FEniCS-HPC: Automated predictive high-performance finite element computing with applications in aerodynamics

Johan Hoffman[1,2,3], Johan Jansson[2,1,4], and Niclas Jansson[1,5]

[1] Computational Technology Laboratory, School of Computer Science and Communication, KTH, Stockholm, Sweden
[2] BCAM - Basque Center for Applied Mathematics, Bilbao, Spain
[3] `jhoffman@kth.se`
[4] `jjan@kth.se`
[5] `njansson@kth.se`

Abstract. Developing multiphysics finite element methods (FEM) and scalable HPC implementations can be very challenging in terms of software complexity and performance, even more so with the addition of goal-oriented adaptive mesh refinement. To manage the complexity we in this work present *general* adaptive stabilized methods with *automated* implementation in the FEniCS-HPC *automated* open source software framework. This allows taking the weak form of a partial differential equation (PDE) as input in near-mathematical notation and automatically generating the low-level implementation source code and auxiliary equations and quantities necessary for the adaptivity. We demonstrate new optimal strong scaling results for the whole adaptive framework applied to turbulent flow on massively parallel architectures down to 25000 vertices per core with ca. 5000 cores with the MPI-based PETSc backend and for assembly down to 500 vertices per core with ca. 20000 cores with the PGAS-based JANPACK backend. As a demonstration of the power of the combination of the scalability together with the adaptive methodology allowing prediction of gross quantities in turbulent flow we present an application in aerodynamics of a full DLR-F11 aircraft in connection with the HiLift-PW2 benchmarking workshop with good match to experiments.

Keywords: FEM, adaptive, turbulence

1 Introduction

As computational methods are applied to simulate even more advanced problems of coupled physical processes and supercomputing hardware is developed towards massively parallel heterogeneous systems, it is a major challenge to manage the complexity and performance of methods, algorithms and software implementations. Adaptive methods based on quantitative error control pose additional challenges. For simulation based on partial differential equation (PDE) models, the finite element method (FEM) offers a general approach to numerical discretisation, which opens for automation of algorithms and software implementation.

In this paper we present the FEniCS-HPC open source software framework with the goal to combine the generality of FEM with performance, by optimisation of generic algorithms [4, 2, 13]. We demonstrate the performance of FEniCS-HPC in an application to subsonic aerodynamics.

We give an overview of the methodology and the FEniCS-HPC framework, key aspects of the framework include:

1. **Automated discretization** where the weak form of a PDE in mathematical notation is translated into a system of algebraic equations using code generation.
2. **Automated error control**, ensures that the discretization error e = u - U in a given quantity is smaller than a given tolerance by adaptive mesh refinement based on duality-based a posteriori error estimates. An a posteri error estimate and error indicators are automatically generated from the weak form of the PDE, by directly using the error representation.
3. **Automated modeling**, which includes a residual based implicit turbulence model, where the turbulent dissipation comes only from the numerical stabilization, as well as treating the fluid and solid in fluid-structure interaction (FSI) as one continuum with a phase indicator function tracked by a moving mesh and implicitly modeling contact.

We demonstrate new optimal strong scaling results for the whole adaptive framework applied to turbulent flow on massively parallel architectures down to 25000 vertices per core with ca. 5000 cores with the MPI-based PETSc backend and for assembly down to 500 vertices per core with ca. 20000 cores with the PGAS-based JANPACK backend. We also present an application in aerodynamics of a full DLR-F11 aircraft in connection with the HiLift-PW2 benchmarking workshop with good match to experiments.

1.1 The FEniCS project and state of the art

The software described here is part of the FEniCS project [2], with the goal to automate the scientific software process by relying on general implementations and code generation, for robustness and to enable high speed of software development.

Deal.II [1] is a software framework with a similar goal, implementing general PDE based on FEM in C++ where users write the "numerical integration loop" for weak forms for computing the linear systems. The framework runs on supercomputers with optimal strong scaling. Deal.II is based on quadrilater (2D) and hexahedral (3D) meshes, whereas FEniCS is based on simplicial meshes (triangles in 2D and tetrahedra in 3D).

Another FEM software framework with a similar goal is FreeFEM++ [3], which has a high-level syntax close to mathematical notation, and has demonstrated optimal strong scaling up to ca. 100 cores.

