Computer Science • 14 (3) 2013 http://dx.doi.org/10.7494/csci.2013.14.3.513

Kazimierz Michalik
Krzysztof Banaś
Przemysław Płaszewski
Paweł Cybułka

MODULAR FEM FRAMEWORK "ModFEM" FOR GENERIC SCIENTIFIC PARALLEL SIMULATIONS

Abstract *We present the design for, and implementation of, a flexible and robust parallel modular finite element (FEM) framework called ModFEM. The design is based on reusable modules which use narrow and well-defined interfaces to cooperate. At the top of the architecture, there are problem-dependent modules. Problem--dependent modules can be additionally grouped together by "super-modules". The structure allows for reusing the sequential code for parallel environments, and also supports solving multi-physics and multi-scale problems.*

Keywords fem, modules, parallelism, grid, framework

1. Introduction

Mesh-based methods, including the finite element method (FEM), support the effective analysis of complex physical phenomena described by partial differential (PDE) equations over one-, two-, and three-dimensional domains. There are many cases when the use of unstructured meshes, with adaptation supported by appropriate error estimators, leads to the most reliable and efficient application of FEM. Besides the fact that, for three-dimensions, FEM with adaptive meshes can have considerably-smaller computational requirements than FEM with uniform mesh for the same level of accuracy, there are still problems so complex that they can only be solved within parallel computer environments. Development of flexible adaptive parallel FEM applications for distributed memory systems requires us to solve many problems. There are technologies that support the deployment of programs in such environments, and libraries addressing those requirements as well as complete FEM applications having this feature. In recent years, all three categories featured many proposals, articles and implementations for parallel adaptive FEM development problems.

The accuracy of the FEM solution is implied by **the chosen** weak formulation, approximation function space, and spatial (geometrical) discretization. Expression of the quality of a solution is done in terms of solution error. Error analysis of the finite element solution can be obtained in a mathematically-rigorous way only for some linear problems. Most real problems contain different non-linearities, and general error analysis becomes difficult or even impossible for them. Therefore, the error **analyses have** to take into consideration changing spatial and time discretization for hyperbolic and parabolic problems. It is necessary that error estimation be at least partially done in an iterative or incremental manner. This leads to further complications of establishing reliable error estimator techniques.

One of the most successful techniques for decreasing error (improving accuracy) of the solution is adaptivity. There are a few main directions of applying adaptivity during FEM computations: h-adaptation, p-adaptation, hp-adaptation, and r-adaptation. H-adaptation decreases error by increasing local density of degrees of freedom (DOFs), which are added. P-adaptation increases order of shape functions within element and, therefore, decreases error within this element. Hp-adaptation is a mixture of both h- and p-adaptation. R-adaptation decreases error by local improvement of spatial discretization without increasing the number of DOFs. Each kind of adaptivity brings about different problems and complications; thus, the successful deployment of adaptive FEM is not trivial. Apart from the different complications, all adaptive techniques usually share substantial demands for more resources, both in regards to memory and CPU time. This naturally leads to large-scale parallel computational environments.

Both parallel architectures and algorithms require the problem domain to be partitioned into set of smaller sub-domains, and the final solution is obtained by the appropriate combination of computed sub-tasks. Parallel environments can be considered as parallel environments with shared memory or parallel environments with

distributed memory. There are also hybrid architectures supporting shared and distributed memory **models** at the same time. All three grapple with several problems, including synchronization, simultaneous memory access, resource allocation, critical sections, and others. To solve these problems, several standards have been developed. Within distributed memory parallel environments, message passing concept is the de facto mandatory standard, and communication libraries provide implementations of Message Passing Interface (MPI). Implementations for shared memory environments commonly use OpenMP for handling multi-threading issues.

The developers often choose to fulfill severe demands of parallel environments using object-oriented programming paradigms. The object-oriented programming techniques provide the programmer with tools that allow them to reduce the complexity of code by applying concepts like dynamic dispatching, encapsulation, polymorphism, inheritance, delegation, self recursion using classes, instances, methods, abstraction, templates, and others. Since the late 90's, many articles on applying OO-techniques were published, and many FEM codes with object oriented architectures were also created. The flexibility of developed codes is often referred in terms of extending existing codes with new weak formulations, error estimators, approximation functions, adaptivity techniques, solvers, and pre-conditioners.

One such effort is the FENICS Project [2], which emphasizes component architecture with the common goal of enabling an automated solution of differential equations. The proposed components provide scientific computing tools for working with computational meshes, finite element variational formulations of ordinary and partial differential equations, and numerical linear algebra. The authors of the **FENICS** Project admit that achieving peak performance was not their purpose; but still, the **FENICS** Project takes advantage of many available HPC libraries, like MPI, PETSC, Trilinos, uBlas, UMFPACK, ParMetis, and others.

