OpenFOAM for Computational Fluid Dynamics

Goong Chen, Qingang Xiong, Philip J. Morris, Eric G. Paterson, Alexey Sergeev, and Yi-Ching Wang

Introduction

There is a revolution going on, impacting and transforming how computational mechanics and the associated design and optimization are done: the emergence, availability, and large-scale use of *OpenFOAM* [1]. It belongs to the contemporary *open-source trend* not unlike the roles played by the Linux operating system or the Internet encyclopedia Wikipedia. OpenFOAM is free and is used by thousands of people worldwide in both academic and industrial settings. The acronym OpenFOAM stands for *Open Source Field Operation and Manipulation.*

Computational mathematics and mechanics provide fundamental methods and tools for simulating physical processes. Numerical computation can offer important insights and data that are either difficult or expensive to measure or to test experimentally. What is more, numerical computation can simulate supernova explosions and galaxy formations, which cannot be produced in earthbound laboratories. It has been recognized for at least thirty years that *computational science* constitutes a third and independent branch of science on equal footing with *theoretical and experimental sciences*. Cutting across disciplines at the center of computational science is *computational fluid dynamics (CFD)*, which makes up the core of OpenFOAM and is the focus of this article.

In the early days of CFD research and development, computer programs ("codes") were primarily developed in universities and national laboratories. Many of these efforts had lifetimes of ten to twenty years and involved numerous Ph.D. students and postdoctoral associates. Fueled by intense Ph.D.-level research, those early codes provided the basis of modern CFD knowledge. However, this model of development had several flaws. The constant turnover of personnel in academic research groups created serious continuity problems, especially if the faculty advisor or group leader was not managing the code architecture. Another challenge was that Ph.D. students and postdocs in engineering and mathematics were often self-taught programmers, which meant that most of the codes were suboptimal programs. Those student-written research codes often became notorious as "spaghetti code", which was hard to extend to new physics or new parallel high-performance computing architectures without extraordinary effort. Finally, because such a significant amount of time and financial resources had been invested in the development, those codes were usually *proprietary* and rarely made available to the public except to those in the extended academic family of the leader.

Goong Chen is professor of mathematics at Texas A&M University (TAMU) and Texas A&M University at Qatar (TAMUQ). His email address is gchen@math.tamu.edu.

Qingang Xiong is postdoctoral researcher of mechanical engineering at Iowa State University. His email address is xiong@iastate.edu.

Philip J. Morris is Boeing/A.D. Welliver Professor of Aerospace Engineering at The Pennsylvania State University. His email address is pjm@psu.edu.

Eric G. Paterson is Rolls Royce Commonwealth Professor of Marine Propulsion and department head of Aerospace and Ocean Engineering at Virginia Tech. His email address is egp@vt.edu.

Alexey Sergeev is postdoctoral fellow of mathematics at TAMU and TAMUQ. His email address is asergeev@asergeev.com.

Yi-Ching Wang is a Ph.D. student in the mathematics department of TAMU. Her email address is ycwang@math.tamu.edu.

DOI: http://dx.doi.org/10.1090/noti1095

If the researcher is not a CFD code developer, then most of the time the only alternative is to buy and use commercial CFD software packages. There are now many such CFD packages (see, e.g., those listed in [2], though this list is not exhaustive). License fees for commercial software typically range from US$10,000 to US$50,000 per year depending on the "added extras", the number of users, whether multiple licenses are required for parallel computation, and the commercial or academic nature of the license. This is not inexpensive. For a faculty member who doesn't have a research grant or is retired, the cost is generally prohibitive.

As long as the Internet has existed, there has been free and open-source software available to download and share. However, over the past decade, the level of sophistication and quality of open-source software has significantly grown, largely aided by the move to *object-oriented programming* and online version-control repositories (e.g., SourceForge [3], GitHub [4]). As in the early days, much of this software finds its roots in academia and national laboratories.

OpenFOAM was born in the strong British tradition of fluid dynamics research, specifically at The Imperial College, London, which has been a center of CFD research since the 1960s. The original development of OpenFOAM was begun by Prof. David Gosman and Dr. Radd Issa, with principal developers Henry Weller and Dr. Hrvoje Jasak. It was based on the *finite volume method (FVM)* [5], an idea to use C^{++} and object-oriented programming to develop a syntactical model of *equation mimicking* (see Box 2) and scalar-vector-tensor operations. A large number of Ph.D. students and their theses have contributed to the project. Weller and Jasak founded the company Nabla Ltd., but it was not successful in marketing its product, FOAM (the predecessor of OpenFOAM), and folded in 2004. Weller founded OpenCFD Ltd. in 2004 and released the GNU general public license of OpenFOAM software. OpenFOAM constitutes a C^{++} CFD toolbox for customized numerical solvers (over sixty of them) that can perform simulations of *basic CFD*, *combustion*, *turbulence modeling*, *electromagnetics*, *heat transfer*, *multiphase flow*, *stress analysis*, and even *financial mathematics* modeled by the *Black-Scholes equation.* In August 2011, OpenCFD was acquired by Silicon Graphics International (SGI). In September 2012, SGI sold OpenCFD Ltd to the ESI Group.

While OpenFOAM may be the first and most widely adopted open-source computational mechanics software, there indeed are other examples. A few are briefly mentioned here. They include *deal.ii* [6], a finite-element Differential Equations Analysis Library, which originally emerged from work at the Numerical Methods Group at Universität Heidelberg, Germany, and today it is a global open-source project maintained primarily at Texas A&M University, Clemson University, and Universität Heidelberg and has dozens of contributors and several hundred users scattered around the world.

