Geosci. Model Dev., 8, 533–547, 2015
www.geosci-model-dev.net/8/533/2015/
doi:10.5194/gmd-8-533-2015

Geoscientific Model Development
Open Access

Firedrake-Fluids v0.1: numerical modelling of shallow water flows using an automated solution framework

C. T. Jacobs and M. D. Piggott

Department of Earth Science and Engineering, Imperial College London, London SW7 2AZ, UK

Correspondence to: C. T. Jacobs (c.jacobs10@imperial.ac.uk)

Received: 14 August 2014 – Published in Geosci. Model Dev. Discuss.: 27 August 2014
Revised: 22 January 2015 – Accepted: 9 February 2015 – Published: 9 March 2015

Abstract. This model description paper introduces a new finite element model for the simulation of non-linear shallow water flows, called Firedrake-Fluids. Unlike traditional models that are written by hand in static, low-level programming languages such as Fortran or C, Firedrake-Fluids uses the Firedrake framework to automatically generate the model's code from a high-level abstract language called Unified Form Language (UFL). By coupling to the PyOP2 parallel unstructured mesh framework, Firedrake can then target the code towards a desired hardware architecture to enable the efficient parallel execution of the model over an arbitrary computational mesh. The description of the model includes the governing equations, the methods employed to discretise and solve the governing equations, and an outline of the automated solution process. The verification and validation of the model, performed using a set of well-defined test cases, is also presented along with a road map for future developments and the solution of more complex fluid dynamical systems.

1 Introduction

Traditional approaches to numerical model development involve the production of hand-written, low-level (e.g. C or Fortran) code for the specific set of equations that need to be solved. This task alone can be highly error-prone, often resulting in sub-optimal code, and can make the efficiency, readability, and longevity of the code base difficult to maintain (Rognes et al., 2013; Farrell et al., 2013; Mortensen et al., 2011; Maddison and Farrell, 2014). Moreover, parallelisation of the code is usually accomplished by introducing explicit calls to parallel programming libraries such as OpenMP or CUDA (Compute Unified Device Architecture). By doing so, computational scientists are frequently faced with the additional task of having to re-write their model's code as new parallel hardware architectures and platforms emerge. At the current rate that new hardware is introduced, this development workflow is unsustainable and places an infeasible requirement on the developer to be not only a subject/domain specialist adept in computational methods but also well versed in software engineering and parallelisation principles. A change to the traditional programming paradigm is clearly necessary if numerical model development is to continue in a sustainable manner.

Recent investigations into the use of automated solution techniques have shown great potential in mitigating some of the issues faced with traditional approaches to writing numerical models. The FEniCS project (Logg et al., 2012) is a well-known example of a framework which uses such a solution technique to automatically generate low-level model code to solve ordinary and partial differential equations (using the finite element method) from a near-mathematical high-level language, rather than the user having to write the low-level code themselves. This hides complexity through abstraction, and allows users to focus only on the problem specification and the end results of simulations. Furthermore, optimal or near-optimal performance can be achieved through code optimisations that would be tedious to implement by hand (Ølgaard and Wells, 2010). These benefits have been realised in numerous applications in the geosciences. For example, the use of the FEniCS framework by Maddison and Farrell (2014) allowed the runtime of their adjoint models to be as small as (or even smaller than) an equivalent model generated and optimised by hand. Also, the extension of FEniCS by Rognes et al. (2013) to solve partial differential equations on the sphere permits ocean and atmospheric models to be written with just a few lines of high-level intuitive code (although the potential of writing fewer, more intuitive lines of model code is not unique to automated code generation approaches, as demonstrated by the interfaces of other modelling frameworks such as OpenFOAM (OpenFOAM, 2014), deal.II (Bangerth et al., 2007), Dune (Dedner et al., 2010) and FreeFem++ (Hecht, 2012)). Several other application areas using automated solution techniques have demonstrated similar benefits (see, e.g., the works by Farrell et al., 2013; Funke and Farrell, 2013; Logg et al., 2012).

The Firedrake project aims to further extend the abstractions offered by automated solution approaches, by creating a separation of concerns between the automated low-level discretisation of the model equations and its execution on the underlying computational mesh (Rathgeber et al., 2015), whilst still keeping the same high-level problem solving interface for end users. This provides the potential for easier portability of the generated code across different hardware platforms (e.g. multi- and many-core CPUs and GPUs), as well as the efficient handling of computations over a given mesh topology (e.g. taking advantage of the semi-structured nature of a three-dimensional layered mesh extruded in the vertical, as often employed in ocean/atmospheric applications). This is achieved by interfacing with the PyOP2 parallel unstructured mesh computation framework, which targets the automatically generated code towards specific high-performance computing platforms (Rathgeber et al., 2012; Markall et al., 2013; Luporini et al., 2015). In addition, the enhanced abstraction-based approach employed by Firedrake can also help future-proof models from hardware changes and removes a great deal of effort required by computational scientists to maintain the code base.

In light of the issues surrounding the use of static, hand-coded numerical models and the benefits that the Firedrake framework can bring, a new numerical model called Firedrake-Fluids has been developed for computational fluid dynamics (CFD)-related applications. The long-term goal of the project is to facilitate a re-engineering of Fluidity (Piggott et al., 2008), another CFD package (also developed at Imperial College London) comprising hand-written Fortran code whose efficiency, readability, and longevity has become challenging to maintain as the package has grown over many years. In contrast to Fluidity, Firedrake-Fluids has been written in the high-level Unified Form Language (UFL) and uses Firedrake to automate the solution process. Currently it is capable of solving the non-linear shallow water equations which are widely used in the ocean modelling community for applications such as tidal turbine dynamics (Divett et al., 2013; Kramer et al., 2014; Martin-Short et al., 2015), array optimisation (Funke et al., 2014), tsunami modelling (Hill et al., 2014), flow dynamics over submerged islands (Lloyd and Stansby, 1997), and dam breaching and flooding (Capart and Young, 1998). In addition to the core model, Firedrake-Fluids offers upwind stabilisation methods, a variety of diagnostic fields, and the Smagorinsky large-eddy simulation (LES) model (Smagorinsky, 1963) for the parameterisation of turbulence.

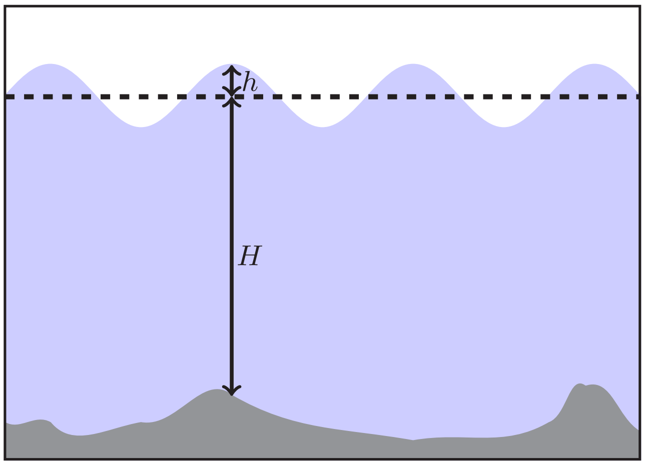


Figure 1. Diagram showing the mean free surface height H (also known as the depth or the distance to the seabed, shaded grey) and the free surface perturbation h, within the shallow water model.

Section 2 details the set of equations that are solved and the assumptions under which they are valid. Section 3 describes the numerical methods that are used to discretise and solve the governing equations, followed by an overview of the automated solution techniques employed by the Firedrake framework. Section 4 presents results from a suite of test cases used to verify the correctness of the numerical model's implementation, and show how well it describes the physics. A discussion regarding the future developments and direction of Firedrake-Fluids is presented in Sect. 5, along with some concluding remarks in Sect. 6.

2 Model equations

The model described in this paper solves the non-linear, non-rotational shallow water equations. These are a set of depth-averaged equations which model the dynamics of a free surface and an associated depth-averaged velocity field (Zhou, 2004). This velocity field is denoted by $\boldsymbol{u} = \boldsymbol{u}(x, y, t)$ (where x and y are the spatial coordinates, and t is time). For modelling purposes, the free surface is split up into a mean component $H = H(x, y)$ and a perturbation component $h = h(x, y, t)$ as illustrated in Fig. 1. Note that h is generally assumed to be much smaller than H.

The shallow water equation set comprises a momentum equation and a continuity equation, each of which are defined below. The unknown fields $\boldsymbol{u}$ and h are sought.

2.1 Momentum equation

The momentum equation is solved in non-conservative form such that

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -g \nabla h + \nabla \cdot \mathbb{T} - C_\mathrm{D} \frac{||\boldsymbol{u}||_2 \boldsymbol{u}}{(H+h)}, \tag{1}$$

where t is time, g is the acceleration due to gravity (set to $9.8\,\mathrm{ms}^{-2}$ throughout this paper), and C_D is a non-dimensional drag coefficient. The Euclidean norm $||\boldsymbol{u}||_2 = \sqrt{\boldsymbol{u} \cdot \boldsymbol{u}}$ is used here, and throughout the rest of this paper. The stress tensor $\mathbb{T}$ is given by

$$\mathbb{T} = \nu \left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^\mathrm{T} \right) - \frac{2}{3} \nu \left(\nabla \cdot \boldsymbol{u} \right) \mathbb{I}, \tag{2}$$

where ν is the kinematic viscosity, which is assumed to be isotropic here, and $\mathbb{I}$ is the identity tensor[1].

2.2 Continuity equation

The continuity equation is given by

$$\frac{\partial h}{\partial t} + \nabla \cdot \left((H+h) \boldsymbol{u} \right) = 0. \tag{3}$$

2.3 Initial and boundary conditions

In order to advance the equations forward in time, initial conditions for the prognostic fields h and $\boldsymbol{u}$

$$h(x, y, t=0) = h^0, \boldsymbol{u}(x, y, t=0) = \boldsymbol{u}^0, \tag{4}$$

must be specified.

Throughout the simulation, values of the free surface perturbation field h may be enforced (in the strong sense) at the boundary using a Dirichlet boundary condition

$$h = h^\mathrm{D} \text{ on } \Gamma, \tag{5}$$

where $\Gamma \subset \Omega$ is the portion of the boundary on which the boundary condition (h^D in this case) is applied. Note that this boundary condition can vary both in time and in space.

