Interdisciplinary Practical Course on Parallel Finite Element Method Using HiFlow^3

Markus Hoffmann[1](✉), Simon Gawlok[2], Eva Treiber[2], Wolfgang Karl[1], and Vincent Heuveline[2]

[1] Institute of Computer Science & Engineering (ITEC), Chair for Computer Architecture and Parallel Processing (CAPP), Karlsruhe Institute of Technology (KIT), Kaiserstrasse 12, 76131 Karlsruhe, Germany
markus.hoffmann@kit.edu

[2] Interdisciplinary Center for Scientific Computing (IWR), Engineering Mathematics and Computing Lab (EMCL), Heidelberg University, Speyerer Strasse 6, 69115 Heidelberg, Germany

Abstract. In many scientific fields one faces partial differential equations that have to be solved numerically. Applying the widely-used finite element method (FEM) leads to huge systems of equations whose solutions often require parallel computing. The practical course presented in this paper aims at introducing the FEM as well as the concept of parallel computing to students with the help of a FEM library, in this case HiFlow^3. To achieve this goal, the students work in interdisciplinary groups on explicit problems originating from different scientific fields. In that way they expand and deepen both their theoretical knowledge concerning numerical mathematics and their practical skills in programming and using HiFlow^3.

1 Introduction

In many fields of science as well as in industrial research and development solving problems both fast and accuratly is often a tough challenge. While on one hand there is a complex theoretical approach to solve a problem, there are, on the other hand, scientists and engineers who want to tackle their specific problems with the considered methodology. They are often experts in their specific areas of knowledge which differ from the skills needed to approach the problem at hand. Additionally, applying a theory or solution method to a problem does not mean that the resulting software application is in any way automatically optimal with respect to the utilized hardware and therefore also with respect to the appropriate programming paradigms, scalability requirements or models of parallelization.

These three challenges,

- facing the theory,
- applying it to a specific problem and
- developing a parallel application based on the features of the available hardware,

S. Hunold et al. (Eds.): Euro-Par 2015 Workshops, LNCS 9523, pp. 160–171, 2015.
DOI: 10.1007/978-3-319-27308-2_14

are forming the grander challenge of parallel scientific computing as shown in Fig. 1:

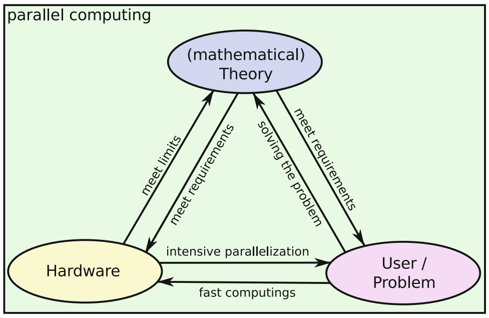

Fig. 1. Challenges of parallel computing

While the compliance with the theoretical requirements of a solution method must be assured throughout the whole process of its application to the specific problem by the user, the choice of a methodology may be limited by the hardware and his' or her's knowledge of programming the hardware platform at hand. Additionally, he or she expects fast results on modern hardware which leads to the need of intense – maybe even hybrid – parallelization which in turn restricts the choice of usable algorithms or even compels a redesign of established solution methods.

This interaction between the given problem, applicable theoretical solution approach, and available hardware, as shown in Fig. 2, challenges the user immensely. Therefore, it is necessary to educate the today's students as early and as extensively as possible to provide them with the knowledge and abilities to face the challenge of parallel computing.

Obviously, it is impossible to teach a single user all the required knowledge about mathematical theory, its application to problems in all fields of science, and all the models, paradigms and requirements of parallel computing on any imaginable combination of hardware. Therefore, an expansion of the challenge of parallel computing to the challenge of parallel computing *within an interdisciplinary team of specialists* is unavoidable. This change in philosophy is particularly demanding for the teaching staff, because, beside the huge amount of knowledge they are required to have in order to teach, they need to find a common base on which a profound education can start.