The Stanford University Unstructured *(SU2)* [7] suite is an open-source collection of C^{++}-based software tools for performing partial differential equation (PDE) analysis and solving PDE constrained optimization problems. The toolset is designed with computational fluid dynamics and aerodynamic shape optimization in mind, but is extensible to treat arbitrary sets of governing equations such as potential flow, electrodynamics, chemically reacting flows, and many others. SU2 is under active development in the Aerospace Design Lab (ADL) of the Department of Aeronautics and Astronautics at Stanford University and is released under an open-source license.

Méfisto [8], 3D finite element software for numerical solutions of a set of boundary value problems, has been posted by Prof. Alain Perronnet of the Laboratoire Jacques-Louis Lions at the Université Pierre et Marie Curie in Paris, France, who is a long-time collaborator with the first author of this article.

Another open-source software package for CFD or PDEs includes *MFIX* (Multiphase Flows with Interface eXchanges), developed by the National Energy Technology Laboratory (NETL) of the Department of Energy [9], suitable for hydrodynamics, heat transfer, and chemical reactions in fluid-solid systems. It is based on the finite volume method and written in Fortran.

Still more open-source *finite element* softwares such as FEniCS, FreeFem++, etc., can be found in [37]. Nevertheless, most of their primary emphases are not built for the purpose of CFD.

The revenue and survival strategy of the company OpenCFD Ltd. (which has been absorbed into ESI Group), is a "Redhat model" [10] providing support, training, and consulting services. While OpenFOAM is open-source, the development model is a "*cathedral*" style [11] where code contributions from researchers are not accepted back into the main distribution due to strict control of the code base. For researchers who want to distribute their developments and find other online documentation, there are a community-oriented discussion forum [12], a wiki [13], and an international summer workshop [14].

Now, with the open-source libraries in OpenFOAM, one does not have to spend one's whole career writing CFD codes or be forced to buy commercial softwares. Many other users of OpenFOAM have developed relevant libraries and solvers that are either posted online or may be requested for

```
// define field scalar u and f
volVectorField u, f;
// construct the Laplacian equation and solve it
solve
(
fvm::laplacian(u) == f
);
```

Box 1. OpenFOAM code for the potential equation (1).

free. The number of OpenFOAM users has been steadily increasing. It is now estimated to be of

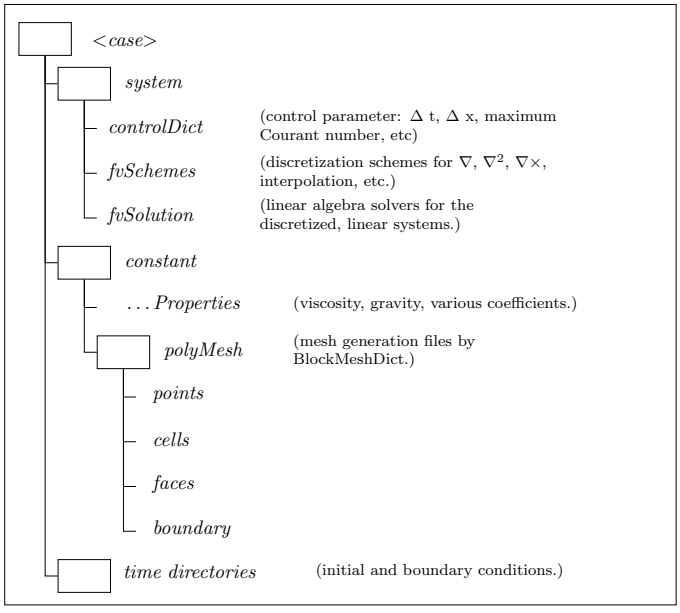


Chart 1. Case directory structure (adapted from [1]).

the order of many thousands, with the majority of them being engineers in Europe. But the U.S. is catching up.

A Sketch of How to Use OpenFOAM

For beginners who are enthusiastic about learning how to use OpenFOAM to obtain CFD solutions the best way is to study the many tutorial examples available in [1]. One such tutorial is the lid-driven cavity case [15]. (Such a case will also be computed in Examples 1 and 2.) It provides nearly all information *from start to finish* as to how to use OpenFOAM, including *preprocessing, solving* (i.e., how to run the codes), and *postprocessing*. The tutorial has a couple dozen pages. If the beginner can get some help from an experienced OpenFOAM user, then it usually takes only a few weeks to run a simple OpenFOAM computer program for this problem.

```
Solve
(
fvm::ddt(rho,U)
+ fvm::div(U,U)
- fvm::laplacian(mu,U)
==
- fvc::grad(p)
+ f
);
```

Box 2. OpenFOAM code for the N-S equation (3). Here, as well as in Box 1, *equation mimicking* is quite obvious. Note that the specifications fvm and fvc are selected by the user from the fvSchemes dictionary in the *system* dictionary; cf. Chart 1. Here fvm::laplacian means an *implicit* finite volume discretization for the Laplacian operator, and similarly for fvm::div for the divergence operator. On the other hand, fvc::grad means an *explicit* finite volume discretization for the gradient operator. The parentheses (,) means a product of the enclosed quantities, including tensor products.