For the authors of Dune framework and the Dune-fem module [1], the leading design principle was a one-to-one correspondence between the mathematical objects within grid-based discretization schemes for stationary and non-stationary partial differential equations and C++ interface classes in Dune-fem. As they note in [1], thanks to advanced C++ programming techniques, experiments demonstrate both the efficiency and the applicability for a very large class of problems, including FEM PDE solutions.

A similar approach to avoiding computational overhead while still taking advantage of OO-programming is presented in [9]. The Deal.II authors assure us that the key to achieving this is a proper separation of concepts such as meshes, finite element spaces, and degrees of freedom, as well as the possibility to arbitrarily combine finite element spaces, numerical quadrature, and mapping information. The creators of Deal.II also find that it is important to use standard programming concepts and to avoid making frameworks overly general. They even point to the OOFEM [14], another FEM framework, as a negative example.

The authors of an Interactive Parallel Multigrid FEM Simulator [10] focus their work on delivering specific FEM-simulated solutions in a real time regime. Accor-

ding to published papers, the key trade-off in the Interactive Parallel Multigrid FEM Simulator is achieving accuracy at a sufficient speed. The proposed architecture is based on two main components: the simulation server and the graphical front-end that communicate through a client/server model. The authors assure us that the graphical front-end client, constituting a graphical user interface, is capable of obtaining the user input – user forces applied to different parts of the objects – and to display the current configuration of the deformable objects in real time. UI receives new configuration from the server whenever it is ready to draw a new frame, and it sends the current position of the user forces to the server. The authors are still working on the interactive system that will be able to simulate the interaction among multiple deformable objects. As they admit, this task requires "clever schemes of collision detection/response (CD/CR), which are computationally extensive" [10].

Domain decomposition and further dynamic load balancing **were** also the subject of research and articles. The Metis and ParMetis libraries [3] are, among others, the successful implementations of developed algorithms for partitioning and repartitioning of computational domain. Many FEM codes deploy them instead of their own implementations.

Incremental error estimators and automatic dynamic adaptivity within parallel environments cause considerable complication of finite element mesh management during simulations. Both articles covering those research areas as well as implementations were provided during the last years, e.g. [5]. Also, solutions for massively-parallel computations for tens and hundreds of thousands of cores were developed [4].

Rapid progress in hardware was made in recent years, as new computational platforms and technologies have been presented. This includes the GPGPU area with CUDA and OpenCL, allowing the use of hundreds of cores located on graphics cards. Also, hybrid architectures like Cell B.E. or CPUs with Intel Xeon Phi co-processors are offered as alternative to classic mutli-core processors.

1.1. Current contribution

The current paper contains the description of modular architecture of ModFEM framework. A description is provided at the abstract level of interface of modules [6] without going into the implementation details of each module. Modules are developed to achieve flexibility, including all of the areas mentioned in the introduction. The flexibility of ModFEM has also a specific aspect that is rarely taken into account in component and object oriented approaches. The proposed modular architecture takes into account the technology that lies below the OO-concepts. ModFEM modularity addresses not only FEM requirements, but is also aware of changing hardware technologies. The authors' goal was to create a research FEM framework whose flexibility goes beyond programming concepts and extends into the area of new computational architectures [8].

1.1.1. ModFEM overview

The name of the framework – ModFEM – stands for two things: first, an abbreviation of the Modular Finite Element Method, which highlights the main modular principle used in ModFEM; and second, an abbreviation from the Modifiable Finite Element Method, which indicates the main functional advantage of the framework.

ModFEM falls into the category of open-source scientific codes and libraries. It is a computational framework for generating a parallel adaptive finite element modelling applications. By its modular architecture, the framework allows the introduction of new mesh algorithms, approximation spaces, adaptation algorithms, or weak formulations.

The Moors' Law **show** clearly that computing power grows and will continue to grow further [16]. For large-scale applications, the dominant role falls to parallel distributed memory environments [17]. We address this requirement by introducing parallel overlay modules in ModFEM. The modules are capable of parallelizing sequential scientific codes without the need of a separate process of developing a distributed application.

Multi-scale and multi-physics modelling is also supported in ModFEM architecture. Modules are responsible for modelling a single physics phenomenon. The super-modules combine two or more problem modules, thus allowing the development of a strongly-coupled solution for complicated multi-physics phenomena. The same principle can be used for multi-scale modelling. Using a super-module, one can benefit by simulating a complicated problem by passing values directly between different scale levels. The wrapper modules also support interfacing with standalone FEM codes and applications using methods other than FEM (e.g. cellular automata).

