

No. 618

December 2019

**Basic Machine Learning Approaches for the
Acceleration of PDE Simulations and
Realization in the FEAT3 Software**

**H. Ruelmann, M. Geveler, D. Ribbrock,
P. Zajac, S. Turek**

ISSN: 2190-1767

Basic Machine Learning Approaches for the Acceleration of PDE Simulations and Realization in the FEAT3 Software

H. Ruelmann, M. Geveler, D. Ribbrock, P. Zajac, S. Turek

Abstract In this paper we present a holistic software approach based on the FEAT3 software for solving multidimensional PDEs with the Finite Element Method that is built for a maximum of performance, scalability, maintainability and extensibility. We introduce basic paradigms how modern computational hardware architectures such as GPUs are exploited in a numerically scalable fashion. We show, how the framework is extended to make even the most recent advances on the hardware market accessible to the framework, exemplified by the ubiquitous trend to customize chips for Machine Learning. We can demonstrate that for a numerically challenging model problem, artificial neural networks can be used while preserving a classical simulation solution pipeline through the incorporation of a neural network preconditioner in the linear solver.

1 Introduction

Multidimensional PDE-based simulation is one of the most important yet also most challenging tasks of our time concerning computational resources both hardware- and software-side. Here the Finite Element Method (FEM) is proven to be a superior tool on complex geometries that however needs sophisticated software approaches to be feasible for production in academia and industry: Local performance on each core / CPU or device has to be provided by exploiting the underlying hardware. Numerical scale-up has to be addressed through the design and implementation of advanced parallelisation techniques. Both aspects have to be taken into account in a holistic software framework design that also provides maintainability and extensibility.

Hannes Ruelmann, Markus Geveler, Dirk Ribbrock, Peter Zajac and Stefan Turek
TU Dortmund University, Vogelpothsweg 87, 44227 Dortmund,
e-mail:
firstname.lastname@math.tu-dortmund.de

In this paper we offer insight into the third generation of the Finite Element Analysis Toolbox software family (FEAT3), developed at TU Dortmund University. In order to address the aforementioned aspects, we introduce the paradigms of *Hardware-oriented Numerics* and *Unconventional High Performance Computing* (UCHPC) in the context of performance, scalability and maintainability in section 2.

Under the aspect of extensibility, we demonstrate how the framework is catching up with modern hardware trends: Machine Learning (ML) opens a variety of options to support traditional methods of the numerical treatment of solving PDEs particularly for application-oriented CFD simulations with new algorithms on the software level. Any such approach provides access to modern hardware, since chip vendors tailor their designs to ML techniques to satisfy the upcoming and rapidly growing AI-market. In this paper we demonstrate, how artificial neural networks can assist in solving PDEs and how this is implemented in the FEAT3 framework in section 3.

2 FEAT3: Unconventional High Performance Finite Elements

2.1 Trends in modern hardware and Green HPC

In the last two decades, it became clear that the continuously increasing single-core speed, which was driven by Moore’s law, will stagnate at some point. In consequence, hardware vendors are switching their focus to parallelism, both in the sense of supercomputer clusters as well as various forms of specialized *many-core* hardware accelerators such as general-purpose Graphics Processing Units or Tensor Processing Units, where the latter are custom-designed chips tailored to Machine Learning applications. From a programmer’s point of view, these specialized hardware units differ from ordinary CPUs and their built-in vector extensions in the sense that one cannot simply utilize them by enabling a compiler switch. Instead, new specialized algorithms, which efficiently exploit the underlying hardware’s strengths, have to be designed and implemented by using third-party libraries. In the context of scientific computing, the concept, where the available hardware determines what algorithms are run on it, can be labeled as Unconventional High Performance Computing.

Another aspect, which is slowly (but steadily) gaining attention, is the continuously increasing energy consumption of supercomputers and the subsequent need for improved energy efficiency. As a consequence, the Green500 list¹ has been launched in 2007 as a means to benchmark the energy efficiency of supercomputers measured in Flops per Watt rather than the total compute power measured in Flops that is used in the famous Top500 list. Again, special accelerator hardware as well as hardware primarily designed for mobile/embedded systems, which are designed to work with a limited battery power supply, play a key role when it comes to achieving high energy efficiency on modern supercomputers.