In addition to the standard Dirichlet boundary condition for the velocity field

$$\boldsymbol{u} = \boldsymbol{u}^\mathrm{D} \text{ on } \Gamma, \tag{6}$$

[1]The interior penalty method (Arnold, 1982) is applied to the stress term when using discontinuous basis functions for the velocity field (see Sect. 3.2), since the gradient of velocity has to be treated carefully at the boundaries between discontinuous elements. Although it is possible to extend the UFL implementation to the full stress tensor, the current implementation of the method restricts the form to $\mathbb{T} = \nu \nabla \boldsymbol{u}$ for simplicity. In the near-future when tensor function spaces are supported in the Firedrake library, the Bassi–Rebay method (Bassi and Rebay, 1997) will be implemented instead and will consider the full form of the stress tensor.

where $\boldsymbol{u}^\mathrm{D}$ is the value of the velocity to be enforced at the boundary, there are two other conditions that may be applied. The no-normal flow condition enforces

$$\boldsymbol{u} \cdot \boldsymbol{n} = 0 \text{ on } \Gamma, \tag{7}$$

where $\boldsymbol{n}$ is the unit normal vector.

The Flather (1976) boundary condition enforces

$$\boldsymbol{u} - \boldsymbol{u}_* = \sqrt{\frac{g}{H}} (h - h_*) \text{ on } \Gamma, \tag{8}$$

where $\boldsymbol{u}_*$ and h_* are the expected velocity and free surface perturbation exterior to the domain, respectively. Any difference between the expected and simulated free surface is allowed to propagate out of the domain, thereby minimising spurious reflections from the boundary. Note that both of these boundary conditions can only be applied in the weak sense; the velocity value must be applied in the surface integral term, which only appears if the divergence term in the continuity equation is integrated by parts. An option for doing this is available in the simulation configuration file, discussed later in Sect. 3.4.

2.4 Turbulence modelling

The core shallow water model on its own has no way of capturing the effects of turbulence, unless the underlying mesh is of a suitably fine resolution to perform a direct numerical simulation at all turbulence length scales. This is often prohibitively expensive, and so turbulence parameterisation is required. The Smagorinsky LES model represents one possible way of doing this. It parameterises the turbulence via an eddy viscosity (Smagorinsky, 1963; Deardorff, 1970)

$$\nu' = (C_\mathrm{s} \Delta_\mathrm{e})^2 |\mathbb{S}|, \tag{9}$$

where C_s is the Smagorinsky coefficient, and Δ_e is an estimate of the local mesh size which is defined here as the square root of the area of each element (in the 2-D case). $|\mathbb{S}|$ is the modulus of the strain rate tensor defined by

$$\mathbb{S} = \frac{1}{2} \left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^\mathrm{T} \right), |\mathbb{S}| = \sqrt{2 \sum_i \sum_j \mathbb{S}_{ij} \mathbb{S}_{ij}}, \tag{10}$$

where $\mathbb{S}_{ij}$ is the $(i,\, j)$th component of $\mathbb{S}$. The eddy viscosity ν', which models the dissipating effects of small-scale turbulent eddies on the resolved flow, is added to the physical viscosity ν in the stress term of the momentum equation.

3 Methods

3.1 Automated code generation

Solving a given set of equations in the Firedrake framework requires only the weak forms of the model equations (along

```
from firedrake import *

# The mesh (a unit square containing 10 x 10 vertices)
mesh = UnitSquareMesh(50, 50)

# Function spaces (simply piecewise continuous linear basis functions here)
fs = FunctionSpace(mesh, "Lagrange", 1)
vfs = VectorFunctionSpace(mesh, "Lagrange", 1)

# Test and trial functions in the variational formulation
w = TestFunction(fs)
c = TrialFunction(fs)

# Initial condition for c (a Gaussian situated in the centre of the domain)
c_initial = Expression('''exp(-( pow(x[0]-x0, 2)/(2*pow(spread_x, 2)) +
                        pow(x[1]-y0, 2)/(2*pow(spread_y, 2)) ))''',
                        x0=0.5, y0=0.5, spread_x=0.075, spread_y=0.075)
# c_old is the old solution of the field 'c'.
# This holds the initial condition at time t = 0.
# Here we evaluate the expression c_initial and interpolate the values
# onto the solution nodes for c_old.
c_old = Function(fs).interpolate(c_initial)

# A constant velocity field to advect c with.
u_initial = Expression(("0.1", "0.0"))
# Interpolate the values of u_initial onto the nodes in the mesh.
u = Function(vfs).interpolate(u_initial)

# Time-step size
dt = 0.1

# Diffusion coefficient
k = 1e-3

# The variational/weak form of the advection-diffusion equation
F = (1.0/dt)*(inner(w, c) - inner(w, c_old))*dx + inner(w*u, grad(c))*dx \
    + k*inner(grad(w), grad(c))*dx

# Output file containing the solution for 'c'
out = File("gaussian.pvd")
solution = Function(fs)

# Time-stepping loop
t = 0; T = 5.0 # Current time and finish time
while t <= T:
   t += dt; print t

   # Solve the system of equations
   solve(lhs(F) == rhs(F), solution)
   out << solution

   # Update c_old
   c_old.assign(solution)
```

Figure 2. Sample Python code which uses the high-level Unified Form Language (UFL) to solve the advection–diffusion equation with the finite element method. The solution field c has a Gaussian profile at $t = 0$, which is then advected with a prescribed velocity field $\boldsymbol{u} = [0.1, 0]^{\mathrm{T}}$.

with associated boundary and initial conditions) to be discretised (both temporally and spatially) and expressed in a near-mathematical language called UFL, an embedded language that uses Python as its host (Alnæs et al., 2014). An example of a model defined in UFL which solves a two-dimensional advection–diffusion problem is given in Fig. 2 (with associated results in Fig. 3), and highlights how the implementation can be accomplished with just a few lines of intuitive statements. This one file containing approximately 50 lines of UFL is automatically compiled into over 600, much more complicated, lines of low-level C code which are executed over the entire mesh by PyOP2 to perform the assembly of the finite element system.

The UFL code is compiled at runtime, using a modified version of the FEniCS form compiler (FFC)[2] (Kirby and Logg, 2006; Luporini et al., 2015), into an intermediate representation as an abstract syntax tree (AST) before being passed into the PyOP2 library, as shown in Fig. 4. Furthermore, optimal numbering of the solution nodes in the domain is important to avoid cache misses and ensure efficient computation; therefore, the topology of any mesh that is provided by the user (e.g. from the Gmsh mesh generator (Geuzaine and Remacle, 2009)) is described using a PETSc (Portable, Extensible Toolkit for Scientific Computation) DMPlex object which is also passed to PyOP2 along with the AST. PyOP2 then performs additional optimisations on the AST using the COFFEE (COmpiler For FinitE Element local assembly) compiler (Luporini et al., 2015) which outputs the model's optimised low-level C code. Finally, PyOP2 calls a back-end compiler (e.g. GNU gcc or the Intel C compiler for CPUs) to compile the generated code on demand at runtime (known as just-in-time compilation), and then executes it efficiently over the entire domain. As previously mentioned, in addition to targeting the code towards multi-core CPUs, PyOP2 can also target the generated code towards a specific parallel platform using, for example, the PyOpenCL and PyCUDA compilers for GPUs. Note that, however, as a result of current implementation restrictions (e.g. the solution of non-linear problems is not yet possible with PyOP2 on GPUs) the work presented in this paper only considers the compilation of code using the GNU gcc compiler on CPUs.

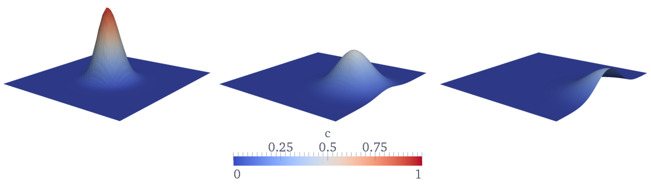


Figure 3. Visualisation of the solution field c at $t = 0$, 2.5, and 5 s from the advection–diffusion problem defined in Fig. 2. The initial Gaussian profile is advected from left to right, out of the domain, and slowly diffuses over time. The field has been warped in the z direction.

3.2 Spatial and temporal discretisation

The spatial discretisation of the model equations is performed using the Galerkin finite element method. The first step of the method involves deriving the variational/weak form of the model equations by multiplying them by a so-called test function $\boldsymbol{w} \in H^1(\Omega)^3$, where $H^1(\Omega)^3$ is the first

[2]The original version of FFC, which is part of the FEniCS project, compiles the UFL into low-level C++ code called UFC (Kirby and Logg, 2006; Logg and Wells, 2010), whereas the modified version in Firedrake first compiles the UFL into an abstract syntax tree (AST) for further manipulation and optimisation by the PyOP2 framework (Luporini et al., 2015). The modified version of FFC is available from the MAPDES Bitbucket repository: https://bitbucket.org/mapdes/ffc. Revision `6c0d70d` in the `master` branch was used when performing the simulations presented in this paper.

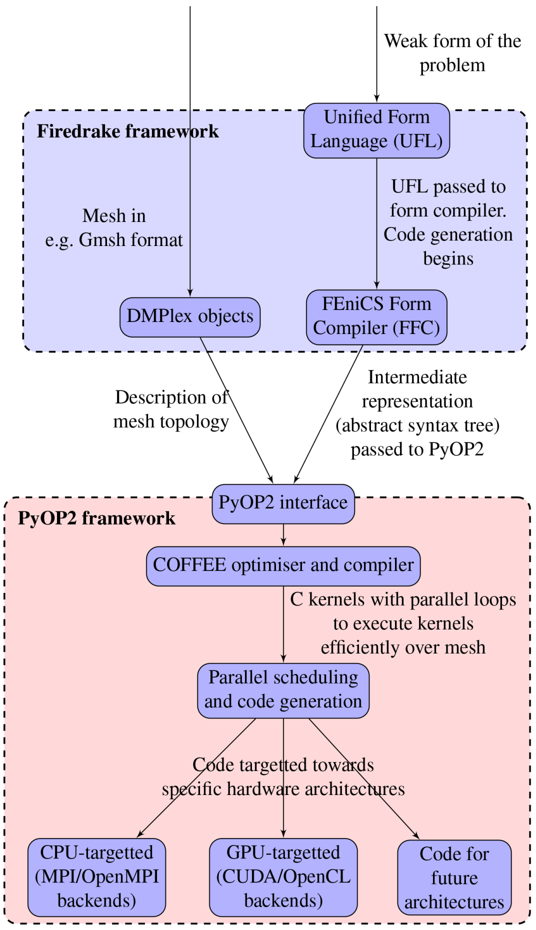

Weak form of the problem
Firedrake framework
Unified Form Language (UFL)
Mesh in e.g. Gmsh format
UFL passed to form compiler. Code generation begins
DMPlex objects
FEniCS Form Compiler (FFC)
Description of mesh topology
Intermediate representation (abstract syntax tree) passed to PyOP2
PyOP2 framework
PyOP2 interface
COFFEE optimiser and compiler
C kernels with parallel loops to execute kernels efficiently over mesh
Parallel scheduling and code generation
Code targetted towards specific hardware architectures
CPU-targetted (MPI/OpenMPI backends)
GPU-targetted (CUDA/OpenCL backends)
Code for future architectures

Figure 4. Overview of the key components of the Firedrake and PyOP2 frameworks (Rathgeber, 2014).