2. Modular structure

The ModFEM framework provides a modular structure, with all interfaces defined and several **sample** modules already implemented. These modules are independent of each other. Each module can be easily exchanged with another module with the same functionality, as long as it properly implements the module interface.

The whole program is split into several (currently, less then ten) modules (Fig. 1). Modules interact only through strictly-defined interfaces. The interfaces have also been created to maintain the similarity to the classic FEM codes formulation:

1. Defining a *problem* and choosing FEM weak formulation.
2. Selecting appropriate *approximation* function space.
3. Selecting conforming geometric discretization with *mesh* type and properties.
4. Adjusting *solver* and its preconditioner.
5. Thus, you have a well-defined *application* for a well-defined problem.

All modules are implemented for cross platform usage and all mainstream **compiler** support. The process of application building is provided by cross platform com-

pilation and linking framework. The entire FEM-application logic has been hidden behind the seven orthogonal interfaces.

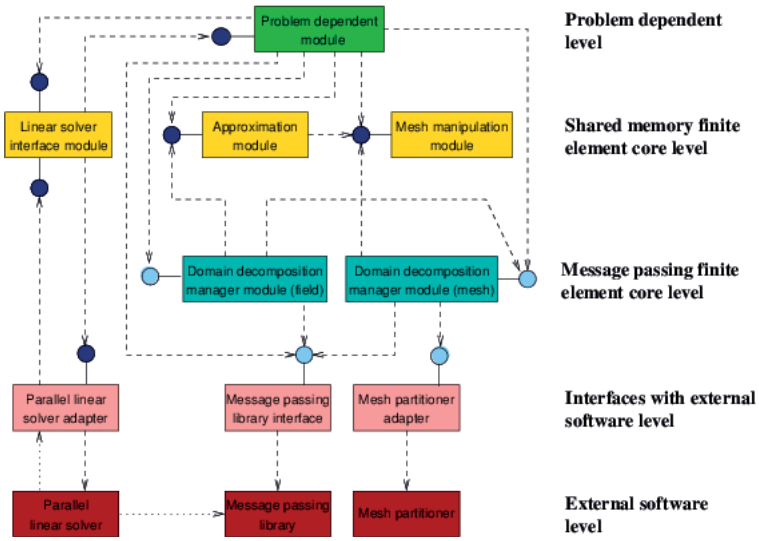

Figure 1. Modules diagram.

Problem module. The problem module is the most important module. It contains a weak formulation of FEM for some **preselected** problem. All other modules are used by the problem module. The link between this module and other modules is not always direct. The problem module defines all procedures associated with the weak formulation of the problem. Many of them are callback-routines called from other modules when necessary.

Mesh manipulation module. The mesh module is the most-basic unit. It does not depend on any other module. It determines some of the most important features of FEM approximation [12]. This module is responsible for all operations directly related to the geometric discretization of the computational domain. All mesh entities are managed by this module. It provides iterators which allow access to parts of the mesh. The inner logic of this module is also responsible for issues related to the orientation of the elements, faces, and edges. Mesh adaptation and de-adaptation routines form a very important part of the module. In addition, a detailed geometry sub-module is included inside this module. The geometry sub-module allows for improvement of boundary geometry during h-adaptation. Finally, mesh module implements procedures for validating a geometric mesh during input/output operations, and also supports the identification and correct location of the boundary conditions.

Approximation module. This module implements methods associated with the selected function space. FEM discretization requires determining the space of func-

tions used in approximation. Shape functions associated with elements, faces, edges, and vertices are defined inside this module. Also, the calculation of the values of shape functions at a given point and their derivatives, integration over elements, faces, and edges is implemented in this module. The approximation module also includes a transformation between local and global coordinate systems.

Solver interface module. The module is responsible for interfacing with external modules for solving systems of linear equations. However, all of the FEM weak formulation and problem specific routines (required by the solvers) are delegated to the appropriate FEM modules. This tiered approach allows for the separation of problem details and selected solver strategy. Through that, the solver interface module is generic and independent from the problem being solved.

Parallel mesh overlay. Parallel mesh module is an overlay for any implementation of the mesh module. The module extends the functionality of the mesh manipulation module. It adds functions responsible for domain decomposition, load balancing, the transfer mesh data between computational nodes, parallel adaptation, inter-sub-domain boundary conditions, and many others. It also expands the mesh entities identifiers, assigning them an unambiguous owner to provide unique global identifiers. The module can handle meshes partitioned into subdomains with overlaps having arbitrary widths.

