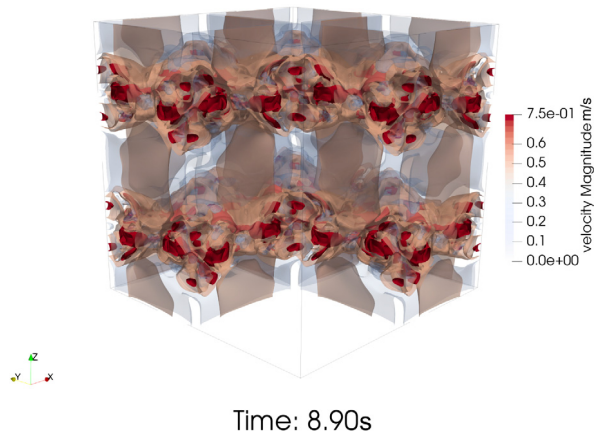


Fig. 4. Isocontours of the velocity profile around the moment of maximal dissipation predicted by the GLS scheme with Q2–Q2 elements.

The temporal decay of the kinetic energy can be compared with the enstrophy integrated throughout the whole domain (ζ). The total enstrophy is defined as :

$$\zeta = \frac{\rho}{\Omega}\int \frac{\boldsymbol{\omega}\cdot\boldsymbol{\omega}}{2} d\Omega \tag{28}$$

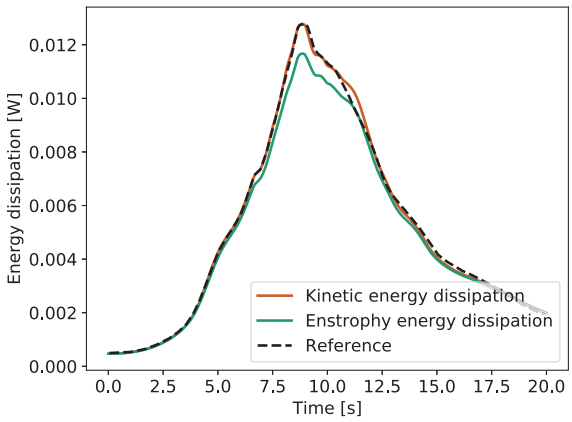
where $\boldsymbol{\omega} = \nabla \times \boldsymbol{u}$ is the vorticity. In the case of a periodic domain, the kinetic energy decay should be linked to the integrated enstrophy through the following relation

$$-\frac{dE_k}{dt} = 2\mu \int \frac{\boldsymbol{\omega}\cdot\boldsymbol{\omega}}{2} d\Omega \tag{29}$$

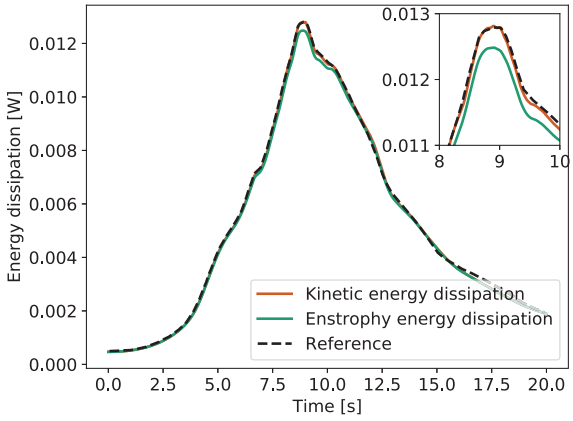
We have two measures for the kinetic energy decay : the time derivative of the total kinetic energy and the volumetric integral of the enstrophy. The volumetric integral of the enstrophy measures the kinetic energy dissipation through viscosity, whereas the time derivative of the kinetic energy measures the total energy dissipation. The difference between the two curves is the impact of numerical dissipation. Consequently, by comparing the two curves for the kinetic energy decay we can measure the degree of numerical dissipation within our simulation. These results can also be compared with DNS results obtained by de-aliased spectral methods from [2].

