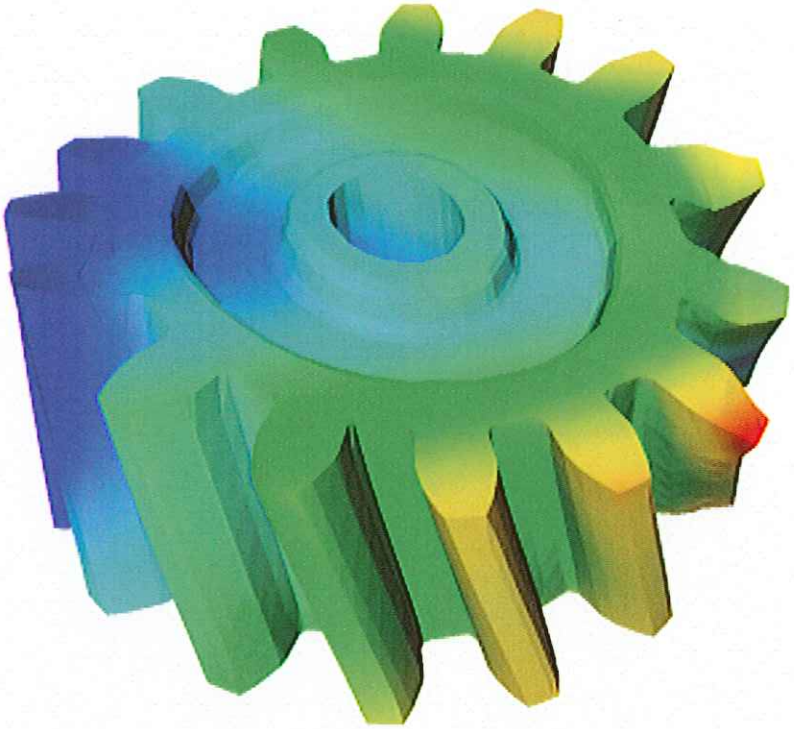

Workshop on Automating the Development of Scientific Computing Software

Book of Abstracts

March 5-7, 2008 at Louisiana State University

SOFTWARE AUTOMATION

Louisiana State University, March 5-7, 2008

L. Ridgway Scott, Peter Brune, Jeff Hammond, Andy Terrel, and Matt Knepley, University of Chicago

We will give a general overview of software automation and explain how it provides a new paradigm in software development that can be viewed as part of a natural evolution that has transformed other areas of systems research. In particular, we compare and contrast the goals and tools of software automation with those of the more traditional area of programming languages and compilers.

We illustrate different areas of software development where this approach has proved essential. In particular, in the area of scientific software development, software automation, is revolutionizing an area with very demanding requirements. We give specific examples of both successes and remaining challenges.

Unicorn - a Unified Continuum Mechanics Solver

Workshop on Automating the Development of Scientific Computing Software

Louisiana State University, March 5-7, 2008

Johan Hoffman

Abstract

We present Unicorn, a FEniCS project (www.fenics.org) focused on the development of a unified continuum mechanics General Galerkin (G2) finite element solver, based on the suite FIAT/FFC/DOLFIN. In a unified continuum formulation (UCF) [1] the continuum mechanics problem is seen as one single continuum with varying constitutive relations, for which the fundamental conservation laws are formulated. For example, with UCF a fluid-structure interaction (FSI) problem is transformed into a multiphase flow problem with the fluid and the structure representing different phases, with different constitutive relations, but which are globally coupled by one single set of conservation laws. UCF not only results in robust FSI discretizations, but also constitutes an ideal framework for phase transition studies and similar.

The implementation in Unicorn is based on an Arbitrary Lagrangian-Eulerian (ALE) G2 discretization, with adaptive mesh refinement based on a posteriori error estimation. G2 is a general and robust methodology for laminar/turbulent compressible/incompressible fluid flow, which together with UCF potentially extends to also general continuum mechanics problems with general constitutive relations.

We present the underlying theoretical framework, the implementation in Unicorn, and applications. We also discuss the roadmap for Unicorn, and future challenges.

References

[1] J.Hoffman, J.Jansson and M.Stöckli, A unified continuum formulation of fluid-structure interaction, *submitted.*

A Symbolic Engine for Finite Element Exterior Calculus

Anders Logg and Kent-Andre Mardal

Simula Research Laboratory
Dept. of Informatics, University of Oslo, Norway

Finite element exterior calculus (Arnold, Falk and Winther 2006) is a framework for design and analysis of stable finite element discretizations of partial differential equations based on concepts from differential geometry, algebraic topology and homological algebra.