Parallel approximation overlay. The parallel approximation module overlay extends the approximation module. It adds functions responsible for exchanging **DOFS** data between computational nodes, synchronizing **DOFS** in overlay managed by parallel mesh module, computing global (whole domain level) vector products, and transferring some additional information in the parallel execution environment.

3. Two levels of parallelism

A generic approach to distributed computing involves two levels of parallelism, which are exploited in the framework [7]. The first, global level of parallelism with distributed memory and message passing, is implemented using parallel overlays for mesh and approximation modules. It handles domain decomposition, load balance, synchronization of mesh changes, etc. At the second level of parallelism, we focus on a single computational node with shared memory. Both levels support multiple meshes. At the global level, **the parallel overly module decomposes** mesh into sub-meshes which can be merged, split, or changed. At the computational node level, there can be distinct meshes for different fields (for multi-physics problems), or several approximation fields can share the same mesh. This hybrid model is already well-known, especially in applications using MPI along with OpenMP. The novelty of our approach is to use general-case interfaces for parallelism. Such an approach allows us to use different combinations of technologies, e.g., combining MPI with CUDA GPGPU kernels, switching from OpenMP to OpenCL, or some new emerging technologies.

The usage of the code for the two levels can be further described as follows:

Global level parallel system. This is a distributed memory model covering the whole computational domain. This level makes usage of pre-built parallel overlays. At this level, we focus on domain decomposition and load balancing problems. The mesh and approximation module parallel overlays are able to handle many meshes partitioned into sub-meshes, provided by domain decomposition algorithms. Our code also supports global id, load balancing, splitting and merging meshes, adding, and removing single entities.

Local level a computational node. This is a shared memory model for multi-threaded execution environment. At the node level, there is the possibility of handling more than one mesh simultaneously. For example, when working on problems of fluid-structure interaction, one mesh could be used for fluid simulation and the other for structure. We can also use different approximation fields on the same geometrical mesh thanks to the distinction between the geometry and the solution degrees of freedom (managed by a separate approximation module). Implementation handles critical situations through a checkpoint-restart mechanism.

4. FEM capabilities

It should be noted that, in ModFEM, problem parameters are defined in several input files. Input files cover meshes, fields, parameters, materials, boundary conditions, solver parameters, output format, etc. Some commonly-used file formats are already supported. **The input-output scheme is open for extension. Both text and binary file formats are supported.**

Authors of the ModFEM framework provide at least one working implementation for each of the seven modules previously described. At this moment, one can choose from the implemented modules listed below.

4.1. Available modules

Generic convection-diffusion module This is a base generic problem module. It is designed to be a basis for all convection-diffusion phenomena simulations. It is also used for testing modules of other types (mesh, approximation, parallel overlays).

Heat transfer module The provided problem module is capable of solving heat transfer problems. Apart from allowing the use of almost arbitrary material coefficients, it also takes into account phase transitions and different boundary conditions (including convection and radiation).

Elasticity module The provided problem module is capable of solving elasticity equations in solid materials. One main characteristic of the module is the ability to estimate errors and adapt meshes for **optimal problem-solving**.

Incompressible fluid-flow module The provided problem module is capable of solving incompressible fluid flows using Navier-Stokes equations with well established SUPG-type stabilization.

Fluid-flow with heat transfer module This problem super-module can be used for coupled fluid flow – heat transfer problems. For this purpose, two problem modules – fluid flow and heat transfer – are coupled together. This shows that interfaces of problem module procedures are specially designed to allow for such multi-physics coupling.

Fluid-structure interaction module This super-module can be used for standalone fluid flow simulations with moving boundaries as well as for coupled fluid-flow with fluid-structure interaction. From a variety of methods, the ALE formulation has been implemented to address this requirement.

Prismatic mesh module with h-adaptation This module contains a fast prismatic mesh module with advanced adaptation schemes, including hanging nodes. **It handles** many meshes at the same time for coupled problems.

Hybrid tetrahedral/prismatic mesh module with h-adaptation This memory-efficient prismatic, tetrahedral, and hybrid hierarchical mesh module is implemented with advanced adaptation schemes, including hanging nodes, anisotropic adaptation, and many others. It handles many meshes at the same time for coupled problems.

Tetrahedral mesh module with re-meshing and mesh improvement This mesh module contains implementation of advanced mesh generation algorithms, including generating thin prism layers for detailed fluid boundary layers. It is capable of mesh improvement, mesh r-adaptation, mesh regeneration, mesh tracking, and mesh movement during computations. It also handles many meshes at the same time for coupled problems.