The problem is simulated on structured Cartesian mesh using various equal order approach (Q1–Q1 to Q3–Q3) using a third order SDIRK33 implicit time integration with a constant time step of $\Delta t = 0.005s$ for $20s$. This guarantees a sufficiently low CFL number which ensures that numerical dissipation principally emanates from the spatial discretization errors. Fig. 4 presents velocity isocontours obtained around the moment of maximal dissipation. It clearly illustrates the complexity of the flow patterns generated therein. An animation, in supplementary material, show the evolution of the velocity isocontours with time.

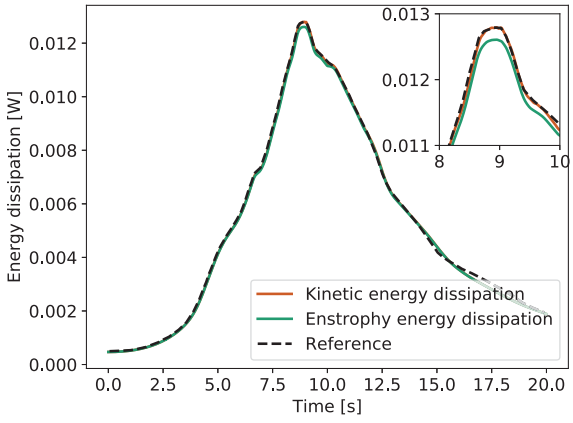
Figs. 5(a)–5(c) show the comparison between the reference simulation results and results obtained with Lethe for Q1–Q1 with 256^3 elements, Q2–Q2 with 128^3 elements and Q3–Q3 with 96^3 elements. Although the results for Q1–Q1 in Fig. 5(a) show important numerical dissipation, this is clearly not the case for Q2–Q2 (Fig. 5(b)) and, even moreso, for Q3–Q3 (Fig. 5(c)). Clearly, Lethe is able to reproduce the kinetic energy decay with low numerical dissipation. Considering the complexity of this test case [2], it clearly demonstrates Lethe's potential for implicit high-order CFD simulations of incompressible flows.



(a) 256^3 mesh simulated using **Q1-Q1** elements



(b) 128^3 mesh simulated using **Q2-Q2** elements



(c) 96^3 mesh simulated using **Q3-Q3** elements

Fig. 5. Comparison between the kinetic energy decay measured from the time derivative of the kinetic energy or by the volumetric integral of the enstrophy with a GLS formulation.

4. Impact and conclusions

Despite their growing popularity and their highly desirable properties, high-order methods for computational fluid dynamics or multiphysics problems have seen little to no adoption in the field of chemical engineering. By themselves, they could have a great impact for the intensification of chemical processes by allowing accurate simulation of the flow through packed bed reactors, agitated vessel, etc. Combined with additional models for multiphase flows, such as Discrete Element Method (DEM) [37] and CFD-DEM [38], they could enable the design, optimization and troubleshooting of a vast array of reactors such as fluidized and spouted beds.

The goal of Lethe is to enable a larger adoption of these methods by making HPC-ready solvers for incompressible CFD and multiphysics problem. Lethe is built on a strong heritage of open-source software, notably through the robust deal.II library. At its present stage, Lethe already has two comprehensive CFD solvers that enable the solution of steady-state and transient problems using high-order methods (time and space) on unstructured mesh, in parallel and with the capacity for dynamic mesh adaptation. Although Lethe is developed with chemical engineering related applications in mind, its capacity extend far beyond these applications and could be applied for any incompressible flow problem. Within this work, we have presented the array of features which Lethe already possesses and have demonstrated its capacity through two benchmark problems of increasing difficulty. The results obtained for the TGV problem, are, to the best of the author's knowledge, one of the first demonstration of the potential of high-order stabilized continuous GLS approaches for turbulent flows.

Future features, which at the time of writing are ongoing development, include the integration of a DEM module coupled with the CFD solver of Lethe, immersed boundaries and support for complex fluid rheology.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Bruno Blais would like to acknowledge the financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC) through the RGPIN-2020-04510 Discovery Grant. The authors would also like to acknowledge technical support and computing time provided by Compute Canada and Calcul Québec.