In this paper, we take a practical approach to finite element exterior calculus and view it as a tool for generating finite element basis functions. In particular, we demonstrate how to generate the two spaces $\mathcal{P}_r\Lambda^k(T)$ and $\mathcal{P}_r^-\Lambda^k(T)$ on simplices T of arbitrary dimension $n > 0$. For $n = 2, 3$, these spaces specialize to well-known finite element spaces such as Lagrange elements, discontinuous elements, Brezzi–Douglas–Marini elements, Raviart–Thomas elements and Nedelec elements of first and second kind.

Many of these spaces have been implemented before in libraries such as FIAT and SyFi. Although both these libraries automate important steps of the generation of finite element basis functions, the definition of each finite element must still be implemented separately.

By implementing the basic concepts of exterior calculus such as the space of alternating k-forms Alt^k, the exterior (wedge) product $\wedge$, the exterior derivative d and the Koszul operator κ, the spaces $\mathcal{P}_r\Lambda^k(T)$ and $\mathcal{P}_r^-\Lambda^k(T)$ may both be defined in a minimal amount of code (two or three lines) and may then be instantiated for particular values of r, k and n to produce the finite element spaces listed above.

We will give a short introduction to the basic concepts of finite element exterior calculus, and then describe how these concepts map to our implementation in Python. No prior knowledge about finite element exterior calculus is assumed.

Workshop on Automating the Development of Scientific Computing Software
Louisiana State University, March 5–7, 2008

Compiling variational forms for $H(\mathrm{div})$ and $H(\mathrm{curl})$ conforming finite elements

Marie E. Rognes, Robert C. Kirby and Anders Logg

$H(\mathrm{div})$ and $H(\mathrm{curl})$ conforming finite elements can be used in a wide range of applications. Important examples include mixed finite element methods for second order elliptic PDEs, robust methods for mixed elasticity and Darcy-Stokes flow and Maxwell's equations of electromagnetism. However, these element spaces have not reached as widespread use as the more standard H^1 conforming elements. This can be partially contributed to the perceived difficulty of implementation and the lack of easily available finite element software supporting these types of elements in a flexible manner. To remedy this, we have extended the FEniCS form compiler FFC, with the necessary framework for the handling of $H(\mathrm{div})$ and $H(\mathrm{curl})$ conforming elements.

A main complicating issue in this regard is the partial continuity requirement for such element spaces, namely, the continuity of directional components over inter-element facets. In particular, the appropriate mapping of the finite element spaces from the reference to the physical elements are now the Piola mappings. These mappings have been implemented within the form evaluation framework of FFC. Further, it is worth noting how the UFC specification for the numbering of mesh entities greatly enhances this process. We conclude by demonstrating convergence rates for a range of element spaces in 2 and 3 dimensions and by illustrating the ease with which complicated mixed systems, such as the equations for linear elasticity with weak symmetry, can be formulated.

Compressible Flow and Vector Boundary Conditions in Unicorn

Workshop on Automating the Development of Scientific Computing Software

Louisiana State University, March 5-7, 2008

Murtazo Nazarov

Abstract

We describe the General Galerkin (G2) finite element method for the compressible Euler equations [1] implemented in Unicorn, a FEniCS project (www.fenics.org) focused on the development of a unified continuum mechanics solver, based on the suite FIAT/FFC/DOLFIN.

The G2 method is characterized by exact conservation of mass, momentum and energy, and satisfies an energy estimate relating the loss of kinetic energy to increase of internal (heat) energy through turbulence and shocks.

An adaptive algorithm based on a general framework for a posteriori error estimation is implemented in Unicorn. This adaptive algorithm is illustrated in several examples, showing an improved approximation as the mesh is refined. The mesh refinement criterion is based on $\epsilon = \|hR(U)\|$, with $h = h(x,t)$ the local mesh size and $R(U)$ the residual for the approximate G2 solution U. The refinement criterion couples to improving approximation of U seen as an approximate weak (ϵ-weak) solution to the compressible Euler equations, see [2].

We also describe our implementation of the special type of boundary conditions connected e.g. to fluid flow, with different boundary conditions in the normal and tangent directions of the boundary. In particular, for the Euler equations we use slip boundary conditions, with the normal velocity component being zero, but the tangent component being free. We discuss issues related to the implementation for linear Lagrange finite elements in Unicorn, and also possible extensions to a general finite element framework.

Computational examples with Unicorn is also presented.