¹ see <https://www.top500.org/green500>

2.2 Finite elements and the need for speed

The finite element method has proven to be a powerful numerical tool for solving PDEs arising from various scientific fields. Researchers in the academia value the FEM due to its underlying variational formulation, which also forms the backbone of a rigorous theory for the analysis of PDEs, as well as the properties that can be derived from this close relationship between mathematical theory and practical implementability. One notable example is the large set of methods that can be used for error estimation and error control. However, simulations of realistic problems – especially those arising from the industry – result in problem sizes, which are often several orders of magnitude larger than what the academia is typically dealing with and therefore require a different focus in software design and implementation. In consequence, any simulation toolkit that aims to tackle large problems needs to be capable of utilizing modern large-scale parallel hardware beyond the usual small-scale workstation setup, which makes parallel programming, hardware-efficient optimizations and performance engineering indispensable.

2.3 Fast linear solvers based on geometric multigrid methods

One major component of any FEM simulation, which often dominates the overall runtime, is the solution of (non-)linear systems of equations (LSEs). In addition to the usual direct factorization solvers and iterative Krylov subspace methods, the FEM also allows for more specialized solvers that take the underlying discretization into account. The most prominent class of such specialized linear solvers is the class of geometric multigrid methods (GMG), which is one of the few solvers that can solve many LSEs arising from the FEM in linear runtime, thus making it an ideal candidate for large-scale simulations. The GMG is an iterative method which (in its simplest form) solves the LSE by recursively restricting the system onto a coarser mesh, solving the LSE on the coarser mesh, and then projecting the coarse solution back onto the original LSE and post-processing this coarse solution by a *smoother*. The smoother is typically the most costly part of the GMG and its convergence properties are a crucial ingredient for obtaining the h -independent convergence (and thus linear runtime) of the GMG, see e.g. [5] [6].

2.4 FEAT3: FEM+GMG meets UHPC

To tackle the above mentioned challenges that come with modern unconventional hardware, we have been implementing the *Finite Element Analysis Toolbox 3* (FEAT3) software package, which is a modular template-based parallel FEM+GMG framework written in C++11. FEAT3 utilizes MPI to implement parallelization paradigms for large-scale supercomputer clusters based on finite element domain

decomposition, which support both simple data-parallel algorithms as well as more powerful geometric multigrid solvers based on the concept of *scalable recursive clustering* (ScaRC), see [10, 8].

FEAT3 supports 2D and 3D triangular, quadrilateral, tetrahedral and hexahedral unstructured meshes as well as structured meshes, and is currently being extended to support PDEs on manifolds. A large variety of finite element ansatz spaces have already been implemented, including (but not limited to) the standard conforming Lagrangian elements up to third order, the non-conforming Crouzeix-Raviart and Rannacher-Turek elements as well as a few higher order elements like the Argyris element, see e.g. [11].

FEAT3 can assemble the arising LSEs in various floating point formats, including the standard IEEE-754 single and double precision formats as well as the quadruple precision format offered by the *libquadmath*, which is part of the GCC’s standard library set. We also currently experimenting with various third-party libraries, which offer simulated support for low-precision data types like the half precision or the competing *bfloat16* format, which is used by many modern TPUs.

Sparse matrices can not only be assembled as generic unstructured matrices in the well-known *compressed sparse rows* (CSR) format, but also in various special matrix formats including banded or stencil-based matrices. In the case of PDE systems with multiple variables, e.g. the (incompressible) Navier-Stokes equations with velocity and pressure variables, FEAT3 additionally offers various forms of nested *meta-matrix* and *-vector* class templates. These templates allow for an almost arbitrary mixing of the *array-of-structures* and *structure-of-arrays* data blocking concepts, which play an important role in the design of flexible and efficient data structures.

As mentioned before, one primary challenge in the context of node-level performance engineering is the development of specialized algorithms which are suitable to utilize the underlying hardware efficiently and unfortunately many of these algorithms cannot be hidden behind an opaque back end which serves as a simple hardware abstraction layer. Based on the experiences we have gained with our previous software packages, see [7, 8], as well as several specialized benchmarking projects, see [9], we have realized that it is often necessary to access low-level hardware API functions throughout the whole simulation code directly and this often competes with the desire to provide an easy-to-use abstract high-level interface, which is what most other academic FEM software packages prioritize. FEAT3, on the other hand, has been designed from ground up to support unconventional hardware (including GPUs and TPUs) in numerical research software applications, especially by offering low-level access to all underlying data structures and algorithms, thus making it an ideal testing ground in early development stages. This focus on specialized hardware support via transparent class templates is a major distinguishing feature of FEAT3.