As most readers may not necessarily be interested in running OpenFOAM codes for now, in this section we will mainly give a brief sketch. We first illustrate this for a simple elliptic boundary value problem

$$(1)\quad \begin{cases} \text{(i)} & \nabla^2 u(x,y,z) = f(x,y,z) \text{ on } \Omega \subseteq \mathbb{R}^3, \\ \text{(ii)} & u(x,y,z) = g(x,y,z) \text{ on the boundary } \partial\Omega. \end{cases}$$

In OpenFOAM, one can use a Laplacian solver in the heat transfer library to obtain numerical solutions. By using the C^{++} language, in OpenFOAM (1)(i) is written as in Box 1.

Note that the inhomogeneous Dirichlet boundary condition, given in (1)(ii), will be prescribed elsewhere, in the "time directories" in the Case Directory Structure, as shown in Chart 1. If instead of (1)(ii) we have inhomogeneous Neumann or Robin boundary conditions such as

$$(2)\quad \frac{\partial u(x,y,z)}{\partial n} = g(x,y,z), \qquad \frac{\partial u(x,y,z)}{\partial n} + \alpha u(x,y,z) = g(x,y,z) \text{ on } \partial\Omega,$$

they can be specified similarly in the time directories.

To numerically solve a PDE by using OpenFOAM, a user needs to create a Case Directory Structure as shown in Chart 1. Normally it contains three subdirectories. The user first gives a name for the <case>. The compositions of the various subdirectories are indicated in Chart 1.

Now we look at the core case of this article, the incompressible Navier-Stokes (N-S) equations in

CFD. The governing equations are

(3)
$$\frac{\partial(\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) - \mu \nabla^2 \mathbf{u} = -\nabla p + \mathbf{f}(x, y, z, t),$$

(4)
$$\nabla \cdot \mathbf{u} = 0.$$

Note that in (2), $\mathbf{uu}$ is defined to be the 3×3 matrix

$$\mathbf{uu} = [u_i u_j]_{3\times 3}.$$

One can specify the given initial and boundary conditions on $\mathbf{u}$ in the "time directories" of Chart 1.

An effective algorithm for solving the coupled system (3) and (4) is the PISO (pressure-implicit with splitting of operators) algorithm of Issa [16]; see also [17]. In OpenFOAM, basically (3) is written as shown in Box 2.

With some details, the PISO algorithm is implemented in OpenFOAM as shown in Box 3. This (largely) takes care of the *equation solving* step.

For *preprocessing* involving *mesh generation*, one can use the utility *blockMesh*, supplied in OpenFOAM, to first generate a rectangular mesh for a cubic domain. The input data consists of coordinates of eight vertices of the cube and numbers of cells in each direction, (n_x, n_y, n_z). The output is a rectangular mesh containing $n_x \times n_y \times n_z$ cells. In case of more complicated geometry, one can use either the *snappyHexMesh* utility or third-party packages such as Gambit meshing software [18], with subsequent conversion into OpenFOAM format.

Finally, for *postprocessing*, to produce graphical output [19] OpenFOAM uses an open-source, multiplatform data analysis and visualization application called *ParaView* [20]. Alternatively, one can also use third-party commercial products such as *EnSight* [21].

As opposed to a monolithic solver as typically seen in commercial software, *pisoFoam* is one of seventy-six *standard solvers* that are included in the OpenFOAM distribution. These solvers are tailored to specific physics in the broad categories of combustion, compressible flow, discrete methods, electromagnetics, financial, heat transfer, incompressible flow, Lagrangian particle dynamics, multiphase flow, and stress analysis. There are also eighty-plus *standard utilities* for pre- and postprocessing of data, parallel computing, and mesh creation and manipulation. For all of these different programs, the burden is on the user to verify that the implemented physics and models match their needs and intended application.

Turbulence Modeling and Examples

In addition to solvers and utilities, OpenFOAM is distributed with numerous *standard libraries*. These libraries address both numerical algorithms

```
//define field vector fluid velocity u and f, face flux phi,
      and pressure p
volVectorField u, f;
volScalarField p;
surfaceScalarField phi;
//define constant parameter fluid dynamical viscosity nu
scalarField nu;
//construct the fluid velocity equation
fvVectorMatrix UEqn (
fvm::ddt(u) + fvm::div(phi, u) - fvm::laplacian(nu, u) - f )      (a)
//solve the momentum equation using explicit pressure
solve (
UEqn == -fvc::grad(p) )
//predict the intermediate fluid velocity to calculate face flux
volVectorField rUA = 1.0/UEqn.A();
u = rUA *UEqn.H();
phi = fvc::interpolate(u) & mesh.Sf();
//construct the pressure equation using the constraint
      from continuity equation
fvScalarMatrix pEqn (
fvm::laplacian(rUA,p) == fvc::div(phi) )
pEqn.solve();
//correct the fluid velocity by the post-solve pressure
      and update face flux
u = u - rUA*fvc::grad(p);
phi = phi - pEqn.flux();                                          (b)
```

Box 3. The OpenFOAM code to solve the N-S equation of incompressible fluid. Note that the codes from lines (a) to (b) implement the PISO algorithm [16], [17].

and basic physics, the latter of which includes thermophysical properties, reaction models, radiation models, chemistry models, liquid properties, and turbulence modeling. For this article, we focus on turbulence modeling due to its universal nature in CFD.