Fortunately, this common base can be found for many fields of natural sciences in terms of partial differential equations. Partial differential equations (PDE) arise in many fields such as meteorology, biology, or energy research. A well-known and widely used method to numerically solve these equations is the so called finite element method (FEM).

FEM is based on a weak formulation of the PDE: By multiplying the partial differential equation with an arbitrary test-function, integrating the resulting equation by parts, and considering the boundary conditions, one achieves in the case of a linear PDE a variational formulation of the problem with the form:

> Find a function u in an appropriate function space such that
>
> $$a(u, v) = l(v) \tag{1}$$
>
> for all test-functions v in a possibly different appropriate function space.

The bilinear form a and the linear functional l depend on the exact problem, as do the function spaces. Due to the fact that these function spaces ease the restrictions on u compared to those imposed on the function u by the original, strong formulation of the PDE, the variational formulation is also called weak formulation. In the next step the function spaces are reduced to finite-dimensional ones to be able to explicitly state a complete basis of the spaces and express u and v as a linear combination of the basis functions. As a result, the above variational formulation yields a finite system of equations that only contains the unknown coefficients of the representation of u. By geometrically dividing the domain of the PDE into small elements and choosing piecewise polynomial functions and basis functions that vanish on almost every element of the mesh, most of the entries of the resulting matrix are zero. This way the task to solve a partial differential equation is converted to the task of solving a linear system of equations with a sparse matrix. Generally, the computation of this matrix is done by calculating the contribution of one element after another and summing them up.

Obviously, the above described construction allows problems with arbitrary many degrees of freedom. Indeed, this is not only a theoretical possibility, but many applications require very fine grids to achieve reasonable results. To solve the huge resulting discrete problems in an acceptable amount of time, parallel computing is unavoidable.

For that reason, students should become familiar with the concepts and methods of parallel computing as early as possible. Furthermore, practical exercises and courses offer a great opportunity to the students to apply their theoretical knowledge in practice, so they can reinforce what they learned in lectures and get deeper insights into the various aspects of being active in the field of numerical mathematics in general, and parallel computing in particular.

A smart introduction into this topic and its handling is offered by different libraries, such as HiFlow^3, for which in particular only a relatively short familiarization period is required. As, additionally, the underlying mathematics can still be seen clearly in the problem-specific part of the code, HiFlow^3 was selected for practical exercises and courses that shall be presented in this paper. Therefore, the next Section aims at introducing HiFlow^3 with its modular approach and provided tools, whereas the ideas and settings of the practical course are centred in Sect. 3. Finally, Sect. 4 offers future prospects regarding this software practical.

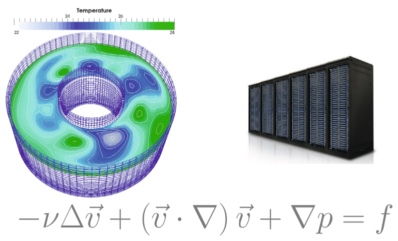
Temperature
$-\nu\Delta\vec{v} + (\vec{v}\cdot\nabla)\,\vec{v} + \nabla p = f$

Fig. 2. Connection of user's perspective, theory and target hardware [6]

2 HiFlow^3

HiFlow^3 [4] is a multi-purpose finite element software developed at EMCL providing powerful tools for the efficient and accurate solution of a wide range of problems modelled by partial differential equations (PDEs). Based on object-oriented concepts and the full capabilities of C++, the HiFlow^3 project follows a modular and generic approach for building efficient parallel numerical solvers. It provides highly capable modules dealing with the mesh setup, finite element spaces, degrees of freedom, linear algebra routines, numerical solvers, and output data for visualization, See Fig. 3. Parallelism — as the basis for high performance simulations on modern computing systems — is introduced on two levels: coarse-grained parallelism by means of distributed grids and distributed data structures, and fine-grained parallelism by means of platform-optimized linear algebra back-ends. The required communication within the distributed grids and distributed data structures is implemented with the aid of the Message-Passing Interface (MPI) standard [10]. Furthermore, the capabilities of different modules can be improved by compiling HiFlow^3 with support for certain third-party libraries, e.g., METIS [8] or Intel Math Kernel Library (MKL) Fig. 2.