References

[1] J.Hoffman, J.Jansson and M.Nazarov, A General Galerkin Finite Element Method for the Compressible Euler Equations, *submitted.*

[2] J.Hoffman and C.Johnson, *Computational Turbulent Incompressible Flow*, Springer, 2007.

LNG_FEM: GENERATING GRADED MESHES AND SOLVING ELLIPTIC EQUATIONS ON 2-D DOMAINS OF POLYGONAL STRUCTURES

HENGGUANG LI AND VICTOR NISTOR

We develop LNG_FEM, a software package for the graded mesh generation and for solving elliptic equations. LNG_FEM generates user-specified graded meshes on arbitrary two-dimensional domains with straight edges, for different boundary conditions, once initial information is passed to the program by appropriately filling out some source files. We focus on a detailed instruction on the implementation of the software after a brief literature review of elliptic boundary value problems and graded meshes. Then, we show examples to point out that LNG_FEM is equipped with advanced algorithms and data structures to perform efficiently. LNG_FEM is to popularize the use and understanding of graded meshes in the finite element method. We hope it triggers more interest and discoveries both in academia and in industry.

Department of Mathematics, The Pennsylvania State University (UP), *E-mail address*: li_h@math.psu.edu; nistor@math.psu.edu.

Adaptivity and Automated Error Control in FEniCS

Workshop on Automating the Development of Scientific Computing Software

Louisiana State University, March 5-7, 2008

Johan Jansson

Abstract

Unicorn is a free software FEniCS project (www.fenics.org) based on the suite FIAT/FFC/DOLFIN, focused on the development of a unified continuum mechanics adaptive finite element solver. An automated process for generating adaptive methods using a posteriori error estimation with duality [3, 4] is described. The process consists of:

1. Generation of a dual problem
2. Computation of a solution to the dual problem
3. Computation of a residual function
4. Computation of error indicators from the a posteriori error estimate
5. Mesh (de)refinement according to the error indicators

This automated process is implementable part in a computer algebra system (computing derivatives symbolically and algebraic manipulation) and part in FEniCS (evaluating forms, assembling and solving discrete systems).

The implementation in Unicorn of steps 3-5 of this algorithm is discussed, and we describe tools and algorithms for error estimation and mesh refinement/coarsening [1, 8, 7, 2], implemented in Unicorn and DOLFIN. In addition, related algorithms for elastic mesh smoothing and projections between non-matching meshes are described.

We show examples of adaptive computations using Unicorn for different applications in continuum mechanics [6, 5], where we illustrate the performance of the algorithms.

References

[1] D. N. ARNOLD, A. MUKHERJEE, AND L. POULY, *Locally adapted tetrahedral meshes using bisection*, SIAM Journal on Scientific Computing, 22 (2000), pp. 431–448.

[2] J. BARRY, *Construction of three-dimensional improved-quality triangulations using local transformations*, SIAM J. Sci. Comput., 16 (1995), pp. 1292–1307.

[3] K. ERIKSSON, D. ESTEP, P. HANSBO, AND C. JOHNSON, *Computational Differential Equations*, Cambridge University Press New York, 1996.

[4] D. ESTEP, *A short course on duality, adjoint operators, Green's functions, and a posteriori error analysis*, 2004.

[5] J. HOFFMAN, *Adaptive simulation of the turbulent flow past a sphere*, J. Fluid Mech., 568 (2006), pp. 77–88.

[6] J. HOFFMAN, J. JANSSON, AND M. STÖCKLI, *A unified continuum formulation of fluid structure interaction*, (in preparation).

[7] C. OLLIVIER-GOOCH, *An unstructured mesh improvement toolkit with application to mesh improvement, generation, and (de-) refinement*, AIAA 36th Aerospace Sciences Meeting, Reno, Nevada, January 1998, (1998).

[8] M.-C. RIVARA, *Local modification of meshes for adaptive and/or multigrid finite-element methods*, Journal of Computational and Applied Mathematics, 36 (1992), pp. 78–89.

A Form Compiler based on Symbolic Computations

Martin Sandve Alnæs and Kent-Andre Mardal, Simula Research Laboratory

February 26, 2008

1 Introduction

SyFi is a software package for symbolic finite element computations, which is now part of the FEniCS project. It consists of two main parts: a kernel and a form compiler.

The kernel consists of a collection of tools for symbolic computations on polynomial spaces and polygonal domains, and a collection of elements. The SyFi kernel, written in C++, is roughly comparable to FIAT. which tabulates finite elements using numerical techniques.