Since exact solutions to the N-S equations are mostly unavailable, we need to rely on numerical methods to find approximate solutions. The computation of such solutions without the introduction of any additional approximations, except those associated with the numerical algorithms is called *Direct Numerical Simulation (DNS)*. From a mathematical viewpoint this would seem to be most logically sound, as mathematicians and many other theoreticians normally desire great purity by being truly faithful to the model of problems under treatment. The great majority of mathematical, numerical analysis papers published are of the DNS type.

However, numerical solutions obtained using DNS are of quite limited usefulness. The reason is that fluids exhibit *turbulent behavior*. Turbulence is characterized by *rapid and irregular fluctuations in the fluid properties with a wide range of length and time scales*. A typical occurrence of turbulence is displayed in Figure 1.

The transition of flow from laminar or smooth to turbulent or irregular is determined by the

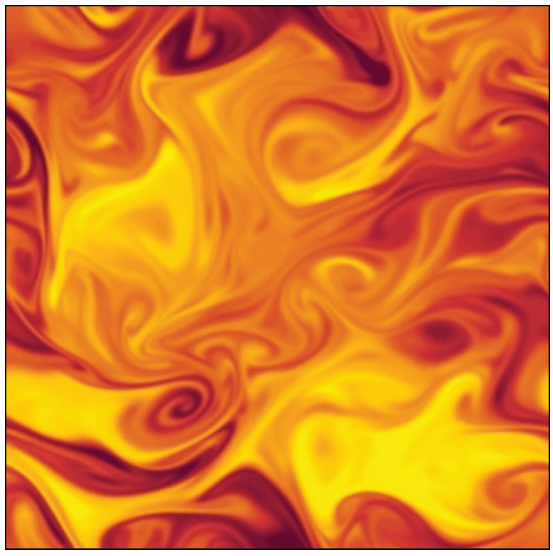


Figure 1. Flow characteristics of turbulence. One can see many swirls (i.e., eddies) of various scales and rapid, irregular fluctuations across the region. *High resolutions* are required for any digital and numerical processing (and also visual display) of turbulence. This signifies an underlying reason why computation is typically large-scale and challenging for turbulent flow. (Excerpted from [22], courtesy of Dr. G. Brethouwer of the Department of Mechanics, KTH, Stockholm, Sweden.)

Reynolds number. For example, in a pipe, transition occurs at a Reynolds number of approximately 2300, where the Reynolds number in this case is defined as

$$Re = \frac{\rho U d}{\mu}, \tag{5}$$

where ρ, μ, d, and U are the fluid density, molecular viscosity, pipe diameter, and average velocity, respectively. Beyond $Re = 4000$ the flow is fully turbulent. The range of length and time scales in a turbulent flow depends on the Reynolds number. Kolmogorov [23] argued that the smallest scales of turbulence should be independent of the largest scales. Dimensional analysis then gives the smallest spatial and time scale, respectively, as

$$\eta = (\frac{\nu^3}{\epsilon})^{\frac{1}{4}}, \ \tau_\eta = (\frac{\nu}{\epsilon})^{\frac{1}{2}}, \tag{6}$$

where ν and ϵ are the fluid kinematic viscosity and average viscous dissipation rate of turbulent energy per unit mass. For turbulence in equilibrium the rate of viscous dissipation at the smallest scales must equal the rate of supply of energy from the large scales. That is, $\varepsilon \sim U^3/L$, where U and L are the largest velocity and length scales of the turbulence. This gives

$$\frac{L}{\eta} \sim \left(\frac{UL}{\nu}\right)^{3/4} = Re^{3/4}. \tag{7}$$

Thus, to simulate all scales of motion in a turbulent flow, the grid size increases as $Re^{9/4}$. Since the Reynolds number in flows of engineering interest are of the order of 10^5 (or much higher in geophysical flows), DNS is of little use in such problems.

To overcome this limitation researchers have resorted to different levels of approximation. This is referred to as *turbulence modeling*. A comprehensive coverage of turbulent flows and turbulence modeling is given by Pope [24]. The two most widely used approaches are *Reynolds-averaged N-S (RANS)* simulations and *large eddy simulation (LES)*; cf., e.g., [24], [5, Sections 3.7 and 3.8]. RANS methods are based on the *time or ensemble averaged* N-S equations. This process results in the appearance of additional terms involving the average of products of the fluctuating velocity, referred to as *Reynolds stresses*. (Additional terms arise in *compressible* turbulent flows where the density fluctuates.) Equations must be developed to describe the Reynolds stresses, of which there are six independent components. These are generally differential equations. The majority of RANS models are based on the concept of an *eddy viscosity*. This is a diffusion coefficient, equivalent to the kinematic viscosity of the fluid, that describes the turbulent mixing or diffusion of momentum. It involves the product of a characteristic turbulent velocity and length scale. *Two-equation turbulence models*, such as the $k-\varepsilon$ and $k-\omega$ models [5, pp. 72–93], provide these scales. Here k is the turbulent kinetic energy per unit mass, ω is the specific dissipation rate, and ε was defined earlier. It should be noted that though exact equations can be developed from the equations of motion for these quantities, additional unknown terms arise that must be modeled. Two other RANS approaches should be mentioned. The first is the one-equation *Spalart-Allmaras model* [5, pp. 89–90]. This involves a differential equation for the eddy viscosity. It was developed specifically for external aerodynamics problems and is not based on modeling terms in the exact equations but on a more general phenomenological approach. The second approach uses *Reynolds stress models*. These involve equations (including modeled terms) for the individual Reynolds stress components.