The mesh module provides functionality that is needed for dealing with unstructured computational grids in finite element simulations. A mesh may consist of lines, triangles, quads, tetrahedrons, and hexahedrons. Independently, how the initial mesh has been obtained, it can be further refined through different refining strategies to increase the grid's resolution. If the mesh has been constructed or read in sequentially by a single process, the mesh can be partitioned and distributed for coarse-grained distributed memory parallelization. To reduce communication costs during the following computations in the simulation process, a layer of ghost cells is attached to each process' local mesh by a provided helper function Fig. 3.

The finite element method (FEM) and degree of freedom (DoF) modules supply the user with functionalities to construct a finite dimensional function space based on Lagrange finite elements. The user can specify the kind of finite element method, i.e., *h* or *hp* version, that is desired for the simulation as well as the degrees of the trial functions of the finite element basis for different solution components, e.g., velocity components and pressure in a computational fluid dynamics (CFD) computation. In the case of *hp* FEM, the degrees can be

Fig. 3. HiFlow^3: Mesh handling, FEM, linear algebra & solvers and applications [6]

specified cell-wise. Furthermore, Gaussian quadrature rules of different orders are provided for all supported element geometries and the user can choose the desired order of accuracy for the assembly process.

HiFlow^3 offers assembly strategies for matrices and vectors where the integrals may be defined on elements as well as the boundary of the computational domain. This process can be conducted fully in parallel on the distributed memory parallelization level due to the distributed grids and distributed matrix and vector structures, respectively.

The linear algebra module consists of different implementations of matrices and vectors on a both global, distributed-memory level, and local, per-process level. On the global level, HiFlow^3 provides a standard implementation, which takes care of the required communication as well as the correct handling and management of the local matrices and vectors in each process. A wrapper to the PETSc library [1–3] is currently under development. On the local level, HiFlow^3 establishes a class hierarchy where different computing devices and implementations on these devices can be configured by the user by simply changing the configuration parameters for the local linear algebra directly in the application code or even in an XML configuration file. This design allows fine-grained parallelism on a shared-memory, i.e., intra-process level. Currently, classical Central Processing Units (CPU) and Graphics Processing Units (GPU) are supported as computing devices. For GPUs, implementations facilitating NVidia CUDA [14] and OpenCL [17] are available. On the CPU side, the following implementations and libraries are supported: a naive, i.e., sequential, and an OpenMP parallelized implementation, as well as wrappers to the BLAS [12], CLAPACK [13] and Intel MKL libraries. Furthermore, the local matrices support a variety of matrix formats, e.g., dense and compressed row (CSR) format. It is important to note that not all matrix formats are available with all implementations.

To solve the arising linear equation systems efficiently, HiFlow^3 provides solver classes for the following iterative Krylov subspace methods [15]: conjugate gradient (CG), stabilized bi-conjugate gradient (BiCGSTAB), and (flexible) generalized minimal residual ((F)GMRES). All these solvers are implemented for the use with and without preconditioning techniques, and all solvers are parallelized by means of the parallelized linear algebra, i.e., in the implementation of the solvers no further parallelization needs to be employed. Preconditioning in HiFlow^3 is done block-wise, i.e., the preconditioner only acts locally on the degrees of freedom which are owned by the respective process. This way no communication is done during the preconditioning step. A disadvantage of this methodology is the lacking parallel scalability for very large numbers of parallel processes. Among the local preconditioners are, e.g., Jacobi, Gauß-Seidel, Approximate Inverse, and Incomplete LU (ILU) factorization methods. The capabilities of the linear solver module are extended by wrappers to the MUMPS [11] and ILU++ [9] libraries, respectively.