Hilbertian Sobolev space (Elman et al., 2005), and integrating them over the whole domain Ω; this yields, in the case of the momentum equation, Eq. (1),

$$\int_\Omega \boldsymbol{w}\cdot\frac{\partial \boldsymbol{u}}{\partial t}\,\mathrm{d}V + \int_\Omega \boldsymbol{w}\cdot(\boldsymbol{u}\cdot\nabla\boldsymbol{u})\,\mathrm{d}V = -\int_\Omega g\boldsymbol{w}\cdot\nabla h\,\mathrm{d}V - \int_\Omega \nabla\boldsymbol{w}\cdot\mathbb{T}\,\mathrm{d}V - \int_\Omega C_\mathrm{D}\boldsymbol{w}\cdot\frac{||\boldsymbol{u}||_2\boldsymbol{u}}{(H+h)}\,\mathrm{d}V.$$

Note that the stress term has been integrated by parts and it is assumed that the normal stress gradient at all boundaries is zero. In this weak form, a solution $\boldsymbol{u}\in H^1(\Omega)^3$ is sought for all $\boldsymbol{w}\in H^1(\Omega)^3$.

The test function and the solution $\boldsymbol{u}$ (also known as the trial function) are then replaced by discrete representations, given by a linear combination of basis functions $\{\phi_i\}_{i=1}^{N_\mathrm{u\_nodes}}$ which may be continuous or discontinuous across the cells/elements of the mesh:

$$\boldsymbol{w} = \sum_{i=1}^{N_\mathrm{u\_nodes}} \phi_i \boldsymbol{w}_i, \tag{11}$$

$$\boldsymbol{u} = \sum_{i=1}^{N_\mathrm{u\_nodes}} \phi_i \boldsymbol{u}_i, \tag{12}$$

where $N_\mathrm{u\_nodes}$ is the number of velocity solution nodes in the mesh, $\boldsymbol{w}_i$ are arbitrary, and the coefficients $\boldsymbol{u}_i$ are sought using a numerical solution method. The free surface perturbation field h, which needs to be solved for in addition to the velocity field, is also represented by a (possibly different) set of basis functions $\{\psi_i\}_{i=1}^{N_\mathrm{h\_nodes}}$:

$$h = \sum_{i=1}^{N_\mathrm{h\_nodes}} \psi_i h_i, \tag{13}$$

where $N_\mathrm{h\_nodes}$ is the number of free surface solution nodes, and h_i are the coefficients to be found.

The discrete system of size $N_\mathrm{u\_nodes}\times N_\mathrm{u\_nodes}$ for the momentum equation then becomes

$$\mathbf{M}\frac{\partial \boldsymbol{u}}{\partial t} + \mathbf{A}(\boldsymbol{u})\boldsymbol{u} = -\mathbf{C}h - \mathbf{K}\boldsymbol{u} - \mathbf{D}(\boldsymbol{u},h)\boldsymbol{u}, \tag{14}$$

where **M**, **A**, **K**, **C** and **D** are the mass, advection, stress, gradient, and drag discretisation matrices, respectively. The notations $\mathbf{A}(\boldsymbol{u})$ and $\mathbf{D}(\boldsymbol{u},h)$ are used to highlight the non-linear dependence of the matrices on the velocity and free surface fields. A similar process is performed for the continuity equation, Eq. (3), resulting in a full block-coupled system.

The temporal discretisation, performed using the implicit backward Euler method, yields

$$\mathbf{M}\frac{\boldsymbol{u}^{n+1}-\boldsymbol{u}^n}{\Delta t} + \mathbf{A}(\boldsymbol{u}^{n+1})\boldsymbol{u}^{n+1} = -\mathbf{C}h^{n+1} - \mathbf{K}\boldsymbol{u}^{n+1} - \boldsymbol{D}(\boldsymbol{u}^{n+1},h^{n+1})\boldsymbol{u}^{n+1},$$

where the superscript n represents the current time level and $n+1$ represents the next time level. The backward Euler method gives first-order accuracy in time. Newton iteration is employed to deal with the non-linearity introduced via the advection and drag terms, although this does not need to be implemented explicitly by the model developer; instead, it can be performed using a PETSc Scalable Nonlinear Equations Solvers (SNES) object. Other temporal discretisation approaches such as the Crank–Nicolson method can be readily implemented in UFL, but are not currently available in Firedrake-Fluids.

A wide variety of basis functions of arbitrary order are available through FIAT (the finite element automatic tabulator) (Kirby, 2004). For the simulations presented in this paper, only the P2–P1 (i.e. piecewise-quadratic basis functions representing the velocity field and piecewise-linear basis functions representing the free surface field) and P0–P1 (i.e. piecewise-constant (discontinuous) basis functions for velocity and piecewise-linear basis functions for the free surface) element pairs will be considered. Unless otherwise stated, the P2–P1 element pair will be used in preference to P0–P1, in order to obtain higher-order solutions. However, users are free to choose the order and continuity of the basis functions through the simulation configuration file (discussed in Sect. 3.4). Currently, Firedrake-Fluids only allows Lagrange polynomial basis functions to be used, although other basis function families are available through FIAT (e.g. Raviart–Thomas).

In Sect. 2, it was mentioned that the form of the stress tensor is currently restricted in the case of using discontinuous basis functions. In the future, once tensor function spaces become available in the Firedrake framework, the Bassi–Rebay method (Bassi and Rebay, 1997) will be implemented instead and will consider the full form of the stress tensor. In addition, some more complicated numerical techniques cannot be formulated in the UFL language, such as slope limiters. However, it is possible to implement them in lower-level code in the form of a PyOP2 C kernel which interacts directly with the nodal data to accomplish this.

3.3 Solution methods

Firedrake assembles the full block-coupled form of the discrete system of linear equations, and attempts to solve it using the PETSc library (Balay et al., 2014). PETSc contains a variety of linear solvers and preconditioners, and has proven itself in facilitating geoscientific model development (Katz et al., 2007). It is possible to use, for example, the generalised minimal residual method (GMRES) or conjugate gradient iterative method, and preconditioners such as Jacobi and SOR (successive over-relaxation). For the simulations presented in this paper, the GMRES linear solver (Saad and Schultz, 1986) is chosen and used in conjunction with the fieldsplit preconditioner (Brown et al., 2012) which is especially suited to block-coupled systems such as the one considered here.

The block-coupled system takes the general form

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{h} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_{\mathrm{u}} \\ f_{\mathrm{h}} \end{bmatrix} \tag{15}$$

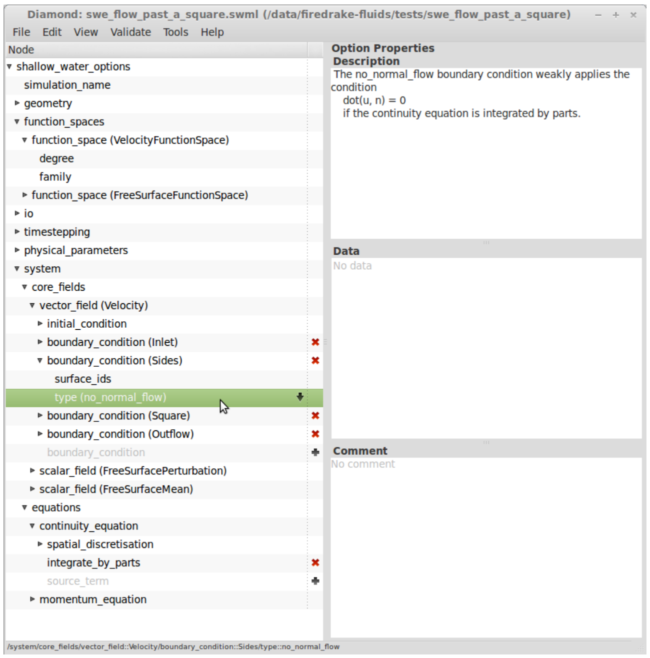
for matrix blocks $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$, and right-hand sides $\mathbf{f}_{\mathrm{u}}$ and f_{h}.

The matrix on the left-hand side can be factorised using LDU (lower-diagonal-upper) block factorisation to give (Elman et al., 2008)

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{C}\mathbf{A}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{S} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \tag{16}$$

where $\mathbf{S} = \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}$ is the Schur complement. The inverse of the factorised system is given by

$$P = \begin{bmatrix} \mathbf{I} & -\mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{C}\mathbf{A}^{-1} & \mathbf{I} \end{bmatrix}. \tag{17}$$



```
Diamond: swe_flow_past_a_square.swml (/data/firedrake-fluids/tests/swe_flow_past_a_square)
File  Edit  View  Validate  Tools  Help
Node
▾ shallow_water_options
    simulation_name
  ▸ geometry
  ▾ function_spaces
    ▾ function_space (VelocityFunctionSpace)
        degree
        family
    ▸ function_space (FreeSurfaceFunctionSpace)
  ▸ io
  ▸ timestepping
  ▸ physical_parameters
  ▾ system
    ▾ core_fields
      ▾ vector_field (Velocity)
        ▸ initial_condition
        ▸ boundary_condition (Inlet)
        ▾ boundary_condition (Sides)
            surface_ids
            type (no_normal_flow)
        ▸ boundary_condition (Square)
        ▸ boundary_condition (Outflow)
          boundary_condition
      ▸ scalar_field (FreeSurfacePerturbation)
      ▸ scalar_field (FreeSurfaceMean)
    ▾ equations
      ▾ continuity_equation
        ▸ spatial_discretisation
          integrate_by_parts
          source_term
      ▸ momentum_equation

Option Properties
Description
The no_normal_flow boundary condition weakly applies the
condition
   dot(u, n) = 0
   if the continuity equation is integrated by parts.

Data
No data

Comment
No comment

/system/core_fields/vector_field::Velocity/boundary_condition::Sides/type::no_normal_flow
```

Figure 5. The Diamond (Ham et al., 2009) graphical user interface for editing Firedrake-Fluids simulation configuration files.