RANS methods involve empirical models with numerous coefficients that must be specified. In general, these coefficients are valid within a particular class of turbulent flow, for example, wall-bounded or free shear flows. This is because the turbulent mixing is controlled by the large-scale turbulent motions that differ from one class of

Turbulence models	Advantages	Disadvantages
DNS	Most accurate. Doesn't need empirical correlations. Capable of characterizing all the flow details.	Highly computationally expensive. Difficult to include accurate initial and boundary conditions for engineering applications.
LES	Capable of capturing the dynamics of the dominant eddies in the system. Relatively more economical than DNS. More accurate than RANS.	Still computationally intensive. Some difficulties in representing flow in complex geometries.
RANS	Suitable for engineering problems. Computational cost is modest.	Incapable of capturing flow details. High dependence on empirical correlations.
DES	Suitable for engineering problems. Captures unsteadiness in separated flows. More generally applicable than RANS.	Incapable of capturing flow details in near-wall region.

Table 1. Comparisons among DNS, LES, RANS, and DES.

flow to another. A more general approach is LES mentioned earlier, which is based on a *spatial average* of the N-S equations using a *box, Gaussian,* or *spectral cutoff filter*. The action of turbulent scales smaller than the cutoff scale is modeled using a *subgrid scale SGS* model. This is usually in the form of an eddy viscosity model that can involve a constant or dynamic coefficient. In the latter case the eddy viscosity or Smagorinsky constant is allowed to vary in space and time and is calculated based on two filterings of the flow variables. Some averaging is generally required for stability. This could be averaging in a homogeneous flow direction or a local spatial average. LES methods are still computationally expensive, though not as much as DNS. This is especially true for wall-bounded turbulent flows, since the "large" scales close to the wall can be very small. An efficient solution to this problem is the use of *hybrid RANS/LES* models. An example is the *Detached Eddy Simulation (DES)*. In DES the turbulence model equations behave as RANS equations in the near-wall region but transition to LES away from the walls. Clearly such solutions cannot simulate the details of the turbulence in the near-wall regions. We refer the reader to Table 1 for a comparison among these basic turbulence models.

OpenFOAM offers all the turbulence modeling methods described here either as standard solvers or libraries for users to simulate turbulence on the proper spatial and temporal scales.

In the following, to compare the simulation capability of the three basic turbulence modeling strategies, i.e., DNS, LES, and RANS, a simple *lid-driven* flow is simulated in both two- (2D) and three-dimension (3D) in Examples 1 and 2. The lid-driven cavity flow [25] is a classical test problem for N-S codes and benchmarks. Its geometry and boundary conditions are indicated in Figure 2.

Length L	0.1 m
Height H	0.1 m
Width W	0.1 m
Kinematic viscosity ν	0.00001 m^2s^{-1}
Lid velocity U	1 m/s
Grid length in RANS, LES and DNS Δx	0.005 m, 0.001 m, 0.0002 m
Unit time step in RANS, LES and DNS Δt	0.005 s, 0.001 s, 0.0002 s

Table 2. Physical and geometrical parameters for the lid-driven flow simulation.

The parameter values of this problem are summarized in Table 2. The turbulent viscosity submodels chosen for RANS and LES are the standard $k-\varepsilon$ model and k-equation eddy-viscosity

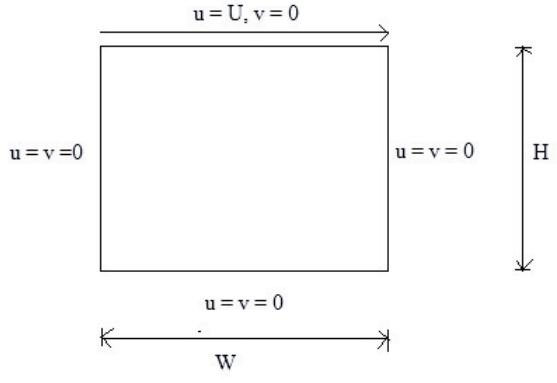
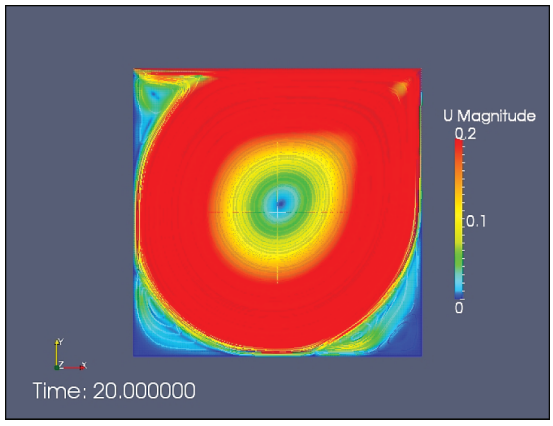
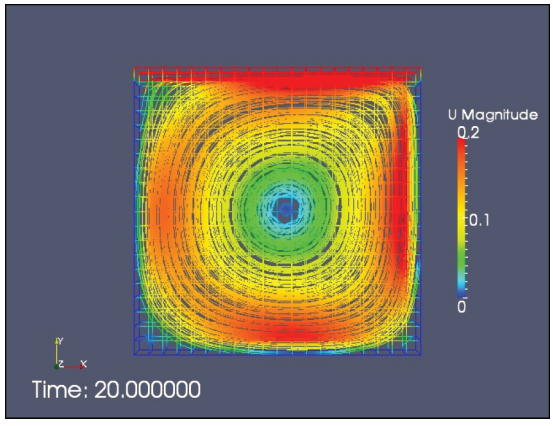