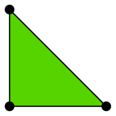
2 The FEniCS-HPC framework

FEniCS-HPC is a problem-solving environment (PSE) for automated solution of PDE by the FEM with a high-level interface for the basic concepts of FEM: weak forms, meshes, refinement, sparse linear algebra, and with HPC concepts such as partitioning, load balancing abstracted away.

The framework is based on components with clearly defined responsibilities. A compact description of the main components follows, with their dependencies shown in the dependency diagram in Figure 1:

FIAT: Automated generation of finite element spaces V and basis functions $\phi \in V$ on the reference cell and numerical integration with FInite element Automated Tabulator (FIAT) [13, 12]

$$e = (K, V, \mathcal{L})$$



where K is a cell in a mesh $\mathcal{T}$, V is a finite-dimensional function space, $\mathcal{L}$ is a set of degrees of freedom.

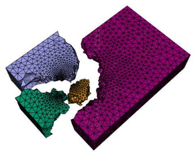
FFC+UFL: Automated evaluation of weak forms in mathematical notation on one cell based on code generation with Unified Form Language (UFL) and FEniCS Form Compiler (FFC) [13, 11], using the basis functions $\phi \in V$ from FIAT. For example, in the case of the Laplacian operator

$$A^K_{ij} = a_K(\phi_i, \phi_j) = \int_K \nabla\phi_i \cdot \nabla\phi_j dx = \int_K lhs(r(\phi_i, \phi_j)dx)$$

where A^K is the element stiffness matrix and $r(\cdot,\cdot)$ is the weak residual.

DOLFIN-HPC: Automated high performance assembly of weak forms and interface to linear algebra of discrete systems and mesh refinement on a distributed mesh $\mathcal{T}_\Omega$ [10].

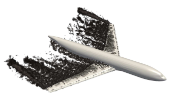
```
$A = 0$
for all cells $K \in \mathcal{T}_\Omega$
    $A$ += $A^K$

$Ax = b$
```

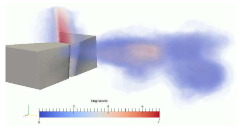


Unicorn: Automated Unified Continuum modeling with Unicorn choosing a specific weak residual form for incompressible balance equations of mass and momentum with example visualizations of aircraft simulation below left and turbulent FSI in vocal folds below right [4].

$$r_{UC}((v,q),(u,p)) = (v, \rho(\partial_t u + (u\cdot\nabla)u) + \nabla\cdot\sigma - g) + (q, \nabla\cdot u) + LS((v,q),(u,p))$$

where LS is a least-squares stabilizing term described in [7].





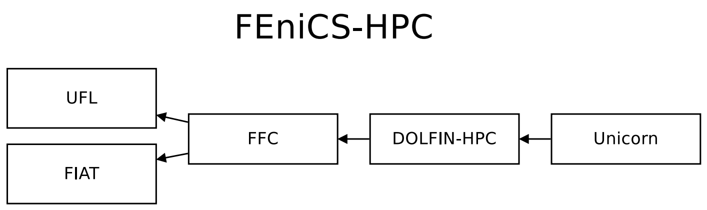


Fig. 1: FEniCS-HPC component dependency diagram.

A user of FEniCS-HPC writes the weak forms in the UFL language, compiles it with FFC, and includes it in a high-level "solver" written in C++ in DOLFIN-HPC to read in a mesh, assemble the forms, solve linear systems, refine the mesh, etc. The Unicorn solver for adaptive computation of turbulent flow and FSI is developed as part of FEniCS-HPC.

2.1 Solving PDE problems in FEniCS-HPC

Poisson's equation To solve Poisson's equation in weak form $\int_{\Omega}(\nabla u, \nabla v) - (f, u) = 0 \quad \forall v \in V$ in the framework, we first define the weak form in a UFL "form file", closely mapping mathematical notation (see Figure 2). The form file is then compiled to low-level C++ source code for assembling the local element matrix and vector with FFC. Finally we use DOLFIN-HPC to write a high-level "solver" in C++, composing the different abstractions, where a mesh is defined, the global matrix and vector are assembled by interfacing to the generated source code, the linear system is solved by an abstract parallel linear algebra interface (using PETSc as back-end by default), and then the solution function is saved to disk. The source code for an example solver is presented in Figure 2.

```
Q = FiniteElement("CG", "tetrahedron", 1)

v  = TestFunction(Q)  # test basis function
u  = TrialFunction(Q) # trial basis function
f  = Coefficient(Q) # function

# Bilinear and linear forms
a = dot(grad(v), grad(u))*dx
L = v*f*dx
```

```
// Define mesh, BCs and coefficients
PoissonBoundary boundary;
PoissonBoundaryValue u0(mesh);
SourceFunction f(mesh);
DirichletBC bc(u0, mesh, boundary);

// Define PDE
PoissonBilinearForm a;
PoissonLinearForm L(f);
LinearPDE pde(a, L, mesh, bc);

// Solve PDE
Function u;
pde.solve(u);

// Save solution to file
File file(``poisson.pvd'');
file << u;
```