It is the goal of the fieldsplit preconditioner to find approximations to the actions of $\mathbf{S}^{-1}$ and $\mathbf{A}^{-1}$ which will in turn give an approximation to the action of $\mathbf{P}$ which can be used to precondition the block-coupled system. PETSc features a wide configuration space for its fieldsplit preconditioner, permitting the use of different iterative (or direct) solvers and preconditioners for the sub-problems $\mathbf{S}^{-1}$ and $\mathbf{A}^{-1}$. Unless stated otherwise, incomplete LU (ILU) factorisation will be used as an approximate solver for $\mathbf{S}^{-1}$ and $\mathbf{A}^{-1}$ in all simulations presented here (except when running in parallel, where block Jacobi is applied globally and the individual blocks are solved sequentially using ILU (Balay et al., 2014)). The convergence criterion for all iterative solvers is a relative error of 10^{-7}.

3.4 Set-up and execution

Firedrake-Fluids uses an XML-based configuration file, normally edited with the Diamond graphical user interface (GUI) (Ham et al., 2009), to set up simulations. Users can enter options concerning the simulation's name, the path to any input files (e.g. mesh files), the fields to be solved, discretisation and linear solver options, and also the inclusion of auxiliary models such as the Smagorinsky LES model (Smagorinsky, 1963). In addition, initial and boundary conditions for each field can be specified either as a constant value, or as a C++ expression for time-varying or spatially varying conditions. An example of the GUI is shown in Fig. 5. In the case of the shallow water model, all simulation configuration files have the extension .swml (shallow water markup language).

All UFL model code is stored in the `models` directory of Firedrake-Fluids. Execution of, for example, the shallow water model is performed by calling the Python interpreter and providing the path to the simulation configuration file; an example for the test case involving flow past a square cylinder (discussed in Sect. 4) would be

```
 python models/shallow_water.py
tests/swe_flow_past_a_square/
swe_flow_past_a_square.swml.
```

Simulation settings are first read in using the `libspud` library (Ham et al., 2009). This is followed by the execution of the UFL statements which define the model. Note that the weak form of the shallow water equations is defined in UFL only once, before entering the time-stepping loop. Upon entering the time-stepping loop for the first time, the form is compiled and the low-level assembly code is generated. For subsequent time steps, caching is used such that no re-compilation of the UFL is necessary. Solution fields are currently written to files in Visualization Toolkit format for visualisation.

Note that the UFL code for the LES model described in Sect. 2.4 is defined in a separate class within the Firedrake-Fluids package (in the file `les.py`) for modularity, and to facilitate its re-use in future numerical models that may require turbulence parameterisation. In the case of the shallow water model implemented in `shallow_water.py`, the UFL for the left-hand side and right-hand side of the eddy viscosity equation, Eq. (9), is first imported, and a separate solver computes the eddy viscosity field at the start of each time step, using the velocity from the previous time step. The viscosity used in the stress term is then updated, but doing so does not require the re-compilation of the UFL. Similarly, at the end of each time step in `shallow_water.py`, diagnostic fields, such as the divergence of a vector field, the Courant number, or the grid Peclet number field, can be computed. The routines used to compute these diagnostic fields are contained in `diagnostics.py`.

4 Verification and validation

The following subsections describe some of the key verification and validation test cases included in Firedrake-Fluids. These tests are executed using the Buildbot automated testing framework whenever a change is made to the software (Farrell et al., 2011) to ensure that any bugs introduced during the development of Firedrake-Fluids (or through the development of Firedrake itself and other dependencies such as PETSc) are detected and promptly resolved by the developers.

4.1 Convergence analysis

Since no general analytical solution to the shallow water equations exists, the method of manufactured solutions (MMS) (Roache, 2002) was used to perform a convergence analysis and verify the correctness of the model implementation. The first step of MMS involves inventing or manufacturing a function and modifying the original equation such that this manufactured function is the analytical solution of the modified equation. Substituting this function into the shallow water equations will generate a non-zero source term which can then be placed on the right-hand side, such that the manufactured/invented solution is now the analytical solution to this modified set of equations.

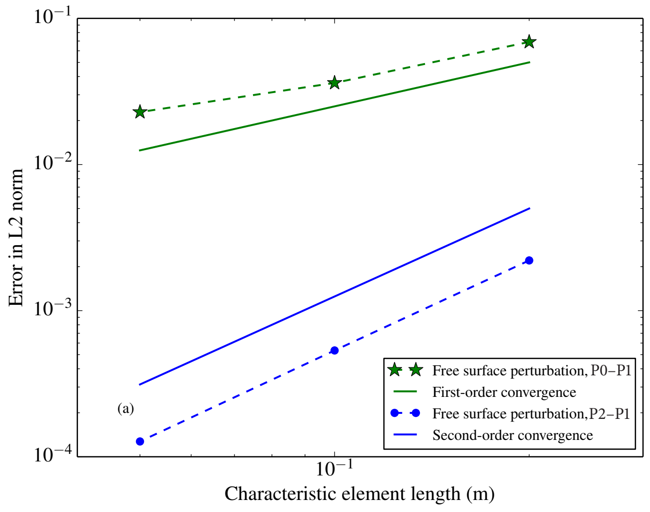
Table 1. Parameters used in the MMS test cases.

Parameter	Description	Value
C_D	Drag coefficient	0.0025
ν	Kinematic viscosity	$0.6\,\mathrm{m^2\,s^{-1}}$
H	Mean free surface height	20 m

A two-dimensional domain with dimensions $0 \le x \le 1$ m and $0 \le y \le 1$ m was used for the MMS simulations. Simulations were run with three different structured meshes with characteristic element lengths $\Delta x = 0.2$, 0.1 and 0.05 m, comprising 36, 121, and 441 vertices, respectively. The time steps were set to $\Delta t = 0.01$, 0.005, and 0.0025 s, respectively, to enforce a near-constant bound on the Courant number. A zero initial condition was used for both the velocity and free surface fields, and Dirichlet boundary conditions which agreed with the analytical/manufactured solutions for the velocity and free surface were enforced along all walls of the domain. All simulations were run until the steady-state conditions $||\boldsymbol{u}^{n+1} - \boldsymbol{u}^n||_2 \le 10^{-6}$ and $||h^{n+1} - h^n||_2 \le 10^{-6}$ were attained.

Both the P2–P1 and P0–P1 element pairs were considered. The manufactured solutions were $h = \sin(x)\sin(y)$ and $\boldsymbol{u} = [\cos(x)\sin(y), \sin(x^2) + \cos(y)]^{\mathrm{T}}$. The physical parameters, given in Table 1, were chosen arbitrarily and used across all the simulations.

The P2–P1 element pair was first considered to check the Galerkin method with continuous basis functions. As shown in Fig. 6a and b, this exhibited second-order spatial convergence for the free surface and approximately third-order convergence for the velocity field, giving confidence in the correctness of the implementation. Note that the discretisation error will be a combination of a first-order error (in Δt) from the backward Euler time-stepping scheme, and (in the case of a P2 velocity field) third-order error (in Δx) from the choice of spatial discretisation. The choices of Δt and Δx in the simulations presented here are such that the spatial term dominates. However, if the mesh is refined further (and the time step decreased accordingly to maintain the same bound on the Courant number), the third-order spatial term will decrease at a much faster rate than the first-order temporal term which may begin to dominate.



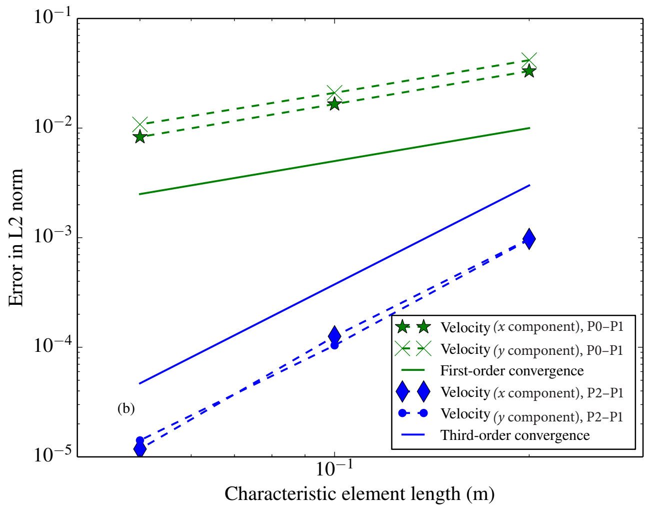


Figure 6. The orders of convergence for **(a)** the free surface field and **(b)** the velocity field, in the P2–P1 and P0–P1 MMS test cases.

In the case of P0–P1, both the velocity and free surface perturbation fields exhibited only first-order spatial convergence. Whilst second-order spatial convergence may have been expected for the free surface field because of the use of a P1 function space, the reduced order of convergence was likely the result of the coupling between the fields and the use of a first-order upwind scheme for the advection term at discontinuous element boundaries. The low-order scheme may have introduced additional, dominating error that polluted both solution fields via the coupled system, thereby keeping the overall spatial convergence at no higher than first-order.

All simulations were run in serial on a dual-core Intel Core i7-3537U processor with a clock speed of 2 GHz and at least 2 GB of available RAM. In the P2–P1 case, the runtimes were 3.8, 6.9, and 31.7 s, for the meshes with $\Delta x = 0.2$, 0.1, and 0.05 m, respectively. Note that these simulations were run with a warm cache such that the high-level UFL has already been compiled down to low-level C code; from a cold cache (i.e. including the code compilation time), the runtimes were 9.4, 12.5, and 37.7 s. In both the P2–P1 and P0–P1 cases, the simulations typically required 2–3 non-linear Newton iterations per time step, and the number of GMRES solver iterations taken per non-linear iteration varied between 12 and 17. However, in the P0–P1 case, the warm cache runtimes were significantly larger as a result of more time steps being required to reach steady state: 41.4, 85.8, and 177.8 s.

4.2 Dam failure

Dam failure (also known as dam break) problems are commonly used to test the performance of shallow water models. The presence of a discontinuity in the initial condition makes them particularly difficult to accurately solve. Both one-dimensional and two-dimensional results are presented.

The one-dimensional case considers a channel $0 \leq x \leq 2000$ m. A dam wall is located at $x = 1000$ m which holds back the water contained in the upstream reservoir. The water in the reservoir has a total depth of 10 m, whilst downstream the total water depth is set to 5 m. The water is initially at rest. At $t = 0$ the dam is instantaneously removed, thereby simulating its failure, allowing water to rush into the downstream section. Typical shock characteristics for the velocity and free surface perturbation fields were observed and compared well with the semi-analytical solutions of the corresponding one-dimensional Riemann problem shown in Fig. 7 at $t = 60$ s. Note that the simulation used an element length of $\Delta x = 5$ m and a time step of 0.25 s, as per the simulations of Liang et al. (2008) which consider the same scenario. The kinematic viscosity was set to $1\,\mathrm{m}^2\,\mathrm{s}^{-1}$, and the drag coefficient was set to zero.