If the underlying PDE is nonlinear, HiFlow^3 provides an implementation of Newton's method for the solution of the resulting discrete system of equations. Due to the object-oriented approach, any of the above mentioned linear solvers can be used to solve the arising linearized system.

To visualize and rate the results, HiFlow^3 comes with support for visualization output in the (P)VTK [16,18] and XDMF [19] file formats.

Besides these core functionalities for the conduction of finite element simulations, the software package HiFlow^3 offers relevant advantages to the numerical simulation of phenomena occuring in a wide range of research topics, e.g. uncertainty quantification (UQ), computational fluid dynamics (CFD) & meteorology, and elasticity simulations.

The capabilities of HiFlow^3 with respect to performance and scalability on HPC systems have been demonstrated by considering a CFD benchmark problem [5].

The functionalities of HiFlow^3 are documented and demonstrated in several tutorials that are available on the HiFlow^3 homepage [6]. The complexity in terms of mathematics and implementational effort by the user of the tutorials ranges from relatively simple problems like Poisson's equation, which is a prototype of an elliptic linear PDE, to complex problems like solving the incompressible Navier-Stokes equations of fluid motion, or the equations of linear elasticity for soft tissue simulation, for example. All together, there are eleven tutorials available at the moment. Furthermore, HiFlow^3 comes with additional examples of applications which are not documented by a tutorial.

3 Practical Course on Parallel Numerics

Based on HiFlow^3, the Karlsruhe Institute of Technology started a practical course on parallel numerics that aims both at the interdisciplinary cooperation of the students and at teaching methods of parallel computing. At Heidelberg University, this course is embedded in the exercise classes of the lecture "Numerical Methods of Continuum Mechanics" given by the Engineering Mathematics and Computing Lab (EMCL). The course is subdivided into separate projects which are designed to be worked on by small groups of students.

To promote the desired interdisciplinary cooperation, the course is open to students of various departments like mathematics, computer science, physics and many kinds of engineering, which leads to a highly heterogeneous mixture of participants. The schedule itself supports the formation of heterogeneous groups due to the fact that specialized knowledge is required to solve every task of each project suitably. As mentioned in the introduction, this composition of heterogeneous teams is important due to the fact that the amount of needed knowledge is too large to be known by students out of the same field of study. Therefore, each group member has to take on a different role:

- The mathematicians have to face the mathematical theory of the PDE as well as the chosen solution methods in detail and have to make sure that all assumptions are met.
- The computer scientists have to deal with the challenge of parallelization based on both the available hardware and their knowledge about programming paradigms and methods of parallelization.
- The natural scientists have to check the validity of both the used models and the results. Furthermore, they have to instruct the others about the details of the particular problem.

Additionally, all the group members together have to face the tasks, that are beyond their fields of study at large.

Due to the different research interests, skills, and objectives of the individual participants of the course, it is tough to find a basic problem all students can work on with equal effort. The basic problem of choice has to provide a strong focus on practical use and applications for as many kinds of engineering as possible as well as a complex background in theory to challenge the mathematicians. Moreover, the theory has to lead to computationally intensive algorithms, to guide the computer scientists into the field of soft- and hardware-architectures, especially with the focus on parallel computing.

As shown in Sect. 1, the finite element method meets these conditions in an almost optimal way. As a standard approach for solving partial differential equations it is highly relevant for the practical use in most engineering fields. The theoretical background is challenging, particularly in the construction of appropriate solution spaces and integration of boundary conditions, the iterative solvers for the systems of equations and dealing with stability problems, just to name a few examples. Furthermore, solving the huge systems of equations does not only need experts in numerical analysis but also specialists in parallelization techniques like domain decomposition and task scheduling as well as skills in programming with these different parallelization paradigms, e.g., with the Message Passing Interface (MPI) and OpenMP standards. Finally, the FEM is addressing a wide range of problems. Therefore, simple problems that help the students to understand the basic theories can be found as well as very complex problems that challenge the students and to show them the limits in every aspect. For these reasons, the FEM is the basic syllabus in this practical course.