Fig. 2: Poisson solver in FEniCS-HPC with the weak form in the UFL language (left) and the solver in C++ using DOLFIN-HPC (right).

The incompressible Navier-Stokes equations We formulate the General Galerkin (G2) method for incompressible Navier-Stokes equations (1) in UFL by a direct input of the weak residual. We can automatically derive the Jacobian in a quasi-Newton fixed-point formulation and also automatically linearize and generate the adjoint problem needed for adaptive error control. These examples are presented in Figure 3

```
V = VectorElement("CG", "tetrahedron", 1)
Q = FiniteElement("CG", "tetrahedron", 1)

v = TestFunction(V); q = TestFunction(Q)
u_ = TrialFunction(V); p_ = TrialFunction(Q)
u = Coefficient(V); p = Coefficient(Q)
u0 = Coefficient(V); um = 0.5*(u + u0)

# Momentum and continuity weak residuals
r_m = (inner(u - u0, v)/k + \
 ((nu*inner(grad(um), grad(v)) + \
 inner(grad(p) + grad(um)*um, v))))*dx + LS_u*dx
r_c = inner(div(u), q))*dx + LS_p*dx

# Newton's method Ju_i+1 = Ju_i - F(u_i)
a = derivative(r_m, u, u_)
L = action(a, u) - r_m
```

```
# Adjoint problem (stationary part) for r_m
a_adjoint = adjoint(derivative(r_m - inner(u, v)/k*dx, u))
L_adjoint_c = derivative(action(r_c, p), u, v)
L_adjoint = inner(psi_m, v)*dx - L_adjoint_c
```

Fig. 3: Example of weak forms in UFL notation for the cG(1)cG(1) method for incompressible Navier-Stokes equations (left) together with the adjoint problem (right).

3 Parallelization strategy and performance

The parallelization is based on a fully distributed mesh approach, where everything from preprocessing, assembly of linear systems, postprocessing and refinement is performed in parallel, without representing the entire problem or any pre-/postprocessing step on a single core

Inital data distribution is defined by the graph partitioning of the corresponding dual graph of the mesh. Each core is assigned a set of whole elements and the vertex overlap between cores is represented as ghosted entities.

3.1 Parallel assembly

The assembling of the global matrix is performed in a straightforward fashion. Each core computes the local matrix of the local elements and add them to the global matrix. Since we assign whole elements to each core, we can minimize data dependency during assembly. Furthermore, we renumber all the degrees of freedom such that a minimal amount of communication is required when modifying entries in the sparse matrix.

3.2 Solution of discrete system

The FEM discretization generates a non-linear algebraic equation system to be solved for each time step. In Unicorn we solve this by iterating between the velocity and pressure equations by a Picard or quasi-Newton iteration [6].

Each iteration in turn generates a linear system to be solved. We use simple Krylov solvers and preconditioners which scale well to many cores, typically BiCGSTAB with a block-Jacobi preconditioner, where each block is solved with ILU(0).

3.3 Mesh refinement

Local mesh refinement is based around a parallelization of the well known recursive longest edge bisection method [15]. The parallelization splits up the refinement into two phases. First a local serial refinement phase bisects all elements marked for refinement on each core (concurrently) leaving several hanging nodes on the shared interface between cores. The second phase propagates these hanging nodes onto adjacent cores.

The algorithm iterates between local refinement and global propagation until all cores are free of hanging nodes. For an efficent implementation, one has to detect when all cores are idling at the same time. Our implementation uses a fully distributed termination detection scheme, which includes termination detection in the global propagation step by using recusive doubling or hypercube exchange type communication patterns [10]. Also, the termination detection algorithm does not have a central point of control, hence no bottlenecks, less message contention, and no problems with load imbalance.