Standard linear approximation module with hanging nodes This module implements standard linear approximation with constraint nodes. It also supports mesh adaptation, and can handle many scalar and vector fields at the same time.

Discontinuous Galerkin approximation module This module uses Discontinuous Galerkin approximation (currently it is used for linear convection-diffusion problems). It supports mesh adaptation and also handles many scalar and vector fields.

Generic parallel mesh overlay module This generic module is compatible with all mesh modules implementing interface routines. It supports any sequential mesh module. This module allows sequential meshes to be effectively used in the distributed memory environment (e.g. grids).

Generic parallel approximation overlay module This generic module is compatible will all approximation modules implementing interface routines. It also supports any sequential approximation module and allows it to be effectively utilized in distributed memory environments.

MPI-based parallel communication library This module supplies parallel overlays with communication infrastructure. For effective internal communication, it uses the MPI standard.

Krylow space block iterative solver module This module includes implementation of Krylow space iterative methods with several preconditioners implemented.

PARDISO solver interface module This module interfaces with the PARDISO direct solver.

Regardless of the available modules listed, the framework successfully cooperates with external software. As a result, coupled simulations with solid state materials have been carried out [13].

5. Examples and results

5.1. Examples

The framework has been tested for a variety of example problems. Since the subject of the current paper is not to describe particular problem, mesh, or approximation modules and their characteristics, we do not focus on the obtained results. Rather, we show results as proof of concept for the idea of creating applications from separately developed modules.

5.1.1. Incopressible fluid flow

Using the ModFEM framework, the application for solving incompressible fluid flow was generated. As the main problem module, the Navier-Stokes equations module was used. Two versions of the application were automatically generated by CMake: first, using the Intel Compiler in the Linux environment; and second, using the Microsoft Visual Studio Compiler in the Windows environment. Both were successfully built with support for the same input and output files. The modules used for the first one were as follows:

- Incompressible fluid-flow module
- Standard linear approximation module with hanging nodes
- Prismatic mesh module with h-adaptation
- PARDISO solver interface module

The second application used:

- Incompressible fluid-flow module
- Standard linear approximation module with hanging nodes
- Hybrid tetrahedral/prismatic mesh module with h-adaptation
- Krylow space block iterative solver module

Figures 2, 3 and 4 show the application of the two codes for solving classic benchmark problems of incompressible fluid flow. The next two Figures, 5 and 6, show results obtained by the code for the simulation of blood flow in artificial heart chambers within the Polish Artificial Heart project.

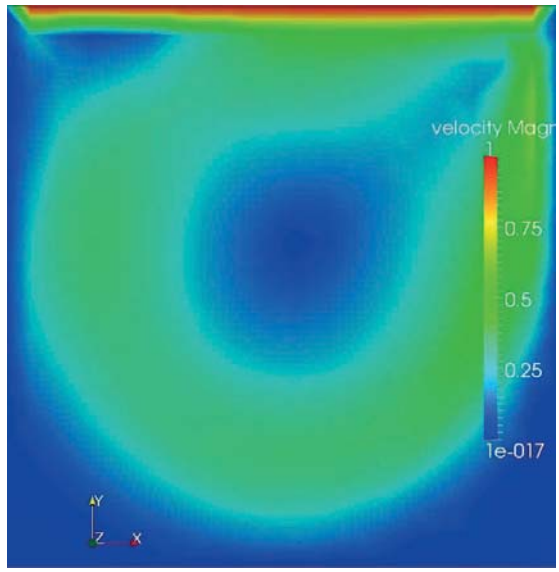

Figure 2. Lid Driven Cavity.

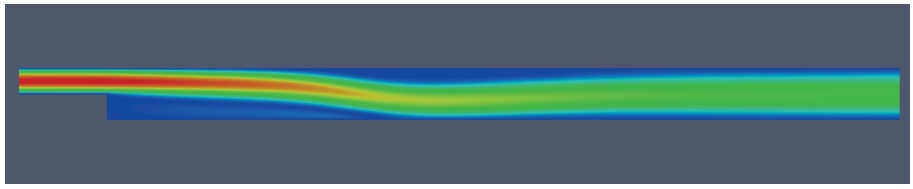

Figure 3. Backward Facing Step.

5.1.2. Incompressible fluid flow with heat transfer

As a "proof of concept" of super-modules using the ModFEM framework, the application for solving incompressible fluid flow with heat transfer was generated. As the problem modules, the Navier-Stokes equations module and the Heat transfer module were used in combination. CMake generated Makefiles for the Intel Compiler in the Linux environment. The selected modules were:

- Fluid-flow with heat transfer super-module.
- Heat transfer module.
- Incompressible fluid-flow module.
- Standard linear approximation module with hanging nodes.
- Hybrid tetrahedral/prismatic mesh module with h-adaptation.
- PARDISO solver interface module.