Furthermore, the usage of HiFlow^3 offers an efficient and fast access to the practical use of the FEM. Through the execution of given tasks, students are given the opportunity to see various theoretical results, for instance the different convergence behaviours of *h*- and *hp*-method, within practical results achieved by self-performed experiments. With the help of this library it is even easier to access complex problems. The incompressible Navier-Stokes equations for example, which need Taylor-Hood finite elements to be solved certainly, can be equipped with these elements through simple changes to an XML configuration file. Therefore, with the help of Hiflow^3, the focus of the curriculum is on discovering the parallel computing and not on the depth of mathematical theory. To be more specific: The students are taught the basics of FEM, i.e., required definitions and basic theorems, such as Lax-Milgram for example. They are also taught parallelization in theory, data reordering strategies or simple domain decomposition methods for example, but because HiFlow^3 is hiding a lot of its parallelism, the practical work on parallelization is limited to some simple exercises in the context of OpenMP. Therefore, the focus is on modeling and theoretical debates on parallelism as well as using the provided library and understanding its limitations when it comes to massively parallel computing.

Based on the usage of the FEM and HiFlow^3, one can define the learning objectives of this practical course. These objectives can vary for each student due to the interdisciplinarity. Ignoring this specialization-based weighting, the learning objectives can be defined as follows:

- *Working in interdisciplinary teams* as described in this section to get an impression of the advantages and disadvantages of this conception.
- *Understanding the basics of PDEs and the FEM* to avoid mistakes based on the lack of a theoretical background and the limits of the used method.
- *Understanding the needs of parallel computing* such as handling data dependencies or understanding the challenge of scalability.
- *Learning the usage of an example finite element method library*, which is Hiflow^3 in our case. Furthermore, it is important to know the advantages and the limits of the library.
- *Application of the FEM to an explicit problem* to learn the usage of the theory on a real problem and to increase the understanding of the method's properties.
- *Application of basic methods of parallel computing* with the focus on rearrangement of the data and simple domain decomposition methods.
- *Evaluation of results* with the help of suitable visualizations.

These objectives in total are giving an overall view of the theory and the challenges of parallel computing as well as working in an interdisciplinary team.

Teaching the students these learning objectives is a challenge we are facing with a strategy of different teaching methods. These methods are chosen in the context of the different backgrounds of the students as well as the fact, that – except for of basic programming skills – no special prerequisites are necessary to visit this practical course, which is a concession to the interdisciplinarity.

First of all, a specific number of classes are marked as theory lectures to provide the opportunity to teach the students the required knowledge within a short time. These lectures are given by way of a presentation with the option to ask questions. Naturally, a compulsory attendance for the students has to be set.

Beside these theory lectures and some other exceptions, all the sessions of this period are practical classes, in which the different groups of students are working on projects independently. During these practical classes they are guided by exercise sheets with tasks leading through the whole project and helping to place the focus on the important items. To teach the students self-studying, some given tasks are including questions, that need very specialized knowledge to be answered. This leads to a mixture of theoretical analysis and practical work within the exercise sheets. This mixture again leads to a smooth transition from learning the theoretical background to applying the theory to a specific problem, all of which helps the students to find their way from theory to practical work.

To describe the single exercise sheets in more detail: A first project can be Poisson's equation on the unit square with Dirichlet boundary conditions. Some tasks on the related exercise sheet can, for example, focus on the derivation of the weak formulation, conditions on the solution u, error computations and different boundary conditions to challenge the mathematicians. One can add tasks like explaining the difference of concepts of MPI and OpenMP, having a look at Amdahl's law, defining terms like speedup and efficiency, starting to work with the library by adding the weak formulation and boundary conditions to an existing code or implement a grid refinement for p-FEM, and explaining why a super-linear speedup can be observed in some cases to involve the computer scientists. Engineers can be included by asking questions with practical relevance like the background of Poisson's equation in physics or the interpretation of the boundary conditions. To bring them all together, one can ask for measurements and result presentation, as well as some literature research, for example.