The SyFi Form Compiler (SFC) is a Python module comparable to FFC. It takes as input a symbolic description of a variational form and a set of finite elements and generates low level C++ code. The generated code complies with the UFC interface and can be used in (Py)DOLFIN to assemble matrices and vectors. SFC supports Just-In-Time compilation via the package Instant, such that we can define the variational form within Python, then automatically compile generated C++ code into an extension module which is dynamically loaded back into Python.

This combination of symbolic mathematics and code generation enables the specification of finite element discretizations in a user-friendly environment while maintaining efficiency. In addition, the symbolic framework allow us to do certain calculations automatically that we earlier typically did by hand, e.g. symbolic differentiation which simplifies e.g. the implementation of complex material laws for hyper-elastic materials, and the calculation of the Jacobian in the case of a nonlinear PDE.

We will present a short code example using SFC together with PyDOLFIN, before we present some benchmarks showing the efficiency of the generated code for computing the element tensor for a few forms. We will compare with quadrature based examples written in Diffpack and Deal.II, and code generated by FFC. The code generated by SFC is run with both analytic integration and quadrature, and with and without attempts at further optimization.

2 Conclusions

In cases where analytic integration is feasible, it can result in a major speedup in the computation of the element tensor. For elements defined on simplices, the performance of SFC is roughly the same as FFC (without FErari). One point to notice is that if coefficient functions are involved in nonlinear terms in the form, the expressions blow up as the order of their finite element spaces increase and the efficiency gain from analytic integration may be lost. A good factorization algorithm could maybe remedy this problem. Preliminary attempts to optimize the symbolic expressions prior to code generation have been only partially successfull.

Moreover, the code generation process can be quite intensive in terms of memory usage and computing time. The code generation based on quadrature is less demanding, and the resulting code is (as will be demonstrated) much more efficient than the implementations in Diffpack and Deal.II, the reason being that Diffpack and Deal.II provide their abstractions at the C++ level. Thus, both the efficiency gain caused by generating low level code and the advantages of working with a more abstract input format based on symbolic tools are retained independent of integration method chosen.

CACTUS: AUTOMATIC COUPLING OF SOFTWARE FRAMEWORK COMPONENTS

Louisiana State University, March 5-7, 2008

Erik Schnetter and Gabrielle Allen

Cactus is a software framework supporting distributed, collaborative code development for high-performance computing. As in other frameworks, an application consists of a set of interacting components, where the main program is provided by the framework. Each component declares variables and functions, and can access public variables from other components. Parallelism is provided by a driver layer that distributes the grid (or mesh) onto the available processors, manages memory for the components' variables, and calls the components' functions as appropriate. Components can be developed independently and are only brought together at run time.

Different from many other software frameworks, Cactus requires no plumbing to connect components. Starting from the components' variable and function declarations, code is generated to orchestrate the dataflow between the components, under run-time control of the driver layer. This ultimately allows efficient, tight coupling between components on distributed-memory high-performance computing systems. Other benefits of this declarative framework architecture include facilitation of unit tests or automated metadata collection, since the framework can inspect the internal state of the components.

KRANC: AUTOMATIC CODE GENERATION FOR HYPERBOLIC PDE USING THE CACTUS FRAMEWORK

Louisiana State University, March 5-7, 2008

Erik Schnetter, Ian Hinder

We perform time evolutions of general relativistic systems such as black holes, neutron stars, or binaries of these. For this we solve the Einstein field equations in 3+1 dimensions. The complexity of the Einstein field equations containing thousands of terms makes it highly desirable to generate the evolution kernels automatically. This simplifies implementing alternative formulations, reduces the likelihood of errors, and also allows performance optimizations requiring restructuring the code.

We introduce Kranc, a mathematics-based toolkit that discretises coupled hyperbolic partial differential equations and implements them in C or Fortran using the Cactus framework, starting from equations given in native mathematical format. One specialty of Kranc is its support for tensors in the abstract index notation, which is commonly used in general relativity.

GENERATION OF DENSE LINEAR ALGEBRA SOFTWARE FOR SHARED MEMORY AND MULTI-CORE ARCHITECTURES

Louisiana State University, March 5-7, 2008

Paolo Bientinesi, Duke University

When writing scientific computing software, programmers often need to identify which algorithm would perform best in a given situation. In dense linear algebra, the answer depends on a large number of factors, ranging from processor type and architectural features to matrix size and performance signature of the used BLAS.

