Life, a modern and unified C++ implementation of finite-element and spectral-element methods in 1D, 2D and 3D

Christophe Prud'homme[1,*]

[1] Université Joseph Fourier Grenoble 1, BP 53 38041 Grenoble Cedex 9

We present a brief overview of Life, a unified framework for finite element and spectral element methods in 1D, 2D and 3D in C++. First, we describe some basic principles, then we focus on the two cornerstones of the library, the polynomial library and FEEL++, a C++ embedded language designed for partial differential equations. The applications range is potentially quite large; at the moment Life is essentially used for the developement of new numerical methods and the exhaustive comparisons between standard ones e.g. high order methods, stabilisation methods.

1 Introduction

We present a brief overview of Life, a unified framework for finite element and spectral element methods in 1D, 2D and 3D in C++. The objectives of this framework are quite ambitious and could be expressed in various ways such as *(i)* the creation of a versatile mathematical kernel easily solving problems using different techniques thus allowing testing and comparing numerical methods, e.g. continuous Galerkin (cG) versus discontinuous Galerkin (dG) , *(ii)* the creation of a *small* and *manageable* library which shall nevertheless encompass a wide range of numerical methods and techniques, *(iii)* build mathematical software that follows closely the mathematical abstractions and language associated with partial differential equations (PDE) — which is often not the case, for example the design could be physics oriented, — and *(iv)* the creation of a library entirely in C++ allowing to create complex multi-physics applications such as fluid-structure interaction or mass transport in haemodynamics — the rationale being that these applications are computing intensive and the use of an interpreted language such as Python would not be satisfying though in many simpler cases that would simplify and accelerate the development.

2 Basic principles

First, the syntax, the semantics and the pragmatics of the library are very close to the mathematics; the design and implementation are mathematics oriented. C++ supports very well multiple paradigms design and offers a wide range of solutions for a given problem. Generic programming, OO programming, meta-programming are such paradigms and they are definitely very useful when dealing with mathematical abstractions. For example, the following code uses all these paradigms:

```
integrate( elements(mesh), im, trace(hessv(u)*trans(hessv(u)))).evaluate();
```

It integrates the scalar function $\nabla^2 u : \nabla^2 u$ over the elements of the mesh where `hessv(u)` $\equiv \nabla^2 u$ denotes the hessian of the scalar function u, `im` is the numerical integration method to use and `elements(mesh)` is a pair of iterators over the set of elements.

Second, regarding linear algebra, the library delegates to well maintained third party libraries through unified interfaces using the Facade design pattern, see [1]. At the moment, the library supports gmm [2], Boost.ublas, PETSc [3] and Trilinos [4]. Life parallel framework is built on top of the PETSc and Trilinos ones depending on which one is used as solver back-end. One of the main visible features of our parallel framework is that it provides powerful seamless iterators over the parallel mesh. To illustrate, the following code

```
// mesh is a parallel mesh partitioned using e.g. parmetis
integrate( elements(mesh), im, trace(gradv(u)*trans(gradv(u)))+
                               trans(curlv(v))*curlv(v)  ).evaluate();
```

will integrate the scalar function $\nabla \mathbf{u} : \nabla \mathbf{u} + \nabla \times \mathbf{v} \cdot \nabla \times \mathbf{v}$ over all elements belonging to the current process where `gradv(u)` $\equiv \nabla \mathbf{u}$ denotes the gradient of the vectorial function $\mathbf{u}$ and `curlv(u)` $\equiv \nabla \times \mathbf{u}$ its curl, while

```
integrate( interprocessfaces(mesh), im, trace(hessv(u))+gradv(u)*N()).evaluate();
integrate( inprocessfaces(mesh), im, trace(hessv(u))+gradv(u)*N()).evaluate();
integrate( boundaryprocessfaces(mesh), im, trace(hessv(u))+gradv(u)*N()).evaluate();
```

* Corresponding author E-mail: christophe.prudhomme@ujf-grenoble.fr, Phone: +33 4 76 63 54 97 Fax: +33 4 76 63 12 63

will integrate the scalar function $\Delta u + \nabla u \cdot N$ over, respectively, the faces between the current process and its neighbors, the faces belonging to the current process and the faces on the boundary of the current process (physical boundary and process boundary). We refer the reader to [5, 6] for a more thorough overview of Life and description of the language.