The two-dimensional case considers a square domain with dimensions $0 \leq x \leq 200$ m and $0 \leq y \leq 200$ m. A 10 m thick dam is placed in the centre of the domain as shown in Fig. 8. In this scenario, only a partial failure of the dam is simulated; water rushes into the downstream area through a 75 m long breach in the dam wall. As before, the water is initially at rest. The upstream reservoir contains water with a total height of 10 m, whilst the downstream section contains water with a total height of 5 m. No-normal flow boundary conditions are applied along all walls (including those of the dam). Once again, the time step ($\Delta t = 0.2$ s) and the characteristic element length ($\Delta x = 5$ m) were the same as those chosen by Liang et al. (2008). The kinematic viscosity was set to $1\,\mathrm{m}^2\,\mathrm{s}^{-1}$, and the drag coefficient was set to zero.

The results at $t = 7.2$ s are shown in Fig. 9. The water that rushed into the downstream area formed a tidal bore wave which has started to spread out laterally, whilst a depression/rarefaction wave has started to propagate upstream. Furthermore, small vortices are visible where the flow has separated from the dam wall immediately downstream of the breach, resulting in a total free surface height of less than 5 m (the initial mean height downstream). These qualitative re-

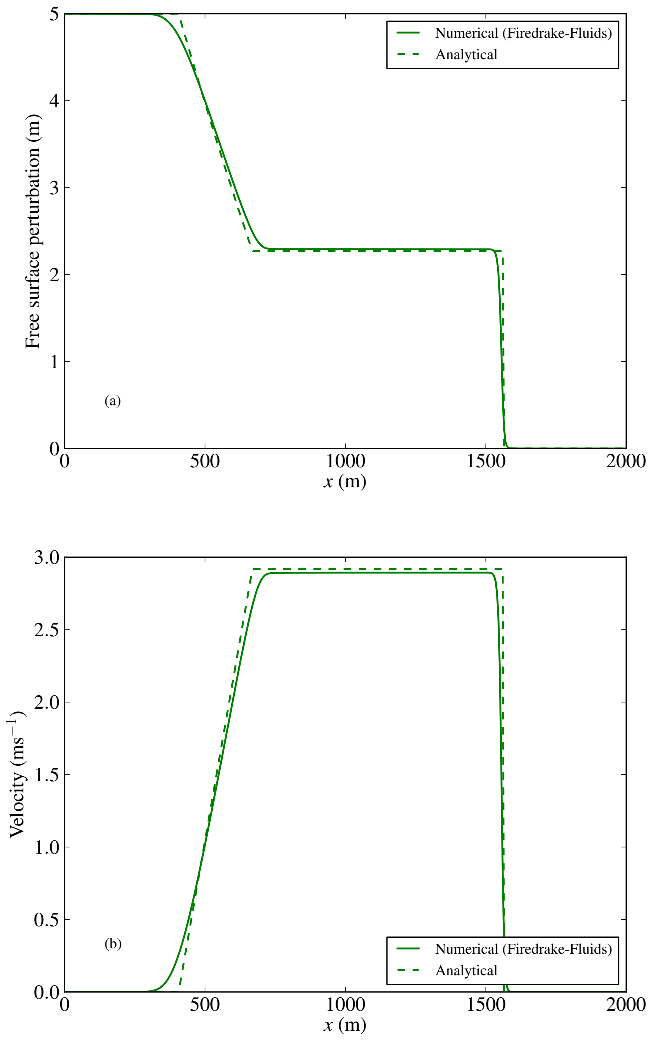


Figure 7. Numerical solutions of the 1-D dam failure problem. The semi-analytical solutions, found by solving a set of equations defined in the book by Trangenstein (2009), are also plotted.

sults closely agree with those from the numerical simulations by Liang et al. (2008) and Mingham and Causon (1998).

4.2.1 Solver performance

The performance of the iterative solver in combination with the fieldsplit preconditioner was investigated on a much larger system. The mesh was refined such that the characteristic element length was set to $\Delta x = 0.25$ m, resulting in 817 488 vertices. The time step Δt was also lowered to 0.01 s to maintain the same upper bound on the Courant number. The strong scaling of the iterative solver (GMRES, with the fieldsplit preconditioner) and the assembly of the system is shown in Fig. 10. All of these performance simulations were performed on ARCHER (a Cray XC30 supercomputer), comprising 12-core Intel Ivy Bridge processors running with

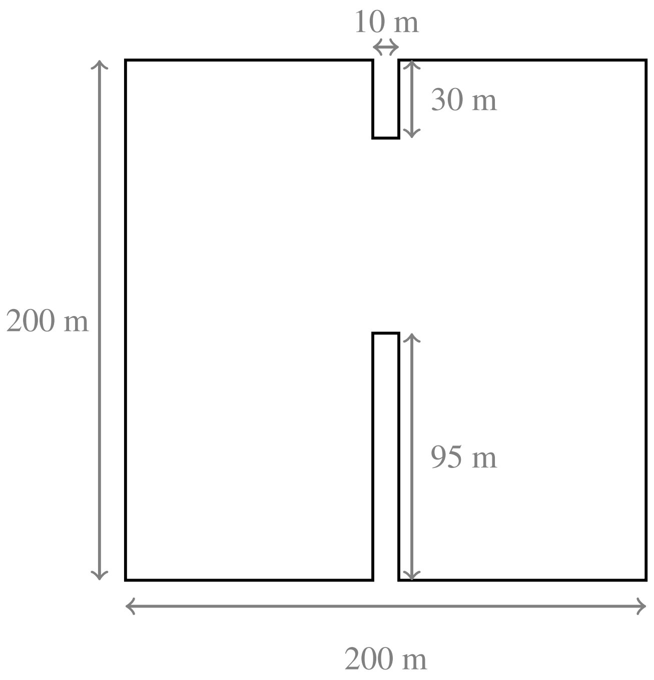


Figure 8. Dimensions of the domain for the 2-D dam failure problem. The dam (with a 75 m wide breach) is situated in the centre.

a clock speed of 2.7 GHz, with 32 GB of RAM available to each one.

The GMRES iterative method was also used in conjunction with the SOR preconditioner when computing the action of the matrices $\mathbf{A}^{-1}$ and $\mathbf{S}^{-1}$. This resulted in fewer outer iterations (typically 2 or 3) when solving the block-coupled system as a whole as a result of a better preconditioned system. On the other hand, ILU decomposition provided a relatively less accurate approximation to $\mathbf{A}^{-1}$ and $\mathbf{S}^{-1}$ (resulting in typically 10–30 outer iterations) but was faster than the GMRES with SOR runs, as shown in Fig. 10, despite the extra outer iterations. It is for this reason that ILU factorisation was used as the preconditioner of the sub-problems $\mathbf{A}^{-1}$ and $\mathbf{S}^{-1}$ throughout this paper. Note that smaller systems with $\Delta x = 0.5$ m and 1 m were also investigated; it was found that the number of solver iterations was near constant as the size of the system changed, regardless of the set-up of the fieldsplit preconditioner.

4.3 Tidal flow over a regular bed

The test case described by Bermudez and Vazquez (1994) considers tidal flow in a one-dimensional domain of length $L = 14\,000$ m. The mean water height (and hence the topography of the bed) is defined by

$$H(x) = 50.5 - \frac{40x}{L} - 10\sin\left[\pi\left(\frac{4x}{L} - \frac{1}{2}\right)\right]. \quad (18)$$

The initial conditions $h(x,0) = 0$ and $u(x,0) = 0$ are applied along with the following Dirichlet boundary conditions

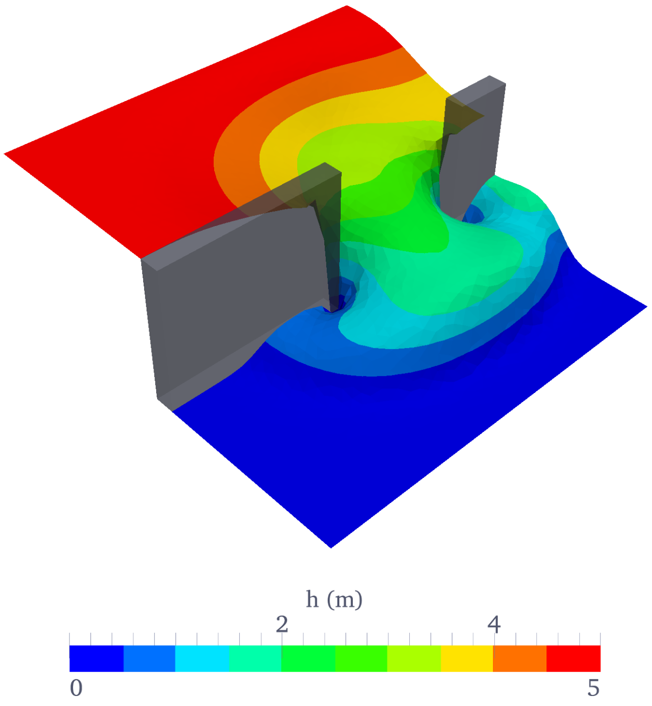


Figure 9. Free-surface perturbation h at time $t = 7.2$ s, from the 2-D dam failure simulation. The field has been warped in the z direction to emphasise the collapse of the water column.

for the free surface and velocity:

$$h(0,t) = 4 - 4\sin\left[\pi\left(\frac{4t}{86\,400} + \frac{1}{2}\right)\right], \tag{19}$$

to simulate an incoming sinusoidally varying tidal wave, and

$$u(L,t) = 0, \tag{20}$$

at the outflow boundary.

This simulation was performed with a mesh element length of $\Delta x = 14$ m. The time step Δt was set to 2.5 s and the simulation finished at $t = 9117.5$ s (the same time considered by Zhou, 2004). The kinematic viscosity was set to 1 m^2 s^{-1}, and the drag coefficient was set to zero. The results in Fig. 11 illustrate how the velocity of the flow increases in deeper regions of the body of water as expected. The numerical results also display good accuracy with the analytical solutions given by Bermudez and Vazquez (1994), thereby further validating the numerical model. The total runtime of the simulation was 26 min when run in serial on a dual-core Intel Core i7-3537U processor with a clock speed of 2 GHz and at least 2 GB of available RAM.

4.4 Tidal flow over an irregular bed

A second version of the tidal flow test case considered previously is one that involves an *irregular* bed topology, with sharp peaks and troughs which can be a challenge to represent accurately. This test case is described by Zhou (2004).