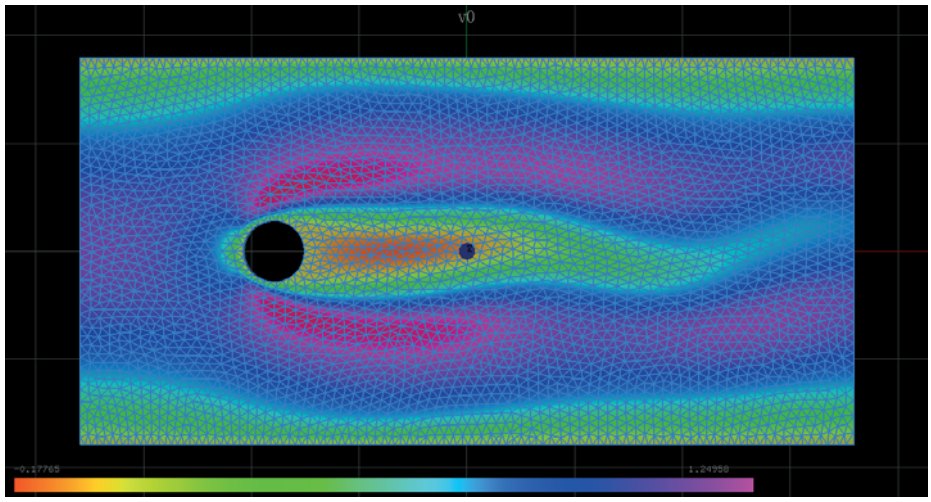

Figure 4. Von Karman vertex.

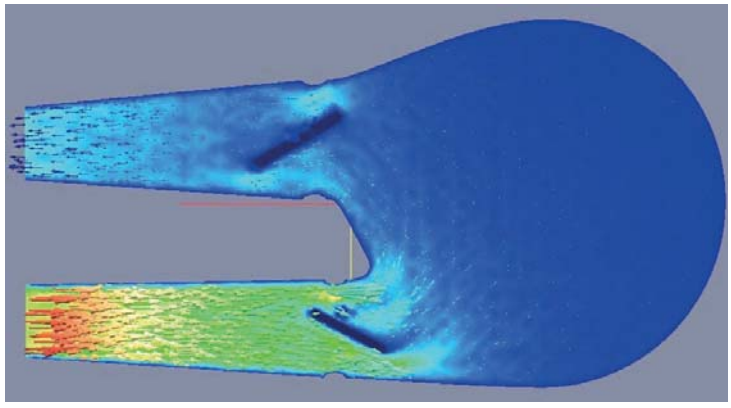

Figure 5. Artificial heart chamber blood flow.

Figure 7 shows the results obtained by the application of the code for simple heat-induced flow in a cube. A more-complex example is shown in Fig. 8, where a welding pool created during the welding process is simulated by the code.

6. Conclusion and further work

ModFEM framework is created by scientists for scientists. It offers a ready-to-use set of FEM modules with a robust tool chain to create **your own** FEM software. Support for both small workstations and large distributed-memory environments, like grids, is provided. ModFEM offers a promising and desirable opportunity to reuse and improve

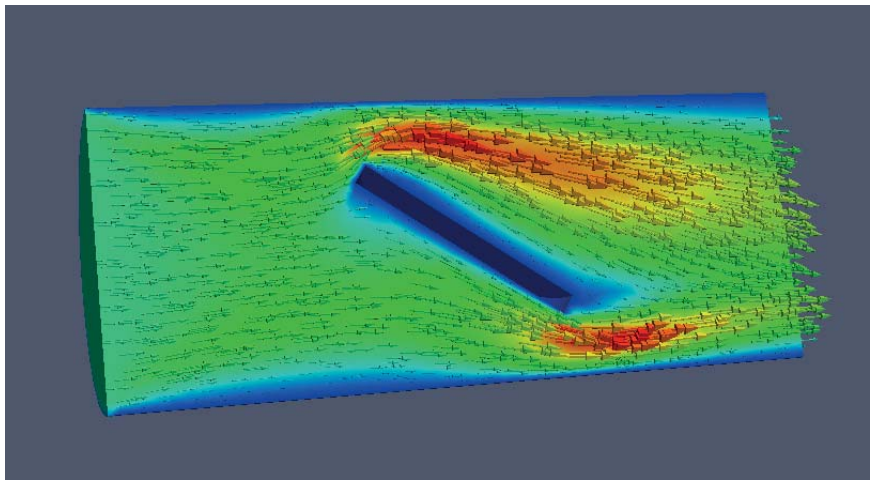

Figure 6. Rotation of heart valve during blood flow.