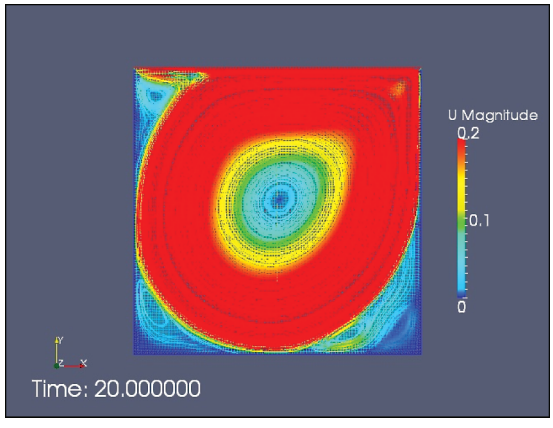
Figure 2. Geometry and boundary conditions for a 2D lid-driven cavity, where (u, v) are the components of flow velocity. The case for 3D is similar. Note that the upper "lid" has a constant horizontal velocity U. Note, however, for the 3D DNS computations, because a huge memory space and CPU time are required, the domain has been reduced to the size of [0, 0.1]×[0, 0.1]×[0, 0.01].



(a) Flow streamlines at t=20 sec obtained by DNS [26].

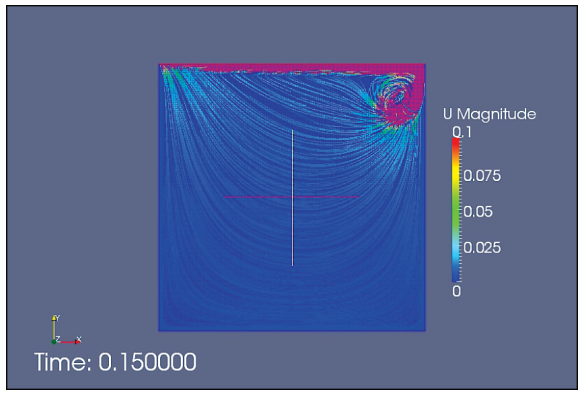


(b) Flow streamlines at t=20 sec obtained by RANS [27].

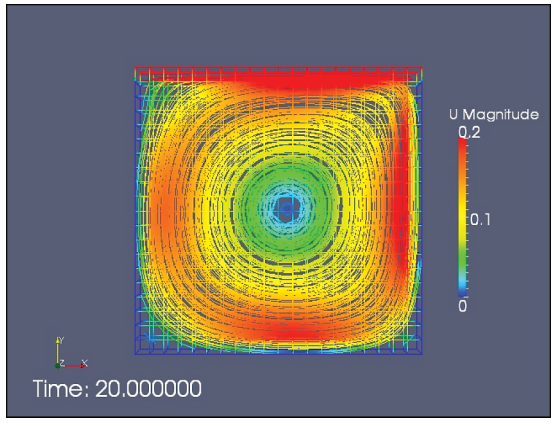


(c) Flow streamlines at t=20 sec obtained by LES [28].

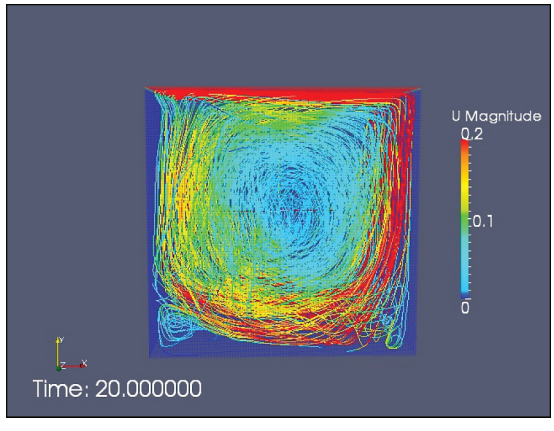
Figure 3. 2D lid-driven flow calculations by OpenFOAM.



(a) Flow field at t=0.15 sec obtained by DNS [26]. (Note that the domain here is [0, 0.1]×[0, 0.1]×[0, 0.01], not a cube as in subcases (b) and (c).)



(b) Flow field at t=20 sec obtained by RANS [27].



(c) Flow field at t=20 sec obtained by LES [28].

Figure 4. 3D lid-driven flow calculations by OpenFOAM. Note that the snapshot of the DNS case in part (a) is at t=0.15, while those in parts (b) and (c) are at t=20. One can rank the richness of fine features of flow in the order of (a), (c) and (b).

model, respectively, in OpenFOAM [27], [28]. Wall functions [5, pp. 76–78] are applied to turbulent viscosity at all wall types. Computations for Examples 1–3 were run on the Texas A&M Supercomputing Facility's Eos, an IBM iDataPlex Cluster 64-bit Linux with Intel Nehalem processors.

Example 1 (2D lid-driven flow)**.** Graphical results are displayed in Figure 3. The numerical data agree favorably with those in the literature (but we omit the details of comparisons here for lack of space).

Example 2 (3D lid-driven flow)**.** The 3D DNS requires huge resources. Here we used 1024 cores for parallel computing at the TAMU Supercomputing Facility to run this case. It took sixty-four hours to run for the numerical simulation for just 0.15 second. The streamline flow pattern computed by DNS at t=0.15 can be seen in Figure 4, part (a).