Appendix A. Examples of input files

Listing 1: Example of sections in a PRM file

```
#-------------------------------------------
# Simulation Control
#-------------------------------------------
subsection simulation control
 set method            = steady
 set output name        = couette
 set output frequency     = 1
 set number mesh adapt    = 2    # Number of mesh
    adaptation
end

#-------------------------------------------
# Physical Properties
#-------------------------------------------
subsection physical properties
 set kinematic viscosity        = 1.0000
end

#-------------------------------------------
```

```
# Analytical Solution
#---------------------------------------------
subsection analytical solution
 set enable            = true
 subsection uvw
  set Function constants = eta=0.25, ri=0.25
  set Function expression = -sin(atan2(y,x))*(-(eta*eta)
     / (1-eta*eta) *
  sqrt(x*x+y*y) + ri*ri/(1-eta*eta)/sqrt(x*x+y*y));
  cos(atan2(y,x))*(-(eta*eta) / (1-eta*eta)* sqrt(x*x+y
    *y)+
  ri*ri/(1-eta*eta)/sqrt(x*x+y*y)) ; 0
 end
end

#---------------------------------------------
# Force
#---------------------------------------------
subsection forces
 set verbosity         = verbose
 set calculate forces     = false
 set calculate torques    = true
 set force name         = force
 set torque name        = torque
 set output precision     = 10
 set calculation frequency = 1
 set output frequency     = 1
end

#---------------------------------------------
# Mesh
#---------------------------------------------
subsection mesh
 set type            = dealii
 set grid type = hyper_shell
 set grid arguments = 0, 0 : 0.25 : 1 : 4:  true
 set initial refinement  = 2
end

# ---------------------------------------------
# Boundary Conditions
#---------------------------------------------
subsection boundary conditions
 set number             = 2
  subsection bc 1
   set type          = noslip
  end
  subsection bc 0
   set type          = function
   subsection u
    set Function expression = -y
   end
   subsection v
    set Function expression = x
   end
   subsection w
    set Function expression = 0
   end
  end
end
```

Listing 2: Example of sections in a JSON file

```
{
 "simulation control":
 {
  "method": "steady",
  "output name": "couette",
  "output frequency": 1,
  "subdivision": 2,
  "number mesh adapt": 2
 },
 "FEM":
 {
  "velocity order": 2,
  "pressure order": 1,
  "qmapping all": true
 },
 "physical properties":
 {
  "kinematic viscosity": 1.000
 },
 "analytical solution": {
  "enable": true,
  "uvw": {
  "Function constants": {
  "eta": 0.25,
  "ri": 0.25
  },
  "Function expression": "-sin(atan2(y,x))*(-(eta*eta) /
  (1-eta*eta)*sqrt(x*x+y*y)+ ri*ri/(1-eta*eta)/sqrt(x*x
    +y*y));
  cos(atan2(y,x))*(-(eta*eta) / (1-eta*eta)* sqrt(x*x+y
    *y)+
  ri*ri/(1-eta*eta)/sqrt(x*x+y*y)) ; 0"
  }
 },
 "forces":
 {
  "verbosity": "verbose",
  "calculate forces": false,
  "calculate torques": true,
  "force name": "force",
  "torque name": "torque",
  "output precision": 10,
  "display precision": 3,
  "calculation frequency": 1,
  "output frequency": 1
 },
 "mesh":
 {
  "type": "dealii",
  "grid type": "hyper_shell",
  "grid arguments": "0, 0 : 0.25 : 1 : 4:  true",
  "initial refinement": 2
 },
 "boundary conditions":
 [
  {
   "type": "noslip"
  },
  {
   "type": "function",
   "u":
    {
    "Function expression": "-y"
    },
   "v":
    {
    "Function expression": "x"
    },
   "w":
    {
    "Function expression": "0"
    }
  }
 ]
}
```

Appendix B. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.softx.2020.100579.