3 The polynomial library and embedded language

The polynomial library and the embedded language are the two cornerstones of Life. The polynomial library is composed of various bricks: *(i)* the geometrical entities or convexes *(ii)* the prime basis in which we express subsequently the polynomials, *(iii)* the definition and construction of point sets in convexes e.g quadrature point sets and finally *(iv)* polynomials and finite elements. Polynomials can be either scalar, vectorial or matricial. The critical feature, see [6–8], is that they are expressed in a primal basis chosen typically for its good numerical stability and features. The primal basis can be L_2 orthonormal basis — e.g. Legendre on tensorized geometries, Dubiner on simplices, — or the standard moment basis. Polynomial operations — integration, divergence, curl, gradient, hessian ...— are expressed in terms of linear algebra operations.

As for the domain specific language *embedded*(DSEL) in C++, also known as FEEL++[1], it provides the vocabulary for numerical integration, projection and variational formulation. The language manipulates essentially expressions (template) whose values are tensors of rank from 0, 1 and 2 and which are evaluated over basic geometric entities. To illustrate again the capabilities of the library and its language, here is a snippet of a diffusion-advection-reaction equation using Galerkin Least Square stabilisation, see also [5, 6] for more details:

```
AUTO( delta, 1/(1/h() + epsilon/(h()*h()) ) );
AUTO( Aepsi, (-epsilon*trace(hess(v))+ grad(v)*beta + mu*id(v)) );
AUTO( Aepsit, (-epsilon*trace(hesst(u))+ gradt(u)*beta + mu*idt(u)) );
form1( Xh, F )  = integrate( elements(*mesh), im, f*id(v) + delta*f*Aepsi );
form2( Xh, Xh, D, _init=true, _pattern=DOF_PATTERN_COUPLED ) =
integrate( elements(*mesh), im, epsilon*gradt(u)*trans(grad(v))
           +(gradt(u)*beta)*id(v) + mu*idt(u)*id(v)+ delta*Aepsi*Aepsit );
```

4 Current applications

Life is being used in a few research projects: *(i)* the development of stabilisation methods in the context of free surface flows, *(ii)* the development and comparisons of stabilisation methods for elliptic and hyperbolic equations using continuous and discontinuous Galerkin discretization *(iii)* the development of high order methods in space and time for fluid structure interactions models in haemodynamics and *(iv)* the development of a Poisson-Boltzmann solver that will be then used by order reduction methods. The library shall eventually become the new mathematical kernel of the LifeV project(http://www.lifev.org).

Acknowledgments

The author wishes to acknowledge the chair of Prof. A. Quarteroni at EPFL which funded this research from 2005 to 2007, Prof. E. Burman at Univ. of Sussex and S. Deparis, C. Winkelmann, G. Pena and G. Steiner from EPFL for the many fruitful discussions we had.

References

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns, Addison Wesley Professional Computing Series (Addison Wesley, 1995).
[2] Y. Renard and J. Pommier, Getfem++: Generic and efficient c++ library for finite element methods elementary computations, http://www-gmm.insa-toulouse.fr/getfem/.
[3] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, PETSc users manual, Tech. Rep. ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.
[4] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, ACM Trans. Math. Softw. **31**(3), 397–423 (2005).
[5] C. Prud'homme, A domain specific embedded language in c++ for automatic differentiation, projection, integration and variational formulations, Scientific Programming **14**(2), 81–110 (2006).
[6] C. Prud'homme, Life: Overview of a unified c++ implementation of the finite and spectral element methods in 1d, 2d and 3d, in: Workshop On State-Of-The-Art In Scientific And Parallel Computing, , Lecture Notes in Computer Science (Springer-Verlag, dec 2006), p. 10, Accepted.
[7] G. E. Karniadakis and S. J. Sherwin, Spectral/hp element methods for CFD, Numerical Mathematics and Scientific Computation (Oxford University Press, New York, 2005).
[8] R. C. Kirby, ACM Trans. Math. Software **30**(4), 502–516 (2004).

[1] Finite Element Embedded Language in C++