For 3D RANS and LES computations, we are able to compute flow fields up to t=20 sec; see their flow patterns in parts (b) and (c) of Figure 4.

To *visualize* the *dynamics* of fluid motion, we have made three short animation videos. The reader can see the dynamic motion of the fluid computed by DNS by clicking on (or pasting) `https://www.dropbox.com/s/htoms253d3ckt0n/DNS3Dstreamline2.avi/`, while that computed by OpenFOAM RANS, containing two different graphical representations, field and streamlines, can be viewed at `https://www.dropbox.com/s/6cwjsdxrmcnud3o/RANS3Dfiledstreamline.wmv/`. The counterpart, computed by OpenFOAM LES, can be seen at `https://www.dropbox.com/s/6hzz3ct1wljur9n/LES3Dfiledstreamline.wmv/`.

Example 3 (Flow field of a Grumman F-14 Tomcat fighter)**.** Examples 1 and 2 involve simple geometry and are intended for easy understanding of simple flow patterns and possible benchmarking. Here we present an example with a more complicated geometry, that of a Grumman F-14 Tomcat [29]. We chose this model for the study of landing gear and airframe noise [30], as the grid-generation for the aircraft body had already been performed and was available on the Internet [31] free of charge. We redacted it in [32].

The aircraft is flying in a headwind of 70 m/sec. The kinematic viscosity of air is chosen to be 1.48×10^{-5} m^2/sec. The OpenFOAM solver PimpleFOAM [33] is used. Four processors of TAMU's Supercomputer were used, taking close to four hours of computing time. The flow profile at time=1.2 sec. is plotted in Figure 5.

To visualize the dynamic motion of air flow, go to `https://www.dropbox.com/s/uuhnesg2pxy6c3i/fjet-udiff-vol-g.avi/`.

We note that numerical results computed here should not be accepted too literally as correct or accurate. As a rule, such results should be subject to the scrutiny of model selection criteria, convergence test and error and multiscales analysis, be validated against experiments (such as wind tunnel data, if available), and be corroborated with those obtained from other numerical schemes.

Concluding Remarks

The "open-source" nature of OpenFOAM is of fundamental importance but is not its *only* beneficial feature. There are many advantages to using OpenFOAM:

- codes are extensibie for many customized applications;
- the generality of the various OpenFOAM libraries and solvers empowers the user to solve nearly all CFD problems comprehensively;
- the pre- and postprocessing interfaces are well designed, powerful, and user friendly;
- dynamic meshes (moving grids) can be used and manipulated for dynamically changing geometry;
- object-oriented C^{++} code development strategy makes it convenient for users to incorporate their own submodels.

The *barrier of entry* into OpenFOAM is also quite nontrivial. The website [1] does not provide a systematic user manual for codes. One must learn how to use OpenFOAM through:

- participating in the community-organized international workshop [14];
- attending company-offered tutorial courses, which can easily exceed a couple thousand dollars per person;
- joining online and/or campus Users Groups or CFD communities [12];
- classroom or personal tutorials by experienced users or instructors.

If a prospective user is already familiar with other CFD codes, then the learning curve is usually less steep, as OpenFOAM shares many common features with other codes. Nevertheless, for those mathematicians and engineers not familiar with the C^{++} programming language, they will need to gain familiarity with C^{++} first.

Will the advent and popularity of OpenFOAM and other open-source libraries signal the demise of commercial CFD software? At this time, the answer is firmly negative; however, open-source software does apply pressure on vendors to continue to innovate and evaluate price models, especially with regard to large-scale parallel computing licenses. In our opinion, this answer will also heavily depend upon secondary developers of front- and back-end software to provide graphical-user interfaces [19] and large-scale cloud computing [34], [35], [36].

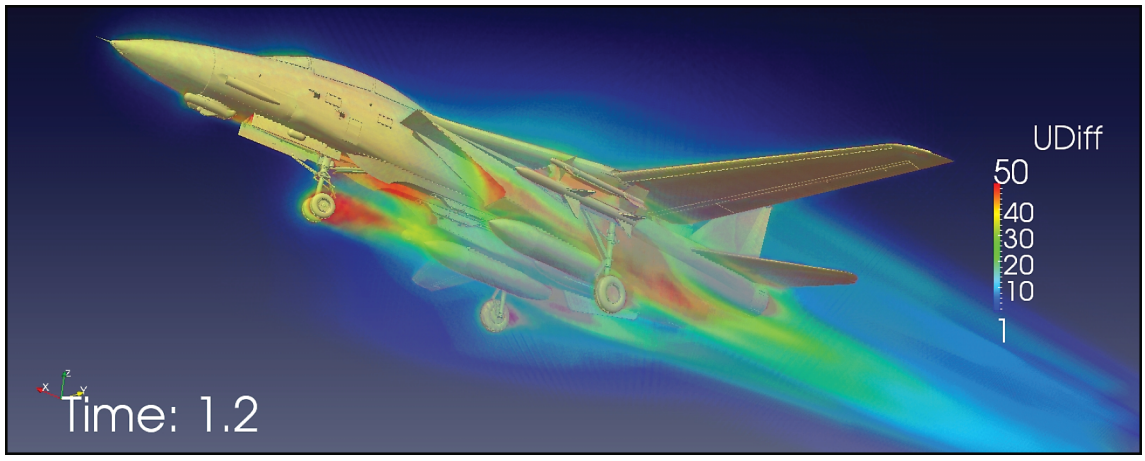


Figure 5. Airflow at t=1.2 of Grumman F-14 Tomcat fighter jet in a headwind of 70 m/sec during takeoff or landing.