I will show that when targeting shared memory and multi-core processors, one must take into account not only different algorithms, but also different types of parallelism. In this talk, I illustrate two approaches. One approach uses blocking and careful scheduling to attain high performance while the other leverages multi-threaded BLAS. I will discuss how the generation of algorithms and code can be automated in both scenarios.

Two Steps Towards Automating Efficient Solution of Inverse Problems

Kent-Andre Mardal*, Bjørn Fredrik Nielsen† and Martin Alnæs‡

January 31, 2008

We have combined symbolic mathematics with code generation to create a user–friendly environment for specifying finite element methods. The motivation behind this approach is to automate the boring and error-prone task of implementing variational forms of PDEs, which is a cornerstone when developing finite element simulators. By employing a symbolic engine in a high–level language we allow the user to specify the weak form of the PDE in an abstract and user–friendly format. Furthermore, the symbolic framework allows us to do certain calculations like differentiation automatically. In addition to the symbolic framework, we use a high–level library for expressing linear algebra algorithms in terms of block matrices and block preconditioners.

Our efforts have resulted in the open source software package SyFi, which is part of the FEniCS project. SyFi stands for Symbolic Finite elements and is implemented in C++ and Python. SyFi is built on top of the symbolic C++ library GiNaC and uses its Python interface Swiginac. In addition to SyFi, we employ the in-house code PyCC for block matrices and preconditioners.

Some FEniCS developments for mechanics problems

Garth N. Wells[1*] Kristian B. Ølgaard[2†]

[1]University of Cambridge
[2]Delft University of Technology

We will present a number of recent FEniCS developments which have been driven by the need to solve various mechanics problems. Developments include: support for quadrature assembly approaches using FFC; the introduction of 'quadrature' functions (functions whose values are known only at discrete points and therefore cannot be differentiated); and support for discontinuous Galerkin methods.

A number of outstanding issues will be discussed, including automation for complicated forms and high-performance parallel computation. Also, the point of how best to capitalise upon external interest in FEniCS developments will be raised.

*gnw20@cam.ac.uk
†k.b.oelgaard@tudelft.nl

FIAT: Overview, Applications and Future Trends

Louisiana State University, March 5-7, 2008

Robert C. Kirby, Texas Tech University

FIAT is a FEniCS software component that provides for the construction and evaluation of general simplicial finite elements, including for H(div) and H(curl).

This talk will describe how Ciarlet's abstract definitions may be turned into a computational paradigm for evaluating nodal basis functions, providing an example of a nonstandard element for Darcy-Stokes flow. Finally, I will describe what FIAT needs to provide for full use in evaluation of variational forms.

On the Role of Computation in Mathematics: Blow Up of Euler Solutions

Louisiana State University, March 5-7, 2008

Johan Hoffman and Claes Johnson

We show that computed solutions could replace exact solutions for well-posed problems. We show well posedness of mean-value outputs of computed viscosity solutions of the incompressible Euler equations.

We find blowup of such solutions and thus present evidence toward a solution of the Clay Institute Millennium Problem on fluid dynamics.

Can We Teach Computers to Write Faster Libraries?

Louisiana State University, March 5-7, 2008

Markus Püschel

As the computing world "goes multicore," high performance library development finally becomes a nightmare. Optimal programs, and their underlying algorithms, have to be adapted to take full advantage of the platform's parallelism, memory hierarchy, and available instruction set. To make things worse, the best implementations are often platform-dependent and platforms are constantly evolving, which quickly renders libraries obsolete. As a consequence, developers are forced to permanently re-implement and re-optimize the same functionality and often even revert to assembly coding just as 50 years ago. A number of research efforts have started to address this problem in a new area called Automatic Performance Tuning with the common goal to rethink the way libraries are created.

In this talk, we present Spiral (www.spiral.net), a program generation system for linear transforms. Spiral generates highly optimized, platform-tuned implementations of transforms directly from a problem specification. For a user-specified transform, Spiral generates alternative algorithms, optimizes them, compiles them into programs, and "intelligently" searches for the best match to the computing platform. The main idea behind Spiral is a mathematical, declarative framework to represent algorithms and the use of rewriting systems to generate and optimize algorithms at a high level of abstraction. Optimization includes parallelization for vector architectures, shared and distributed memory platforms, GPUs, and even FPGAs. Experimental results show that the code generated by Spiral competes with, and sometimes outperforms, the best available human-written library code. Further, recent research shows that it may be possible to extend Spiral into other domains such as coding or linear algebra.

As for the question in the title: Spiral shows that, at least for well-understood problem domains, a positive answer may be in reach.