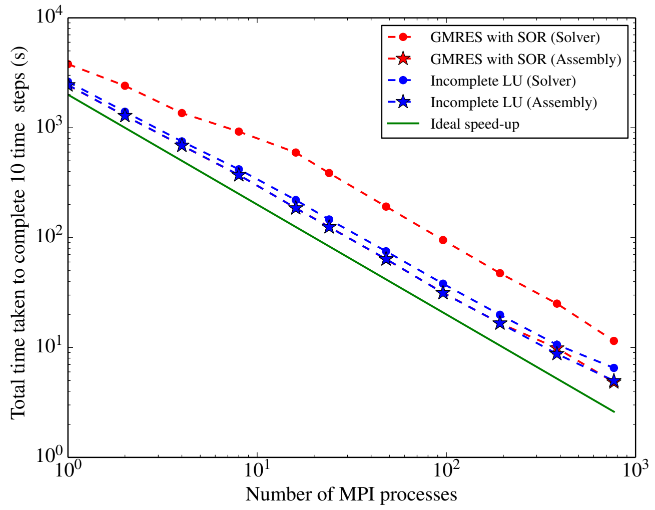


Figure 10. Strong scaling of the 2-D dam break simulation, with $\Delta x = 0.25$ m. The total runtime spent in the assembly and solver over 10 time steps is shown. The internal Firedrake/PyOP2 and PETSc timers were used to obtain the timing data. As expected, the time spent in assembly does not vary significantly between runs since it is independent of the difference in solver set-ups.

The test case considers a one-dimensional domain of length $L = 1500$ m. The irregular topography of the bed $B(x)$ is defined in Table 2, and the mean water height is given by $H(x) = 20 - B(x)$. The initial conditions $h(x,0) = -4$ and $u(x,0) = 0$ are applied along with the following Dirichlet boundary conditions for the free surface and velocity:

$$h(0,t) = -4\sin\left[\pi\left(\frac{4t}{86\,400} + \frac{1}{2}\right)\right], \tag{21}$$

$$u(L,t) = 0. \tag{22}$$

The element length $\Delta x = 7.5$ m and the time step $\Delta t = 0.3$ s, as per the set-up of Zhou (2004). The simulation was performed until $t = 10\,800$ s. All remaining components of the set-up were the same as the regular bed test case described in Sect. 4.3.

Figure 12 once again demonstrates a good match between the numerical results and the analytical solution, and demonstrates the robustness of the numerical model in accurately representing more rapidly varying areas of the solution. The total runtime of the simulation was 56.7 min when run in serial on a dual-core Intel Core i7-3537U processor with a clock speed of 2 GHz and at least 2 GB of available RAM.

4.5 Flow past a square cylinder

Simulations of laboratory-scale flow past solid objects are commonly used to validate turbulence models due to the vast number of available experimental data at high Reynolds numbers. In this work, the Smagorinsky LES model (Smagorinsky, 1963) in Firedrake-Fluids was employed to evaluate its ability to parameterise the effects of

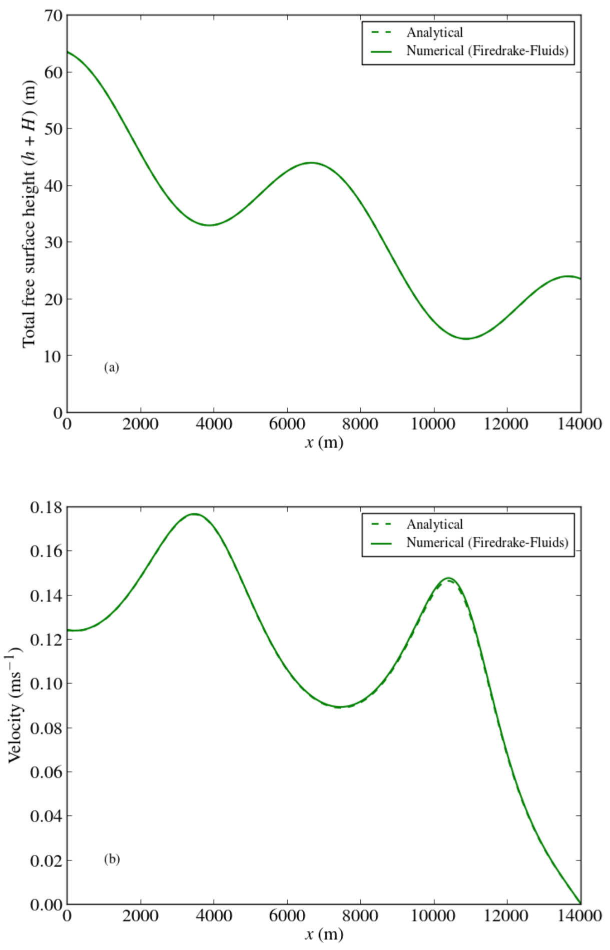


Figure 11. Numerical solutions from the tidal flow simulation over a regular bed, at $t = 9117.5$ s. The analytical solutions are given by Bermudez and Vazquez (1994) and almost completely overlap the numerical solutions. Note that the free surface plot **(a)** includes the mean free surface height, such that the y axis represents $h+H$.

turbulent flow past a square cylinder. The set-up used in the experiments by Lyn and Rodi (1994) and Lyn et al. (1995) (and the numerical simulations by Rodi et al., 1997) was considered.

The dimensions of the domain are given in terms of the width/length of the square $d = 0.04$ m in Fig. 13. An unstructured mesh with a characteristic element length $\Delta x = d/15$, generated with Gmsh (Geuzaine and Remacle, 2009), was used; this value of Δx is comparable to the minimum element lengths used in the numerical simulations presented in the paper by Rodi et al. (1997). The free surface mean height was set to $H = 4d$ (the depth of the experimental flow tank). The physical kinematic viscosity of the fluid was set to 10^{-6} m^2 s^{-1}, which corresponded to a Reynolds number of 21 400 when using d as the length scale. The Smagorinsky coefficient C_s in the Smagorinsky LES model (Smagorinsky, 1963) was set to 0.164, within the typical range of C_s values (Deardorff, 1971).

Table 2. Bed heights along the seabed from Zhou (2004).

x (m)	Bed height $B(x)$ (m)
0	0
50	0
100	2.5
150	5
250	5
300	3
350	5
400	5
425	7.5
435	8
450	9
475	9
500	9.1
505	9
530	9
550	6
565	5.5
575	5.5
600	5
650	4
700	3
750	3
800	2.3
820	2
900	1.2
950	0.4
1000	0
1500	0

Initially the velocity and free surface perturbation fields were set to zero. At the inlet, a constant velocity boundary condition of 0.535 ms^{-1} was enforced; the inflow was laminar and no turbulent eddies were seeded along the boundary. No-normal flow boundary conditions were applied along the side walls, whilst no-slip boundary conditions were applied along all walls of the square. At the outflow, a Flather boundary condition (Flather, 1976) (specifying an external velocity equal to that at the inlet, and a free surface perturbation of zero) was used to allow flow out of the domain whilst minimising reflections. A time step of $\Delta t = 5 \times 10^{-4}$ s was chosen, and the simulation was performed until $t = 15$ s.

The simulation was performed on ARCHER (a Cray XC30 supercomputer) using two 12-core Intel Ivy Bridge processors running with a clock speed of 2.7 GHz, with 32 GB of RAM available to each one. The total run time was 7.7 h.

Soon after the flow began to enter the domain through the inlet, boundary layers began to form around the sides of the square where the transition to turbulence took place. A strong recirculating region formed immediately behind the square, followed by continuous turbulent vortex shedding which commenced after approximately 4 s of simula-

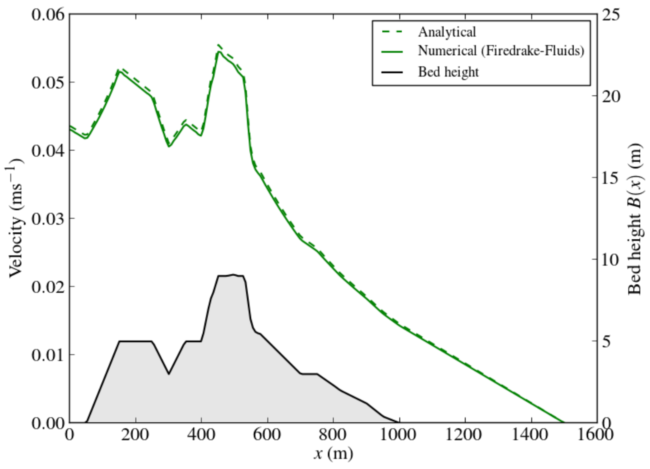


Figure 12. Numerical solutions from the tidal flow simulation with an irregular bed topography. The analytical solutions (Zhou, 2004; Bermudez and Vazquez, 1994) agree very well with the numerical solutions from Firedrake-Fluids.

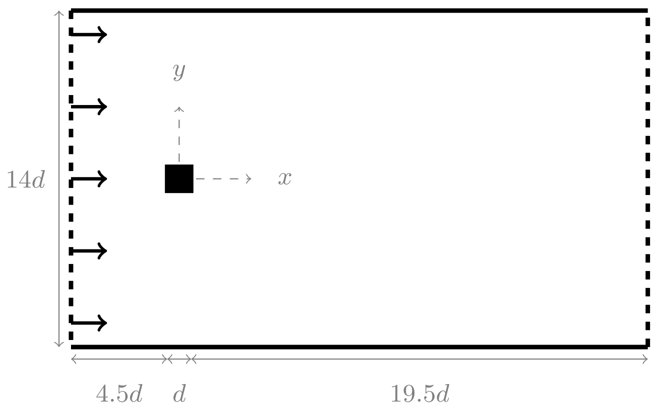


Figure 13. The dimensions of the two-dimensional domain containing a square cylinder (filled black) of length/width d. The incoming flow is from the left boundary, as denoted by the black arrows.

tion time. The vortex street is clearly visible in Fig. 14 which shows the x component of the velocity field at $t = 10$ s.

The stream-wise velocity along the centreline, time-averaged over a period of 15 s from the start of the simulation, was compared with the experimental data presented by Lyn et al. (1995) and Rodi et al. (1997); the results in Fig. 15 show a good match with the experimental data behind the square cylinder in the recirculating region where turbulent vortex shedding occurs, thereby illustrating the benefits of using the Smagorinsky LES model to accurately capture the turbulent flow characteristics. However, the wake recovery region was poorly represented; the unfortunate lack of accuracy in this region has also been observed in other numerical models (Rodi et al., 1997), and additional parameterisations and the full three-dimensionality of the problem may need to be considered to properly represent the wake.