After all, for the ultimate health and benefits of the CFD community, it is important to have *diversity* of CFD code developers, whether open source or by commercial vendors.

Only a few years ago the applied mathematicians in this group at Texas A&M would never have imagined that they could compute CFD problems such as those given in Examples 1–3 in this article and beyond. OpenFOAM has really opened up a grand vista for all researchers and empowered them to do CFD and more: general computational mechanics and the affiliated design and optimization problems. This enhances the industrial job prospects for math Ph.D. students up to a par with those of engineering and physics graduates. It also presents numerous opportunities for all researchers to compute, collaborate (in the same computing platform of OpenFOAM), and do interdisciplinary research on complex systems governed by PDEs in nearly every field of science and technology.

Acknowledgements

The authors thank the reviewers for constructive criticisms, and the staff of TAMU's Supercomputing Center for allocation and assistance. They express their gratitude for partial financial support from the Qatar National Research Funds grant National Priority Research Project #5-674-1-114 and Texas Norman Hackman Advanced Research Program Grant #010366-0149-2009 from the Texas Higher Education Coordinating Board.

References

[1] OpenFOAM, http://www.openfoam.com/.
[2] Direct Industry, CFD software, http://www.directindustry.com/industrialmanufacturer/cfdsoftware74890.html.
[3] SourceForge, http://en.wikipedia.org/wiki/SourceForge, http://sourceforge.net, http://sf.net.
[4] GitHub, http://en.wikipedia.org/wiki/GitHub, http://github.com/.
[5] H. K. Versteeg and W. Halalasekera, *An Introduction to Computational Fluid Dynamics—The Finite Volume Method*, 2nd ed., Pearson Prentice Hall, Harlow, UK, 2007.
[6] Deal.II, http://en.wikipedia.org/wiki/Deal.II.
[7] Stanford University Unstructured (SU2), http://en.wikipedia.org/wiki/Stanford_University_Unstructured.
[8] A. Perronnet, Méfisto, http://www.ann.jussieu.fr/perronnet/mefistoa.gene.html.
[9] MFIX, https://mfix.netl.doe.gov/.
[10] Redhat, http://en.wikipedia.org/wiki/RedHat.
[11] The Cathedral and the Bazaar, http://en.wikipedia.org/wiki/The_Cathedral_and_the_Bazaar.
[12] OpenFOAM discussion forum, http://www.cfd-online.com/Forums/openfoam/.
[13] OpenFOAM Wiki, http://openfoamwiki.net/index.php/Main_Page.
[14] OpenFOAM Workshop, http://www.openfoamworkshop.org/.
[15] Lid-driven cavity flow (in OpenFOAM), http://openfoam.org/docs/user/cavity.php.
[16] R. I. Issa, Solution of the implicitly discretized fluid flow equations by operator-splitting, *J. Comp. Phys.* **62** (1986), 40–65.
[17] Pressure-correction method, http://www.en.wikipedia.org/wiki/Pressure-correction_method.
[18] Gambit meshing software, http://www.cfd-online.com/Forums/ansys-meshing/114276-gambit-ansys-meshing-software.html.
[19] Graphical user interfaces for OpenFOAM, http://openfoamwiki.net/index.php/GUI.
[20] Paraview, http://en.wikipedia.org/wiki/ParaView.
[21] EnSight, http://ensight.com/.
[22] eFluids gallery image of turbulence, http://www.efluids.com/efluids/gallery/gallery_pages/iso_turbulence_page.htm.
[23] Kolmogorov microscales, http://www.en.wikipedia.org/wiki/Kolmogorov_microscales.

[24] S. B. Pope, *Turbulence Flows*, Cambridge University Press, Cambridge, UK, 2000.
[25] Lid-driven cavity problem, http://www.cfd-online.com/Wiki/Lid-driven_cavity_problem.
[26] OpenFOAM DNS, http://en.wikipedia.org/wiki/Direct_numerical_simulation.
[27] OpenFOAM RANS, http://en.wikipedia.org/wiki/Turbulence_modelling.
[28] OpenFOAM LES, http://en.wikipedia.org/wiki/Large_eddy_simulation.
[29] Grumman F-14 Tomcat fighter jet, http://en.wikipedia.org/wiki/Grumman_F-14_Tomcat.
[30] P. J. Morris, G. Chen, and A. Sergeev, Modeling and numerical simulation of landing gear and airframe noise, work in progress.
[31] DX123, 5 Military Aviation Models for Element 3D, http://thepiratebay.se/torrent/7666716/5_Military_Aviation_Models_for_Element_3D/.
[32] A. Sergeev, Redacted Grumman F-14 Tomcat fighter-jet CAD, http://www.dropbox.com/s/2i498Ozzpjwfcp9/orbit-fjet.avi/, http://www.dropbox.com/s/uuhnesg2pxy6c3i/fjet-udiff-vol-g.avi/.
[33] OpenFOAM PimpleFOAM, http://www.openfoam.com/features/standard-solvers.php.
[34] Ciespace, http://www.ciespace.com/.
[35] Amazon EC2, http://aws.amazon.com/ec2/.
[36] Sabalcore, http://www.sabalcore.com/OF_details_engineering.html.
[37] A list of finite element software packages: http://en.wikipedia.org/wiki/List_of_finite_element_software_packages.