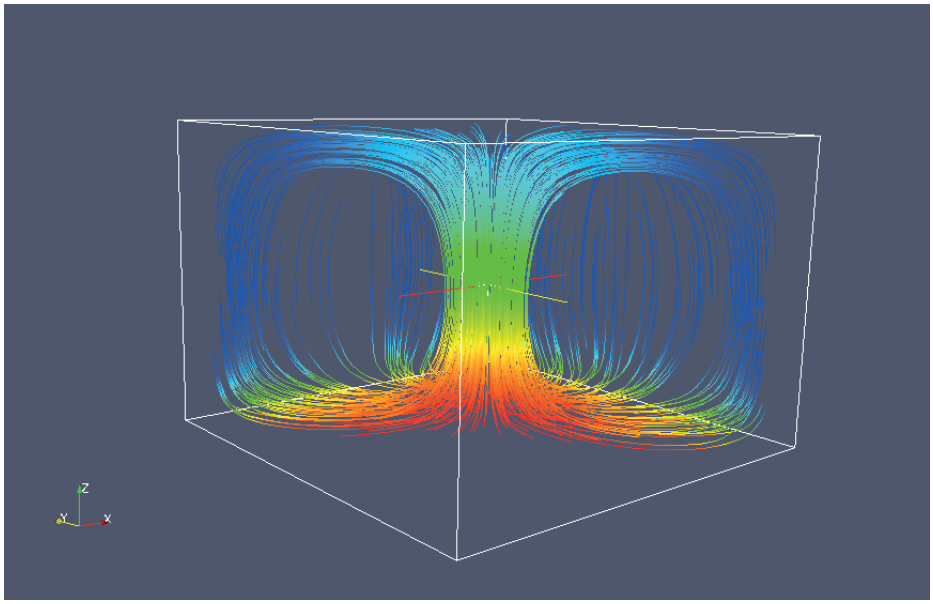

Figure 7. Bottom heating.

existing scientific applications. It strongly supports focusing on an area of interest, avoiding development of the whole FEM code.

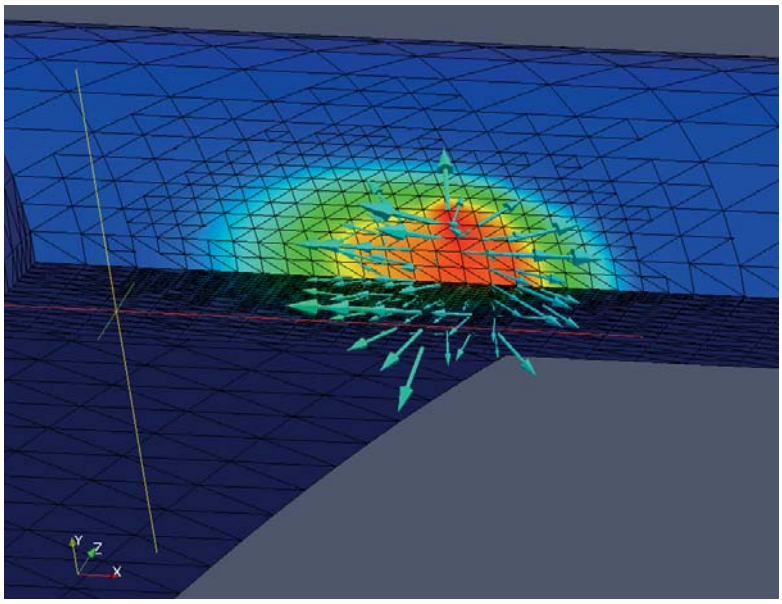

Figure 8. Plasma welding pool.

Still, we are working on improving parallel environment execution, extending the set of problem modules, and adding new approximation modules. Modules currently under development are:

- Fluid-structure interaction with heat transfer module [15].
- Elasto-plasticity module.
- Elasto-plasticity with heat transfer module.
- OpenCL shared memory approximation overlay module.
- Discontinuous Petrov-Galerkin approximation module.
- Multi-grid solver module.

The authors believe that the practical application of the idea of code modularity is quite beneficial for the scientific process, **without exception** of the field of FEM modelling, especially including supporting changing technologies for new parallel and hybrid architectures.