Dynamic load balancing In order to sustain good load balance across several adaptive iterations, dynamic load balancing is needed. DOLFIN-HPC is equipped with a scratch and remap type load balancer, based on the widely used PLUM scheme [14], where the new partitions are assigned in an optimal way by solving the maximally weighted bipartite graph problem. We have improved the scheme such that it scales linearly to thousands of cores [10, 8].

Furthermore, we have extended the load balancer with an a priori workload estimation. With a dry run of the refinement algorithm, we add weights to a dual graph of the mesh, corresponding to the workload after refinement. Finally, we repartition the unrefined mesh according to the weighted dual graph and redistribute the new partitions before the refinement.

4 Strong scalability

To be able to take advantage of available supercomputers today the entire solver in FEniCS-HPC needs to demonstrate good strong scaling to at least several thousands of cores. For planned “exascale” systems with many million cores, strong scalability has to be attained for at least hundreds of thousands of cores.

In this section we analyze scaling results using the PETSc parallel linear algebra backend based on pure MPI and the JANPACK backend based on PGAS.

In Figure 4 we present strong scalability results with the PETSc pure MPI backend for the full G2 method for turbulent incompressible Navier-Stokes equations (1) (assemble linear systems and solve the momentum and continuity equations) in 3D on a mesh with 147M vertices on the Hornet Cray XC40 computer. We observe near-optimal scaling to ca. 4.6 kcores for all the main algorithms (assembly and linear solves). Going from 4.6 kcores to 9.2 kcores we start to see a degradation in the scaling with a speedup of ca. 0.7, and from 9.2 kcores to 18.4 kcores the speedup is 0.5. It's clear that it's mainly the assembly that shows degraded scaling.

In Figure 5 we present results for assembling four different equations using the JANPACK backend, where FEniCS-HPC is running in a hybrid MPI+PGAS mode. We observe that for large number of cores, the low latency one-sided communication of PGAS languages in combination with our new sparse matrix format [9] greatly improves the scalability.

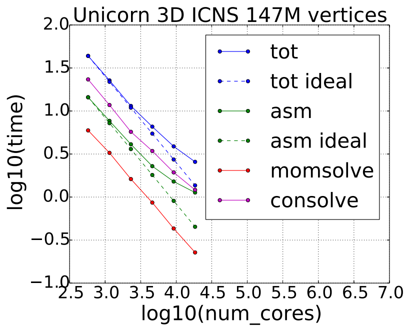


Fig. 4: Strong scalability test for the full G2 method for incompressible turbulent Navier-Stokes equations (assemble linear systems and solve momentum and continuity) in 3D on a Cray XC40.

5 Unicorn simulation of a full aircraft

In the Unicorn component we implement the full G2 method and fix the weak residual to the cG(1)cG(1) stabilized space-time method for incompressible Navier-Stokes equations (or a general stress for FSI)

In a cG(1)cG(1) method [7] we seek an approximate space-time solution $\hat{U} = (U, P)$ which is continuous piecewise linear in space and time (equivalent to the implicit Crank-Nicolson method). With I a time interval with subintervals $I_n = (t_{n-1}, t_n)$, W^n a standard spatial finite element space of continuous piecewise linear functions, and W_0^n the functions in W^n which are zero on the boundary Γ, the cG(1)cG(1) method for constant density incompressible flow with homogeneous Dirichlet boundary conditions for the velocity takes the form: for $n = 1, ..., N$, find $(U^n, P^n) \equiv (U(t_n), P(t_n))$ with $U^n \in V_0^n \equiv [W_0^n]^3$ and

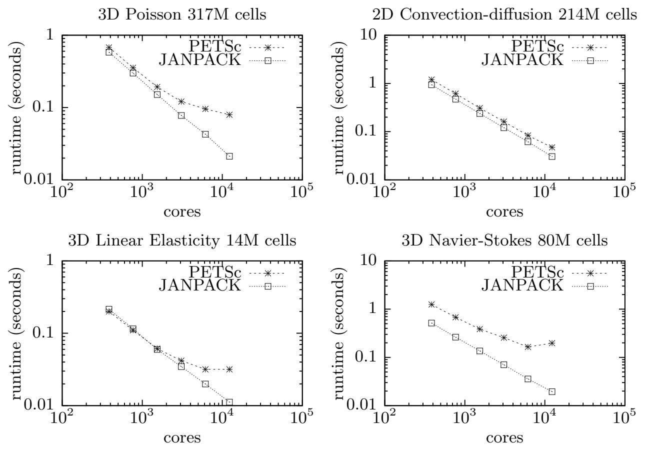


Fig. 5: Sparse matrix assembly timings for four different equations on a Cray XC40.