5 Road map

The long-term aim is to extend Firedrake-Fluids into a suite of numerical models which encompass a much wider range of flow types, as well as additional equation sets (e.g. the full Navier–Stokes equations) and constitutive equations (e.g. for describing Darcy's law in porous media). Essentially, Firedrake-Fluids seeks to facilitate a complete re-engineering of the Fluidity CFD code, whilst maintaining the mature modelling functionality that Fluidity offers. In addition to the potential for portability of the low-level code across different back ends, such as the Intel C compiler and CUDA, it is hoped that Firedrake will also enable the portability of the code's performance. This has yet to be demonstrated on large-scale problems, and will therefore be one of the main focusses of this work in the future.

One of the first application areas that Firedrake-Fluids will consider, using the shallow water model described in this paper, is flow around tidal turbines. This will contribute to an on-going effort towards understanding the potential of renewable energy systems. The multi-scale nature of the application will necessitate the use of high-performance computing, and Firedrake's ability to target code towards more modern hardware architectures such as GPU clusters will be utilised. Regarding the application area itself, the integration of adjoint optimisation models is of particular related interest. For example, recent progress in the optimisation of the layout of a tidal turbine farm using the FEniCS automated solution framework has proven to be a successful technique for maximising the theoretical amount of generated power (Funke et al., 2014). The DOLFIN-adjoint library (Farrell et al., 2013) was used for this purpose. Although FEniCS and Firedrake both expect UFL statements as input, not all of the UFL interfaces are compatible with each other at present; a similar adjoint library for Firedrake (Firedrake-adjoint) is therefore under development by the authors of DOLFIN-adjoint, and its use in the shallow water model is one of the shorter-term goals of the Firedrake-Fluids project. The issue of compatibility is being addressed by the developers of Firedrake.

Realistic tidal and atmospheric modelling simulations will require boundary values to be read in from forcing files. Popular formats include NetCDF and ERA-40/GRIB, for which robust data readers will be required. Therefore, another short-term item on the road map is the evaluation and integration of existing readers into the Firedrake-Fluids framework (or their development in-house, should no suitable reader exist).

Further to the existing Smagorinsky LES turbulence model (Smagorinsky, 1963), the road map features support for additional turbulence parameterisations including RANS-type models, such as those considered by Mortensen et al. (2011) for the FEniCS framework. Alternative discretisation schemes, including control volume methods which have desirable boundedness and conservativeness properties (Wilson, 2009), and high-order slope limiters for the existing dis-

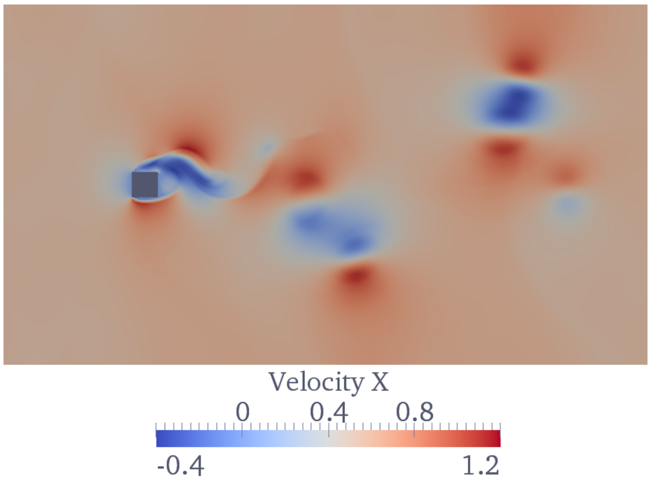


Figure 14. Visualisation of the x component of the velocity field, from the simulation of flow past a square at $t = 10$ s.

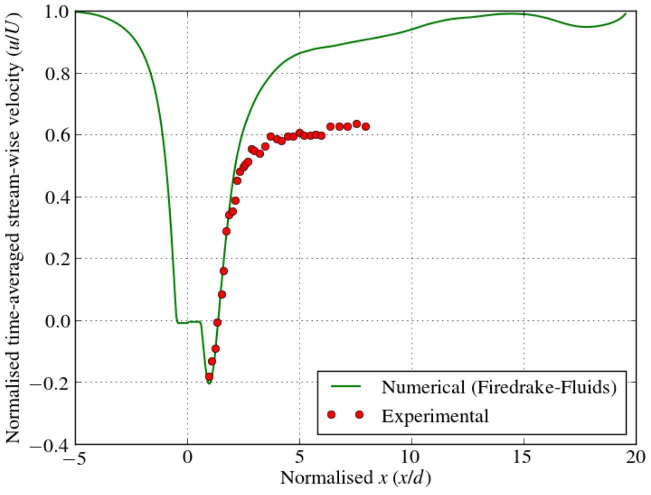


Figure 15. Time-averaged stream-wise velocity along the centreline from the simulation of flow past a square. Note that the velocity has been normalised by the inlet velocity $U = 0.535\,\mathrm{ms^{-1}}$.

continuous Galerkin method will also be implemented. It is expected that a large proportion of this work will need to be undertaken within the Firedrake and PyOP2 frameworks, in addition to Firedrake-Fluids, in order to correctly describe the mesh topology (including that of the dual mesh in the case of control volume methods).

6 Conclusions

This model description paper has introduced a new open-source finite element model, Firedrake-Fluids, for the simulation of shallow water flows. The model is written in the high-level, near-mathematical UFL and uses the Firedrake framework (coupled with the PyOP2 library) to automate the solution process. Furthermore, the Firedrake library provides the potential for porting the code across to different hardware back ends, although this has not been demonstrated here and will be a consideration of future work. The automated solution approach allows the focus to be on the equations that are solved and the numerical results, and removes the requirement for model developers to be experts in parallel programming and software engineering. Furthermore, the high-level specification of the problem facilitates better maintainability of the Firedrake-Fluids code base; in comparison with the shallow water model in the Fluidity CFD code, which features static hand-written Fortran, the Firedrake-Fluids source code is considerably shorter and more intuitive. This is a result of the near-mathematical notation used, and the fact that code generation and assembly are handled by the external Firedrake and PyOP2 libraries. Firedrake-Fluids uses approximately 400 lines (excluding comments and blank lines), compared to many thousands to perform the same task in Fluidity. Note that the 400 lines include code to obtain user settings, initial conditions, etc., from the simulation configuration file, and to make the model as generic as possible; if the model were to be written for a specific set-up, the number of lines could potentially be further minimised to just a few dozen. It should be noted that this benefit is not unique to the Firedrake-Fluids model nor to UFL in general, since it is also possible to write models with a relatively small amount of code with other packages such as OpenFOAM (OpenFOAM, 2014), deal.II (Bangerth et al., 2007), Dune (Dedner et al., 2010), and FreeFem++ (Hecht, 2012).

At runtime, the high-level model specification defined in Firedrake-Fluids is converted by Firedrake (and the PyOP2 framework) into optimised, low-level C code. This is then compiled with a back-end compiler appropriate for the target architecture; however, this work has only considered the GNU gcc compiler for CPUs, since it is not yet possible to assemble and run the non-linear problems detailed in this paper on GPUs. As new high-performance architectures are introduced in the future, only the PyOP2 layer which deals with code targeting needs to be modified; model developers are not burdened with the task of specialising the model code itself, which is presently a common problem even in modern finite element models.

Several verification and validation test cases were performed to ensure the correctness of Firedrake-Fluids and its ability to accurately simulate physical problems. These included a convergence analysis with different finite element pairs, a simulation of dam breaching, and tidal flow dynamics over different seabed topologies. Overall, the numerical results were highly satisfactory and displayed good agreement with analytical solutions, experimental data, and observations.

Code availability

Firedrake-Fluids is an open-source software package that has been released under the GNU General Public License (Version 3). The code base is hosted by GitHub in a public repository and can be obtained at the following URL: https://github.com/firedrakeproject/firedrake-fluids. The particular version of Firedrake-Fluids considered in this paper (version 0.1) is available from the releases page.

Acknowledgements. The authors would like to acknowledge funding from Imperial College London and an ARCHER eCSE grant (eCSE03-7). The authors also acknowledge the use of the Imperial College High Performance Computing Service, and the UK National Supercomputing Service (ARCHER). Thoughtful reviews from Marie Rognes and Andreas Dedner were gratefully received.

Edited by: H. Weller

References

Alnæs, M. S., Logg, A., Ølgaard, K. B., Rognes, M. E., and Wells, G. N.: Unified Form Language: A domain-specific language for weak formulations of partial differential equations, ACM Trans. Math. Softw., 40, 9, doi:10.1145/2566630, 2014.

Arnold, D. N.: An Interior Penalty Finite Element Method with Discontinuous Elements, SIAM J. Num. Analysis, 19, 742–760, doi:10.1137/0719052, 1982.

Balay, S., Abhyankar, S., Adams, M., Brown, J., Brune, P., Buschelman, K., Eijkhout, V., Gropp, W., Kaushik, D., Knepley, M., Curfman McInnes, L., Rupp, K., Smith, B., and Zhang, H.: PETSc Users Manual, Tech. Rep. Revision 3.5, Argonne National Laboratory, 2014.

Bangerth, W., Hartmann, R., and Kanschat, G.: deal.II–A general-purpose object-oriented finite element library, ACM Trans. Math. Softw., 33, 24, doi:10.1145/1268776.1268779, 2007.

Bassi, F. and Rebay, S.: A High-Order Accurate Discontinuous Finite Element Method for the Numerical Solution of the Compressible Navier-Stokes Equations, J. Comput. Phys., 131, 267–279, doi:10.1006/jcph.1996.5572, 1997.

Bermudez, A. and Vazquez, M. E.: Upwind methods for hyperbolic conservation laws with source terms, Comput. Fluids, 23, 1049–1071, doi:10.1016/0045-7930(94)90004-3, 1994.

Brown, J., Knepley, M. G., May, D. A., McInnes, L. C., and Smith, B. F.: Composable Linear Solvers for Multiphysics, in: Proceeedings of the 11th International Symposium on Parallel and Distributed Computing (ISPDC 2012), 55–62, doi:10.1109/ISPDC.2012.16, 2012.

Capart, H. and Young, D. L.: Formation of a jump by the dam-break wave over a granular bed, J. Fluid Mech., 372, 165–187, doi:10.1017/S0022112098002250, 1998.

Deardorff, J.: A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers, J. Fluid Mech., 41, 453–480, doi:10.1017/S0022112070000691, 1970.

Deardorff, J. W.: On the magnitude of the subgrid scale eddy coefficient, J. Comput. Phys., 7, 120–133, doi:10.1016/0021-9991(71)90053-2, 1971.

Dedner, A., Klöfkorn, R., Nolte, M., and Ohlberger, M.: A generic interface for parallel and adaptive discretization schemes: Abstraction principles and the DUNE-FEM module, Computing, 90, 165–196, doi:10.1007/s00607-010-0110-3, 2010.

