A Modern And Unified C++ Implementation Of Finite Element And Spectral Elements Methods In 1D, 2D And 3D

Christophe Prud'homme

EPFL SB IACS CMCS MA B2 534 (Bâtiment MA) Station 8 CH-1015 Lausanne
christophe.prudhomme@epfl.ch

Abstract. We present in this paper a modern and unified c++ implementation of finite and spectral element methods in 1, 2 and 3D that provides a mathematical kernel for a number of complex multiphysics and multiscale simulations, such as fluid structure interaction or mass transport, in the context of haemodynamics. The main result consists of a library of components that provide domain-specific abstractions, in our case numerical analysis, using generic, meta-programming and object-oriented paradigms. At the top of these components sits a so-called domain-specific language embedded in C++ that allows to formulate variational problems in terms close to the mathematics. This has a number of advantages among which using implicitly the properties of the C++ compilers such as their optimization framework or not having to use external languages such as Python or home-made to assemble and define the end-user application.

In [4], we presented a preliminary version of a domain specific language embedded in *C++* that can be used in various contexts such as numerical projection onto a functional space, numerical integration, variational formulations and automatic differentiation. In this article, only the language was described along with some short description of lower level mathematical concepts such as approximation spaces, basis functions or operators. In the present paper, we would like to describe a modern and unified *C++* implementation of the finite and spectral element methods in 1, 2 and 3D. In particular, we shall focus on the mathematical framework at the core of our implementation and show that we reproduce it in *C++* closely which allows for greater flexibility in the way we define the problem at hand and not at the expense of performance.

1 Key Features

1.1 General Features

A number of requirements have been setup on the library such as *(i)* numerical type independence or more precisely the support of an extensible set of numerical types — standard *C++* types, higher and arbitrary precision types — with a consistent set of mathematical operations, *(ii)* development of abstractions close to the mathematical abstractions in *C++* such that we don't need to rely on external languages or GUIs to define the end-user application and we can use the compiler capabilities for optimizing, debugging, profiling or tuning the code on a particular platform and *(iii)* support for seamless parallel and distributed computing.

In many aspects, our framework is an active software library as described in [5,6]

1.2 Seamless Polynomial Basis Construction

One of the key feature is the somewhat seamless construction of polynomial basis functions, following the new paradigm proposed in [2,3], defined over a subset of $\mathbb{R}^d$, typically simplices and simplex products, and with values in $\mathbb{R}$, $\mathbb{R}^d$ or $\mathbb{R}^{d\times d}, d = 1, 2, 3$. The basic ideas consist in *(i)* expressing the polynomials and polynomial sets in an L_2 orthonormal (hierarchical) basis defined on a reference convex, see [1], *(ii)* the ability to define sets of functionals or constraints and *(iii)* the ability to define degree of freedoms either for continuous(cG) or discontinuous(dG) expansions.
This library component allows to define and use a wide range of approximation spaces and methods for partial differential equations very easily and in a unified way.

1.3 A Language for Variational Formulations Embedded in *C++*

Another key feature is the ability to express variational formulations directly in *C++* using an embedded language close the mathematics. To illustrate what we want to achieve with the domain-specific embedded language(DSEL), here is a simple example with a mathematical formulation of a bilinear form, see (1), and its *C++* counterpart using the proposed DSEL, see listing 1.1.

$$\begin{aligned} a : X_h \times X_h &\to \mathbb{R} \\ (u, v) &\to \int_\Omega \nabla u \cdot \nabla v + uv \end{aligned} \tag{1}$$

More complete examples are available in [4]. In the proposed paper, we shall supply an update on the recent work on the DSEL.

2 Applications

This mathematical kernel is now being used to define higher level operators and solvers for multiphysics strategies that are being proposed and developed within the European project HaeMOdel (`http://mox.polimi.it/haemodel`) and its associated software library LifeV (`http://www.lifev.org`). Some preliminary mass transport simulations coupling Navier-Stokes, Darcy and Convection-Diffusion-Reaction equations will be presented.

Listing 1.1: Variational Formulation in *C++*; the `t` extension of `gradt` and `idt` identify the trial functions

```
// a mesh of $\Omega \subset \mathbb{R}^d, d = 1,2,3$
Mesh mesh;
// Finite element scalar space $\mathbb{P}_K, K = 1,2,3,\dots$
Space<Mesh,Lagrange<d,K> > Xh(mesh);
// two elements of the Space Xh
Space<Mesh,Lagrange<d,K> >::element_type u(Xh), v(Xh);
// A matrix in CSR format
csr_matrix_type M;
// bilinear form with M as its matrix representation
// with integration over all elements of the mesh
// and a method for exact integration of polynomials of
// degree $\leq K$ (IM_PK<d,K>)
BilinearForm<Xh,Xh,csr_matrix_type> a(Xh,Xh,M);
a = integrate( elements(mesh),
               dot(gradt(u),grad(v))+idt(u)*id(v),
               IM_PK<N,K>() );
```

References

1. George Em Karniadakis and Spencer J. Sherwin. *Spectral/hp element methods for CFD*. Numerical Mathematics and Scientific Computation. Oxford University Press, New York, 1999.
2. Robert C. Kirby. Algorithm 839: Fiat, a new paradigm for computing finite element basis functions. *ACM Trans. Math. Softw.*, 30(4):502–516, 2004.
3. Robert C. Kirby. Optimizing fiat with the level 3 blas. *to appear in ACM Trans. Math. Software*, 2005.
4. Christophe Prud'homme. A domain specific embedded language in c++ for automatic differentiation, projection, integration and variational formulations. *Scientific Programming*, 2005. Accepted.
5. Todd Veldhuizen. Software libraries and the limits of reuse: Entropy, kolmogorov complexity, and zipf's law. `http://www.livejournal.com/users/endoprogramming/1257.html`, August 2005.
6. Todd L. Veldhuizen and Dennis Gannon. Active libraries: Rethinking the roles of compilers and libraries. In *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing (OO'98)*. SIAM Press, 1998.