$P^n \in W^n$, such that

$$\begin{aligned} r((U,P),(v,q)) = ((U^n - U^{n-1})k_n^{-1} + (\bar{U}^n \cdot \nabla)\bar{U}^n, v) + (2\nu\epsilon(\bar{U}^n), \epsilon(v)) \\ - (P, \nabla \cdot v) + (\nabla \cdot \bar{U}^n, q) + LS = 0, \ , \forall \hat{v} = (v,q) \in V_0^n \times W^n \end{aligned} \tag{1}$$

where $\bar{U}^n = 1/2(U^n + U^{n-1})$ is piecewise constant in time over I_n and LS a least-squares stabilizing term described in [7].

We formulate a new general adjoint-based method for adaptive error control based on the following error representation and adjoint weak bilinear and linear forms with the error $\hat{e} = \hat{u} - \hat{U}$, adjoint solution $\hat{\phi}$, output quantity ψ and the hat signifying the full velocity-pressure vector $\hat{U} = (U,P)$, with $r_G = r - LS$:

$$(\hat{e},\psi) = \overline{r'}(\hat{e},\hat{\phi}) = r_G(\hat{U};\hat{\phi}) \quad a_{adjoint}(v,\hat{\phi}) = \overline{r'}(v,\hat{\phi}) \quad L_{adjoint}(v) = (v,\psi) \tag{2}$$

We have used our adaptive finite element methodology for turbulent flow and FEniCS-HPC software to solve the incompressible Navier-Stokes equations of the flow past a full high-lift aircraft model (DLR-F11) with complex geometry at realistic Reynolds number for take-off and landing. This work is an extension of our contributed simulation results to the 2nd AIAA CFD High-Lift Prediction Workshop (HiLiftPW-2), in San Diego, California, in 2013 [5].

In the following results we focus on the angle of attack $\alpha = 18.5°$. To quantify mesh-convergence we plot the coefficients and their relative error compared to the experimental values (serving as the reference) versus the number of vertices in the meshes, and plot meshes and volume renderings of quantities related to the adaptivity in Figure 6.

We see that our adaptive computational results come very close to the experimental results on the finest mesh, with a relative error under 1% for cl and cd. For other angles we observe similar results presented in [5].

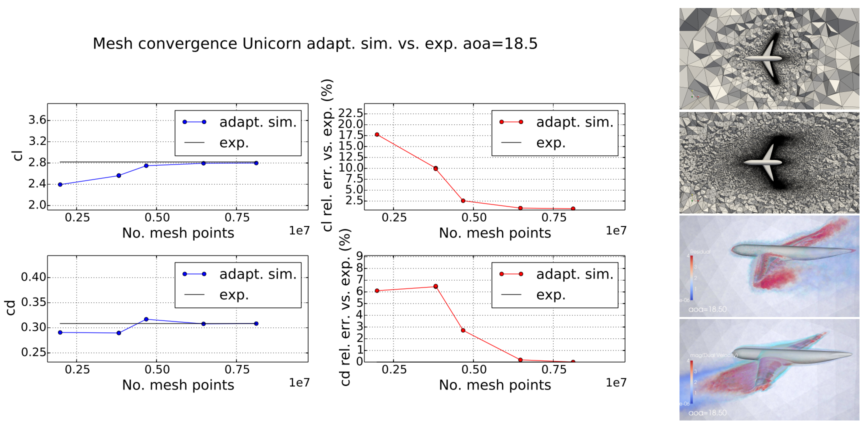


Fig. 6: Plots for the aircraft simulation at $\alpha = 18.5°$. Lift coefficient, cl, and drag coefficient, cd, vs. angle of attack, α, for the different meshes from the iterative adaptive method (left). Slice aligned with the angle of attack showing the tetrahedra of the starting mesh versus the finest adaptive mesh (top right). Volume rendering of the velocity residual and adjoint velocity magnitude (bottom right).