References

[1] Versteeg HK, Malalasekera W. An introduction to computational fluid dynamics: The finite volume method. Pearson Education; 2007.

[2] Wang Z, Fidkowski K, Abgrall R, Bassi F, Caraeni D, Cary A, et al. High-order CFD methods: current status and perspective. Internat J Numer Methods Fluids 2013;72(8):811–45. http://dx.doi.org/10.1002/fld.3767.

[3] Blais B, Lassaigne M, Goniva C, Fradette L, Bertrand F. A semi-implicit immersed boundary method and its application to viscous mixing. Comput Chem Eng 2016;85. http://dx.doi.org/10.1016/j.compchemeng.2015.10.019.

[4] Deng X, Mao M, Tu G, Zhang H, Zhang Y. High-order and high accurate CFD methods and their applications for complex grid problems. Commun Comput Phys 2012;11(4):1081–102. http://dx.doi.org/10.4208/cicp.100510.150511s.

[5] Ahrabi BR, Mavriplis DJ. A scalable solution strategy for high-order stabilized finite-element solvers using an implicit line preconditioner. Comput Methods Appl Mech Engrg 2018;341:956–84. http://dx.doi.org/10.1016/J.CMA.2018.07.026.

[6] Carton de Wiart C, Hillewaert K, Duponcheel M, Winckelmans G. Assessment of a discontinuous Galerkin method for the simulation of vortical flows at high Reynolds number. Internat J Numer Methods Fluids 2014;74(7):469–93. http://dx.doi.org/10.1002/fld.3859.

[7] Tucker P. Trends in turbomachinery turbulence treatments. Prog Aerosp Sci 2013;63:1–32. http://dx.doi.org/10.1016/J.PAEROSCI.2013.06.001.

[8] Arndt D, Bangerth W, Blais B, Clevenger TC, Fehling M, Grayver AV, Heister T, Heltai L, Kronbichler M, Maier M, Munch P, Pelteret J-P, Rastak R, Thomas I, Turcksin B, Wang Z, Wells D. The `deal.ii` library, version 9.2. Journal of Numerical Mathematics 2020. http://dx.doi.org/10.1515/jnma-2019-0064, https://dealii.org/deal92-preprint.pdf. in press.

[9] Arndt D, Bangerth W, Clevenger TC, Davydov D, Fehling M, Garcia-Sanchez D, et al. The deal.II library, version 9.1. J Numer Math 2019. http://dx.doi.org/10.1515/jnma-2019-0064.

[10] Alzetta G, Arndt D, Bangerth W, Boddu V, Brands B, Davydov D, et al. The deal.II library, version 9.0. J Numer Math 2018. http://dx.doi.org/10.1515/jnma-2018-0054.

[11] Donea J, Huerta A. Finite element methods for flow problems. John Wiley & Sons; 2003, p. 28–9.

[12] Reddy JN, Gartling DK. The finite element method in heat transfer and fluid dynamics. CRC press; 2010.

[13] Hachem E, Rivaux B, Kloczko T, Digonnet H, Coupez T. Stabilized finite element method for incompressible flows with high Reynolds number. J Comput Phys 2010;229(23):8643–65. http://dx.doi.org/10.1016/J.JCP.2010.07.030.

[14] Tezduyar TE. Stabilized finite element formulations for incompressible flow computations. In: Advances in applied mechanics, vol. 28, Elsevier; 1992, p. 1–44.

[15] Tezduyar T, Sathe S. Stabilization parameters in supg and pspg formulations. J Comput Appl Mech 2003;4(1):71–88.

[16] Tezduyar TE, Mittal S, Ray SE, Shih R. Incompressible flow computations with stabilized bilinear and linear equal-order-interpolation velocity-pressure elements. Comput Methods Appl Mech Engrg 1992;95(2):221–42. http://dx.doi.org/10.1016/0045-7825(92)90141-6.

[17] Blais B, Ilinca F. Development and validation of a stabilized immersed boundary CFD model for freezing and melting with natural convection. Comput & Fluids 2018. http://dx.doi.org/10.1016/j.compfluid.2018.03.037.