Divett, T., Vennell, R., and Stevens, C.: Optimization of multiple turbine arrays in a channel with tidally reversing flow by numerical modelling with adaptive mesh, Philos. Trans. Roy. Soc. A, 371, 1471–2962, doi:10.1098/rsta.2012.0251, 2013.

Elman, H., Howle, V. E., Shadid, J., Shuttleworth, R., and Tuminaro, R.: A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible Navier-Stokes equations, J. Computat. Phys., 227, 1790–1808, doi:10.1098/rsta.2012.0251, 2008.

Elman, H. C., Silvester, D. J., and Wathen, A. J.: Finite Elements and Fast Iterative Solvers: with applications in incompressible fluid dynamics, Oxford University Press, 2005.

Farrell, P. E., Piggott, M. D., Gorman, G. J., Ham, D. A., Wilson, C. R., and Bond, T. M.: Automated continuous verification for numerical simulation, Geosci. Model Dev., 4, 435–449, doi:10.5194/gmd-4-435-2011, 2011.

Farrell, P. E., Ham, D. A., Funke, S. W., and Rognes, M. E.: Automated derivation of the adjoint of high-level transient finite element programs, SIAM J. Scientific Comput., 35, C369–C393, doi:10.1137/120873558, 2013.

Flather, R. A.: A tidal model of the northwest European continental shelf, Memoires de la Société Royale des Sciences de Liège, 10, 141–164, 1976.

Funke, S. W. and Farrell, P. E.:A framework for automated PDE-constrained optimisation, arXiv:1302.3894, submitted, 2013.

Funke, S. W., Farrell, P. E., and Piggott, M. D.: Tidal turbine array optimisation using the adjoint approach, Renewable Energy, 63, 658–673, doi:10.1016/j.renene.2013.09.031, 2014.

Geuzaine, C. and Remacle, J.-F.: Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities, Int. J. Num. Methods Eng., 79, 1309–1331, doi:10.1002/nme.2579, 2009.

Ham, D. A., Farrell, P. E., Gorman, G. J., Maddison, J. R., Wilson, C. R., Kramer, S. C., Shipton, J., Collins, G. S., Cotter, C. J., and Piggott, M. D.: Spud 1.0: generalising and automating the user interfaces of scientific computer models, Geosci. Model Dev., 2, 33–42, doi:10.5194/gmd-2-33-2009, 2009.

Hecht, F.: New development in FreeFem++, J. Num. Math., 20, 251–265, doi:10.1515/jnum-2012-0013, 2012.

Hill, J., Collins, G. S., Avdis, A., Kramer, S. C., and Piggott, M. D.: How does multiscale modelling and inclusion of realistic palaeobathymetry affect numerical simulation of the Storegga Slide tsunami?, Ocean Model., 83, 11–25, doi:10.1016/j.ocemod.2014.08.007, 2014.

Katz, R. F., Knepley, M. G., Smith, B., Spiegelman, M., and Coon, E. T.: Numerical simulation of geodynamic processes with the Portable Extensible Toolkit for Scientific Computation, Phys. Earth Planet. Int., 163, 52–68, doi:10.1016/j.pepi.2007.04.016, 2007.

Kirby, R. C.: Algorithm 839: FIAT, a New Paradigm for Computing Finite Element Basis Functions, ACM Trans. Math. Softw., 30, 502–516, doi:10.1145/1039813.1039820, 2004.

Kirby, R. C. and Logg, A.: A compiler for variational forms, ACM Trans. Math. Softw., 32, 417–444, doi:10.1145/1163641.1163644, 2006.

Kramer, S. C., Piggott, M. D., Hill, J., Kregting, L., Pritchard, D., and Elsaesser, B.: The modelling of tidal turbine farms using multi-scale, unstructured mesh models, in: Proceedings of the 2nd International Conference on Environmental Interactions of Marine Renewable Energy Technologies, (EIMR2014), Stornoway, Scotland, 2014.

Liang, S.-J., Tang, J.-H., and Wu, M.-S.: Solution of shallow-water equations using least-squares finite-element method, Acta Mechanica Sinica, 24, 523–532, doi:10.1007/s10409-008-0151-4, 2008.

Lloyd, P. M. and Stansby, P. K.: Shallow-Water Flow around Model Conical Islands of Small Side Slope. II: Submerged, J. Hydraul. Eng., 123, 1068–1077, doi:10.1061/(ASCE)0733-9429(1997)123:12(1068), 1997.

Logg, A. and Wells, G. N.: DOLFIN: Automated finite element computing, ACM Trans. Math. Softw., 37, 20, doi:10.1145/1731022.1731030, 2010.

Logg, A., Mardal, K.-A., Wells, G. N., Alnæs, M. S., Clark, S. R., Davidson, D. B., Vilela de Abreu, R., Degirmenci, C., Haga, J. B., Hake, J., Hoffman, J., Jansson, J., Jansson, N., Johnson, C., Kirby, R. C., Knepley, M. G., Langtangen, H. P., Lezar, E., Linge, S., Logg, A., Lopes, N. D., Løvgren, A. E., Mardal, K.-A., Mortensen, M., Narayanan, H., Nazarov, M., Nikbakht, M., Pereira, P. J. S., Ring, J., Rognes, M. E., Schroll, H. J., Scott, L. R., Selim, K., Støle-Hentschel, S., Terrel, A. R., Trabucho, L., Valen-Sendstad, K., Vynnytska, L., Wells, G. N., Wilbers, I. M., and Ølgaard, K. B.: Automated Solution of Differential Equations by the Finite Element Method, Springer, doi:10.1007/978-3-642-23099-8, 2012.

Luporini, F., Varbanescu, A. L., Rathgeber, F., Bercea, G.-T., Ramanujam, J., Ham, D. A., and Kelly, P. H. J.: Cross-Loop Optimization of Arithmetic Intensity for Finite Element Local Assembly, ACM Trans. Archi. Code Optimization, 11, 57, doi:10.1145/2687415, 2015.

Lyn, D. A. and Rodi, W.: The flapping shear layer formed by flow separation from the forward corner of a square cylinder, J. Fluid Mech., 267, 353–376, doi:10.1017/S0022112094001217, 1994.

Lyn, D. A., Einav, S., Rodi, W., and Park, J.-H.: A laser-Doppler velocimetry study of ensemble-averaged characteristics of the turbulent near wake of a square cylinder, J. Fluid Mech., 304, 285–319, doi:10.1017/S0022112095004435, 1995.

Maddison, J. R. and Farrell, P. E.: Rapid development and adjoining of transient finite element models, Comput. Meth. Appl. Mech. Eng., 276, 95–121, doi:10.1016/j.cma.2014.03.010, 2014.

Markall, G. R., Rathgeber, F., Mitchell, L., Loriant, N., Bertolli, C., Ham, D. A., and Kelly, P. H.: Performance-Portable Finite Element Assembly Using PyOP2 and FEniCS, in: 28th International Supercomputing Conference, ISC, Proceedings, vol. 7905 of Lecture Notes in Computer Science, 279–289, Springer, 2013.

Martin-Short, R., Hill, J., Kramer, S. C., Avdis, A., Allison, P. A., and Piggott, M. D.: Tidal resource extraction in the Pentland Firth, UK: potential impacts on flow regime and sediment transport in the Inner Sound of Stroma, Renewable Energy, 76, 596–607, doi:10.1016/j.renene.2014.11.079, 2015.

Mingham, C. G. and Causon, D. M.: High-Resolution Finite-Volume Method for Shallow Water Flows, J. Hydraul. Eng., 124, 605–614, doi:10.1061/(ASCE)0733-9429(1998)124:6(605), 1998.

Mortensen, M., Langtangen, H. P., and Wells, G. N.: A FEniCS-based programming framework for modeling turbulent flow by the Reynolds-averaged Navier-Stokes equations, Adv. Water Resour., 34, 1082–1101, doi:10.1016/j.advwatres.2011.02.013, 2011.

Ølgaard, K. B. and Wells, G. N.: Optimisations for quadrature representations of finite element tensors through automated code generation, ACM Trans. Math. Softw., 37, 8, doi:10.1145/1644001.1644009, 2010.

OpenFOAM: OpenFOAM User Guide, Version 2.3.1, 2014.

Piggott, M. D., Gorman, G. J., Pain, C. C., Allison, P. A., Candy, A. S., Martin, B. T., and Wells, M. R.: A new computational framework for multi-scale ocean modelling based on adapting unstructured meshes, Int. J. Num. Methods Fluids, 56, 1003–1015, doi:10.1002/fld.1663, 2008.

Rathgeber, F.: Productive and Efficient Computational Science Through Domain-specific Abstractions, Ph.D. thesis, Imperial College London, 2014.

Rathgeber, F., Markall, G. R., Mitchell, L., Loriant, N., Ham, D. A., Bertolli, C., and Kelly, P. H.: PyOP2: A High-Level Framework for Performance-Portable Simulations on Unstructured Meshes, in: High Performance Computing, Networking Storage and Analysis, SC Companion, IEEE Computer Society, 1116–1123, 2012.

Rathgeber, F., Ham, D. A., Mitchell, L., Lange, M., Luporini, F., McRae, A. T. T., Bercea, G.-T., Markall, G. R., and Kelly, P. H. J.: Firedrake: automating the finite element method by composing abstractions, ACM Trans. Math. Softw., http://arxiv.org/abs/1501.01809, submitted, 2015.

Roache, P. J.: Code Verification by the Method of Manufactured Solutions, J. Fluids Eng., 124, 4–10, doi:10.1115/1.1436090, 2002.

Rodi, W., Ferziger, J. H., Breuer, M., and Porquié, M.: Status of Large Eddy Simulation: Results of a Workshop, Trans. ASME, 119, 248–262, 1997.

Rognes, M. E., Ham, D. A., Cotter, C. J., and McRae, A. T. T.: Automating the solution of PDEs on the sphere and other manifolds in FEniCS 1.2, Geosci. Model Dev., 6, 2099–2119, doi:10.5194/gmd-6-2099-2013, 2013.

Saad, Y. and Schultz, M. H.: GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Scient. Stat. Compu., 7, 856–869, doi:10.1137/0907058, 1986.

Smagorinsky, J.: General Circulation Experiments with the Primitive Equations, Mon. Weather Rev., 91, 99–164, doi:10.1175/1520-0493(1963)091<0099:GCEWTP>2.3.CO;2, 1963.

Trangenstein, J. A.: Numerical Solution of Hyperbolic Partial Differential Equations, Cambridge University Press, 2009.

Wilson, C.: Modelling Multiple-Material Flows on Adaptive Unstructured Meshes, Ph.D. thesis, Imperial College London, 2009.

Zhou, J. G.: Lattice Boltzmann Methods for Shallow Water Flows, Springer, 2004.