Acknowledgements

This research has been partially supported by the European Regional Development Fund program no. POIG.02.03.00-00-096/10 as part of the PL-Grid PLUS project and Polish National Science Centre under grant DEC-2011/01/B/ST6/00674.

References

[1] Dedner A., Klöfkorn M. N. M. O.: A general object oriented framework for discretizing nonlinear evolution equations. *Proc. of The 1st Kazakh-German Advanced Research Workshop on Computational Science and High Performance Computing*, 2005.

[2] Anders Logg, Kent-Andre Mardal G. W.: Automated solution of differential equations by the finite element method. *Lecture Notes in Computational Science and Engineering* vol. 84, 2012.

[3] LaSalle D., Karypis G.: Multi-Threaded Graph Partitioning. *27th IEEE International Parallel & Distributed Processing Symposium*, 2013.

[4] Shephard M. S., Seol S.: Flexible Distributed Mesh Data Structure for Parallel Adaptive Analysis. *Advanced Computational Infrastructures for Parallel and Distributed Adaptive Applications*, chapter 19, Wiley 2009.

[5] Oliker L., Biswas R., Gabow H. N.: Parallel tetrahedral mesh adaptation with dynamic load balancing *Parallel Computing*, 26: 1583–1608, 1999

[6] Banaś K.: A model for parallel adaptive finite element software. *Domain Decomposition Methods in Science and Engineering*, Vol. 40 of Lecture Notes in Computational Science and Engineering, pp. 159–166, 2004.

[7] Banaś K.: A modular design for parallel adaptive finite element computational kernels. *Computational Science* – ICCS 2004, 4th International Conference, Krakow, Poland, June 2004, Proc., Part II, vol. 3037 of Lecture Notes in Computer Science, pp. 155–162, 2004.

[8] Banaś K.: Parallelization of large scale adaptive finite element computations. Parallel Processing and Applied Mathematics, *Proc. of Vth International Conference*, PPAM 2003, Czestochowa, Poland, vol. 3019, pp. 431–438, 2004.

[9] Bangerth W., Hartmann R., Kanschat G.: deal.II – A general purpose object--oriented finite element library. *ACM Trans. Math. Softw.*, 33(4): 24/1–24/27, 2007.

[10] Jaein Jeong T. G., Wu X.: An interactive parallel multigrid fem simulator. *Medical Simulation*, Vol. 3078 of Lecture Notes in Computational Science, pp. 124–133, 2004.

[11] Banaś K., et al.: Towards using adaptive hybrid meshes in FEM simulations of flow in artificial heart chambers. *Procedia Computer Science*; ISSN 1877-0509. vol. 1, pp. 2037–2045, 2011.

[12] Banaś K., Michalik K.: Design and development of an adaptive mesh manipulation module for detailed FEM simulation of flows. *Procedia Computer Science*; ISSN 1877-0509. vol. 1, pp. 2037–2045, 2010.

[13] Kopernik A. M.: Two-scale finite element model of multilayer blood chamber of polvad-ext. *Archives of Civil and Mechanical Engineering* / Polish Academy of Sciences. Wrocław Branch, Wrocław University of Technology; ISSN 1644-9665, pp. 178–185, 2010.

[14] Patzák B., Rypl D.: Object-oriented, parallel finite element framework with dynamic load balancing. *Advances in Engineering Software* vol. 47, pp. 35–50, 2012.

[15] Cybułka P., et al.: Simulation of droplet motion in welding arcs as a case study of remeshing. *Computer Methods in Materials Science* : quarterly / Akademia Górniczo-Hutnicza ; ISSN 1641-8581, page 381–386, 2011.

[16] Resch M. M.: Sustained Simulation Performance 2012. *Proc. of the joint Workshop on High Performance Computing on Vector Systems*, Stuttgart (HLRS), and Workshop on Sustained Simulation Performance, Tohoku University. Springer Berlin Heidelberg, 2012.

[17] Selim G., Akl M. N.: *Parallel Computing. The Future of Parallel Computation.* Springer London, 2009.

Affiliations

Kazimierz Michalik
ACC Cyfronet AGH, ul. Nawojki 11, 30-950 Kraków 61, P.O.Box 386, kamich@agh.edu.pl

Krzysztof Banaś
AGH University of Science and Technology, Krakow, kbanas@pk.edu.pl

Przemysław Płaszewski
AGH University of Science and Technology, Krakow, pplaszew@agh.edu.pl

Paweł Cybułka
AGH University of Science and Technology, Krakow, cybulka@agh.edu.pl

Received: 11.11.2012
Revised: 19.12.2012
Accepted: 26.04.2013