To give the students the possibility of an open time management, the compulsory attendance is cancelled for the related practical sessions. This supports the interdisciplinary teamwork and it leads to the desired effect, that the students teach each other their specialized knowledge. This method of open time management also gives the students the possibility to determine in which way or speed they learn best.

To support the students during these practical lessons, external access to the needed hard- and software resources is provided. Furthermore, the appointed time of the course can be used by students to ask questions.

Because of the missing compulsory attendance the progress of each group in the context of the learning objectives cannot be checked by observing the groups and their work. For this reason, there is a need for a special lecture with compulsory attendance at the end of each project, in which the students have to present their results and talk about their work. This also includes a short question and answer session, in which the students can be asked questions about their work and results.

The last project in the lecture period is complemented with a special task to give the possibility of an objective grading. The students have to write a report about the theory, their practical work, and the results obtained in their last project. Together with the presentations held at the end of each project, this report is the basis to check in which way the learning objectives are achieved and therefore how to grade the students. Additionally, important skills like proper presentation of scientific results as well as speaking and writing skills are automatically practised in this way.

The time management for the practical course is done by subdividing the lecture period into several sections. The first section is a theory part, in which a general overview of the basics of the FEM is given. This overview includes fundamental knowledge about partial differential equations, the Galerkin Method with focus on variational formulations and weak solutions, discretization based on finite dimensional subspaces, definition of finite elements, grid structures and grid construction, shape functions and degrees of freedom, iterative solvers like CG-method, error estimation, and basics of parallel implementation.

After building interdisciplinary and therefore heterogeneous teams, the second section begins with practical work on a simple problem, the Poisson's equation with different boundary conditions as described for example, guided by exercise sheets. The section ends with the aforementioned presentations of the groups about their work including question and answer sessions.

Because the next practical section requires some more theoretical knowledge, the third section is again a theory part. Here, an overview of some advanced finite element methods is given, including method of lines and Rothe's method for transient problems, solving stability problems, preconditioning methods and basic benchmarks. There is also a focus on parallelism, especially for the preconditioning methods.

The fourth section is filled with a second project which is also guided by exercise sheets. The specific task formulation for this practical section differs for each group depending on their special interests, group composition, or fields of study. The thematic framework includes problems like Convection-Diffusion equations or incompressible Navier-Stokes equations. As in the first project, the students have to face the mathematical theory, investigate stability problems for instance, some practical work like implementing fractional step time-stepping methods, and the challenge of parallelization in terms of load balancing for example. Of course, some of the in project one newly acquired knowledge is also relevant for the second project, such as the usage of the library, or the application of the mathematical theory.

As mentioned before, the course ends with student presentations reviewing the theory and their own work as well as results within their specific problem. Especially for the second project, the students have to write a report to give account of their work and results.

Table 1 shows an exemplary time schedule for the outlined sections of the course based on 14 weeks per lecture period and two 90-minute-classes per week. Based on this schedule and the chosen projects, each student gains 4 ECTS for the practical course on parallel numerics, which means that each student has to achieve about 120 working hours over the lecture period.

The needed human resources are depending on the number of participants of the course. With about 20 participants per semester in our case up to two teaching assistants are needed.

Table 1. Time schedule of the practical course

Number of lecture	Content
01 – 03	Part 1: *parallelization and FEM basics, short introduction to Hiflow*^{3}*, team building*
04 – 11	Part 2: *practical work on project 1*
12 – 13	Part 3: *advanced methods*
14 – 23	Part 4: *practical work on project 2*
24 – 28	Part 5: *report writing*

4 Summary and Future Work

The practical course introduced here provides an introduction into the important field of parallel computing. Based on HiFlow^3, a library for solving PDEs with FEM, which incorporates a high level of parallelism, the students gain an impression of techniques and tools of parallel computing while working on projects within an interdisciplinary team.