[18] Ilinca F, Yu KR, Blais B. The effect of viscosity on free surface flow inside an angularly oscillating rectangular tank. Comput & Fluids 2019;183:160–76. http://dx.doi.org/10.1016/j.compfluid.2019.02.021.

[19] Codina R, Principe J. Dynamic subscales in the finite element approximation of thermally coupled incompressible flows. Internat J Numer Methods Fluids 2007;54(6–8):707–30. http://dx.doi.org/10.1002/fld.1481.

[20] Heroux MA, Willenbring JM. A new overview of the trilinos project. Sci Program 2012;20(2):83–8. http://dx.doi.org/10.3233/SPR-2012-0355.

[21] Heroux MA, Willenbring JM. Trilinos users guide. United States. Department of Energy; 2003.

[22] Elman HC, Silvester DJ, Wathen AJ. Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics. Oxford University Press, USA; 2014.

[23] Trottenberg U, Oosterlee CW, Schuller A. Multigrid. Elsevier; 2000.

[24] Heister T, Rapin G. Efficient augmented Lagrangian-type preconditioning for the oseen problem using grad-div stabilization. Internat J Numer Methods Fluids 2013;71(1):118–34. http://dx.doi.org/10.1002/fld.3654.

[25] Zhao M, Heister T. deal.II Tutorial program step-57. 2020, http://www.dealii.org/developer/doxygen/deal.II/step_57.html.

[26] Olshanskii M, Lube G, Heister T, Löwe J. Grad–div stabilization and subgrid pressure models for the incompressible Navier–Stokes equations. Comput Methods Appl Mech Engrg 2009;198(49–52):3975–88. http://dx.doi.org/10.1016/J.CMA.2009.09.005.

[27] Geuzaine C, Remacle J-F. Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. Internat J Numer Methods Engrg 2009;79(11):1309–31.

[28] Jasak H, Jemcov A, Tukovic Z. OpenFOAM: A C++ library for complex physics simulations. In: International workshop on coupled methods in numerical dynamics, vol. 1000. IUC Dubrovnik Croatia; 2007, p. 1–20.

[29] Berg I. Muparser-a fast math parser library. 2011.

[30] Gamma E, Helm R, Johnson R, Vlissides J. Design patterns: Abstraction and reuse of object-oriented design. In: European conference on object-oriented programming. Springer; 1993, p. 406–31.

[31] Hay A, Etienne S, Pelletier D, Garon A. Hp-adaptive time integration based on the BDF for viscous flows. J Comput Phys 2015;291:151–76. http://dx.doi.org/10.1016/j.jcp.2015.03.022.

[32] Holst KR, Glasby RS, Bond RB. On the effect of temporal error in high-order simulations of unsteady flows. J Comput Phys 2019;108989. http://dx.doi.org/10.1016/J.JCP.2019.108989.

[33] Martin K, Hoffman B. Mastering CMake: a cross-platform build system. Kitware; 2010.

[34] Beller M, Gousios G, Zaidman A. Oops, my tests broke the build: An explorative analysis of Travis CI with GitHub. In: 2017 IEEE/ACM 14th International conference on mining software repositories (MSR). IEEE; 2017, p. 356–67.

[35] Oberkampf WL, Roy CJ. Verification and validation in scientific computing. Cambridge University Press; 2010.

[36] DeBonis JR. Solutions of the taylor-green vortex problem using high-resolution explicit finite difference methods. In: 51st AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition 2013. 2013, http://dx.doi.org/10.2514/6.2013-382.

[37] Blais B, Vidal D, Bertrand F, Patience GS, Chaouki J. Experimental methods in chemical engineering: Discrete element method—DEM. Can J Chem Eng 2019;97(7):1964–73. http://dx.doi.org/10.1002/cjce.23501.

[38] Bérard A, Patience GS, Blais B. Experimental methods in chemical engineering: Unresolved CFD-DEM. Canad J Chem Eng 2020;98(2):424–40. http://dx.doi.org/10.1002/cjce.23686.