6 Summary

We have given an overview of the general FEniCS-HPC software framework for automated solution of PDE, taking the weak form as input in near-mathematical notation, with automated discretization and a new simple method for adaptive error control, suitable for parallel implementation. On the Hornet Cray XC40 supercomputer we demonstrate new optimal strong scaling results for the whole adaptive framework applied to turbulent flow on massively parallel architectures down to 25000 vertices per core with ca. 5000 cores with the MPI-based PETSc backend and for assembly down to 500 vertices per core with ca. 20000 cores with the PGAS-based JANPACK backend.

Using the Unicorn component in FEniCS-HPC we have simulated the aerodynamics of a full DLR-F11 aircraft in connection with the HiLift-PW2 benchmarking workshop. We find that the simulation results compare very well with experimental data; moreover, we show mesh-convergence by the adaptive method, while using a low number of spatial degrees of freedom.

Acknowledgments

This research has been supported by EU-FET grant EUNISON 308874, the European Research Council, the Swedish Foundation for Strategic Research, the Swedish Research Council, the Basque Excellence Research Center (BERC 2014-2017) program by the Basque Government, the Spanish Ministry of Economy and

Competitiveness MINECO: BCAM Severo Ochoa accreditation SEV-2013-0323 and the Project of the Spanish MINECO: MTM2013-40824.

We acknowledge PRACE for awarding us access to the supercomputer resources Hermit, Hornet and SuperMUC based in Germany at The High Performance Computing Center Stuttgart (HLRS) and Leibniz Supercomputing Center (LRZ), from the Swedish National Infrastructure for Computing (SNIC) at PDC – Center for High-Performance Computing and on resources provided by the "Red Española de Supercomputación" and the "Barcelona Supercomputing Center - Centro Nacional de Supercomputación".

We would also like to acknowledge the FEniCS and FEniCS-HPC developers globally.

References

1. W. Bangerth, R. Hartmann, and G. Kanschat. deal.II — a general-purpose object-oriented finite element library. *ACM Trans. Math. Softw.*, 33(4), 2007.
2. FEniCS. FEniCS project, 2003. http://www.fenicsproject.org.
3. F. Hecht. New development in freefem++. *J. Numer. Math.*, 20, 2012.
4. J. Hoffman, J. Jansson, R. Vilela de Abreu, N. C. Degirmenci, N. Jansson, K. Müller, M. Nazarov, and J. H. Spühler. Unicorn: Parallel adaptive finite element simulation of turbulent flow and fluid-structure interaction for deforming domains and complex geometry. *Comput. Fluids*, 80(0):310 – 319, 2013.
5. J. Hoffman, J. Jansson, N. Jansson, and R. Vilela De Abreu. Towards a parameter-free method for high reynolds number turbulent flow simulation based on adaptive finite element approximation. *Computer Methods in Applied Mechanics and Engineering*, 288(0):60 – 74, 2015.
6. J. Hoffman, J. Jansson, and M. Stöckli. Unified continuum modeling of fluid-structure interaction. *Math. Mod. Meth. Appl. S.*, 2011.
7. Johan Hoffman and Claes Johnson. *Computational Turbulent Incompressible Flow*, volume 4 of *Applied Mathematics: Body and Soul*. Springer, 2007.
8. Niclas Jansson. *High Performance Adaptive Finite Element Methods: With Applications in Aerodynamics*. PhD thesis, KTH Royal Institute of Technology, 2013.
9. Niclas Jansson. Optimizing Sparse Matrix Assembly in Finite Element Solvers with One-sided Communication. In *High Performance Computing for Computational Science – VECPAR 2012*, volume 7851 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013.
10. Niclas Jansson, Johan Hoffman, and Johan Jansson. Framework for Massively Parallel Adaptive Finite Element Computational Fluid Dynamics on Tetrahedral Meshes. *SIAM J. Sci. Comput.*, 34(1):C24–C41, 2012.
11. R. C. Kirby and A. Logg. A compiler for variational forms. *ACM Transactions on Mathematical Software*, 32(3):417–444, 2006.
12. Robert C Kirby. Algorithm 839: Fiat, a new paradigm for computing finite element basis functions. *ACM Transactions on Mathematical Software (TOMS)*, 2004.
13. Anders Logg, Kent-Andre Mardal, Garth N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.
14. Leonid Oliker. PLUM parallel load balancing for unstructured adaptive meshes. Technical Report RIACS-TR-98-01, RIACS, NASA Ames Research Center, 1998.
15. MC Rivara. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. *Int. J. Numer. Meth. Eng.*, 1997.