The teaching contents and the syllabus of this practical course offer a wide spectrum of development. In this paper, we will give an idea of two possible improvements, that are planned for the near future.

First of all, the focus on methods of parallel computing can be intensified by teaching a selection of programming models like MPI, OpenMP, NVidia CUDA, OpenCL, and more. Since we are actually only teaching the theory of some of these paradigms in most cases, this has to be done along with a lot of practical work. Although these extensions on the syllabus can be easily combined with the FEM and HiFlow^3 in terms of additional projects, it is far too much to teach it additionally to the current curriculum within a single semester. Therefore, these improvements will inevitably lead to a syllabus for a two-semester course.

Because most of the lectures don't need a compulsory attendance and most of the work is done outside the lectures, a development towards e-learning methods or blended learning methods, is considered as the second improvement. This includes the installation of simple communication platforms, a forum for example, for questions and answers, and the use of virtual desks such as the ILIAS-platform [7]. It is also conceivable to create small private online courses for special topics, introduction to programming models or background knowledge for the current project for example, or to develop online applications to simplify the access to the content, the PDEs for instance, and to visualize the results comprehensively.

These two approaches for developing the curriculum and the teaching methods will help to move the challenges, techniques and tools of parallel computing into focus more clearly and will support the communication within a single team and across multiple teams to intensify the interdisciplinary cooperation.

References

1. Balay, S., Abhyankar, S., Adams, M.F., Brown, J., Brune, P., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Rupp, K., Smith, B.F., Zhang, H.: PETSc Web page (2014). http://www.mcs.anl.gov/petsc
2. Balay, S., Abhyankars, S., Adams, M.F., Brown, J., Brune, P., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Rupp, K., Smith, B.F., Zhang, H.: PETSc Users Manual. Argonne National Laboratory, ANL-95/11 - Revision 3.5 (2014). http://www.mcs.anl.gov/petsc
3. Balay, S., Gropp, W.D., McInnes, L.C., Smith, B.F.: Efficient management of parallelism in object oriented numerical software libraries. In: Arge, E., Bruaset, A.M., Langtangen, H.P. (eds.) Modern Software Tools in Scientific Computing, pp. 163–202. Birkhäuser Press, Boston (1997)
4. Heuveline, V., et. al.: HiFlow^3: A hardware-aware parallel finite element package. In: Brunst, H., Muller, M.S., Nagel, W.E., Resch, M.M., (eds.) Tools for High Performance Computing 2011, pp. 139–151. Springer, Heidelberg (2012)
5. Heuveline, V., Ketelaer, E., Ronnas, S., Schmidtobreick, M., Wlotzka, M.: Scalability Study of HiFlow^3 based on a Fluid Flow Channel Benchmark. Preprint Series of the Engineering Mathematics and Computing Lab (EMCL) (2012)
6. http://www.hiflow3.org/
7. http://www.ilias.de/
8. Karypis, G., Kumar, V.: A fast and highly quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**(1), 359–392 (1999)
9. Mayer, J.: ILU++: A new software package for solving sparse linear systems with iterative methods. PAMM Proc. Appl. Math. Mech. **7**, 2020123–2020124 (2007)
10. http://www.mpi-forum.org/
11. http://mumps-solver.org/
12. http://www.netlib.org/blas/
13. http://www.netlib.org/clapack/
14. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with CUDA. ACM Queue **6**(2), 40–53 (2008)
15. Saad, Y.: Iterative methods for sparse linear systems. 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia (2003)
16. Schroeder, W., et al.: The Visualization Toolkit, 3rd edn. Kitware, Inc. (2003)
17. Stone, J.E., Gohara, D., Shi, G.: OpenCL: a parallel programming standard for heterogeneous computing systems. IEEE Des. Test **12**(3), 66–73 (2010)
18. http://www.vtk.org/
19. http://www.xdmf.org/