It is important to mention that *all* third-party libraries (including CUDA and MPI) are supported in an *opt-in* fashion, i.e. FEAT3 can be compiled and used in a *naked* build mode (with reduced functionality) without any other dependencies than the C++ standard library. This ensures that FEAT3 can be easily ported to new hardware and operating systems other the usual Linux/Unix ecosystems, even if one or more third-party libraries cannot be compiled on these platforms, which allows us to easily

exploit a broad range of hardware from PowerPC Clusters over Windows desktop machines to embedded ARM systems. FEAT3 has already been tested successfully on low-energy systems running on solar-powered battery power supplies, see [3, 4].

The build system for FEAT3 is based on the popular *CMake* system along with a small set of scripts written in the *Python* programming language, which help to enhance the capabilities of *CMake* to support various build settings via a custom user-controlled build-id system. Our build system also includes a basic test system based on the test driver of *CMake*, which is executed nightly on our Linux compute servers as well as our university’s cluster LiDO3². The correctness of most core classes of the FEAT3 kernel is ensured by a set of *unit-tests*, which test individual classes and their member functions in an isolated testing environment. In addition, the test system also contains a basic set of more complex *regression test* applications, which help to determine whether changes to the kernel classes have changed the behaviour of code that is composed of many interacting classes, which therefore cannot be tested by isolated unit-tests. These nightly tests are compiled with a set of different compilers and different build configurations to continuously ensure the C++11 standard conformity and to detect unexpected changes in code behavior induced by platform changes or compiler bugs. This unit test system is complemented by several specialized benchmarking projects, e.g. the CFD Benchmarking Project [2].

The source code of FEAT3 is released under the GPL3 open source license and is publicly available in the form of a git repository, which can be accessed from the FEATFLOW website³.

3 A concise Machine Learning framework to accelerate linear solvers

3.1 Poisson problem and anisotropies

When solving the incompressible Navier-Stokes equation with global Multilevel Pressure Schur Complement techniques, the so called Pressure Poisson problem is dominant regarding calculation time [2]. For the sake of simplicity we therefore choose the Poisson equation to be our model problem, which reads:

Find $u : \Omega \rightarrow \mathbb{R}$ such that

$$-\Delta u = f \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \partial\Omega \quad (1)$$

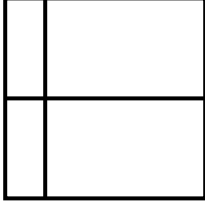
and discretize it with the Finite Element method.

For the unit square $\Omega = (0, 1)^2$ domain the standard quadrilateral triangulation results in h-independent convergence of the multigrid method with a fixed number of smoothing basis iterations for different smoother. By introducing some anisotropies

² see <https://www.lido.tu-dortmund.de/cms/en/home/index.html>

³ see <http://www.featflow.de/en/software/feat3.html>

to the initial mesh, see figure 1, the convergence commences to be dependent on the grid size. Merely the ILU method with the Reverse Cuthill-McKee renumbering algorithm maintains the h-independence for the directional anisotropy (fig. 1 a). In case of aspect ratios in both direction (fig. 1 b) renumbering has no further effect and the ILU even with renumbering as well as the other smoothing methods will lead to more multigrid iterations for finer meshes.

a)		lvl	dofs	Jac (0.5)	GS (1.0)	SPAI-1(1.0)	ILU-0 (0.5)	ILU-RCMK
		10	2,100,225	109	33	24	26	7
		9	525,825	106	32	23	25	7
		8	131,841	103	30	22	24	7
		7	32,960	98	28	21	23	7
		6	8,240	90	25	19	21	6
		5	2,060	78	22	17	18	6
		4	515	55	16	13	12	6

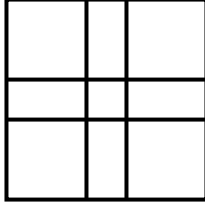
b)		lvl	dofs	Jac (0.5)	GS (0.7)	SPAI-1(1.0)	ILU-0 (0.7)
		9	2,362,369	654	370	140	102
		8	591,361	619	350	130	95
		7	148,225	562	319	118	85
		6	37,249	486	289	103	75
		5	9,409	377	218	80	57
		4	2,401	258	166	51	34
		3	625	175	96	31	20

Fig. 1 Left: coarse grid; right: Number of multigrid V-cycles for different smoothers with 8 pre- and post-smoothing steps each. Damping parameter in brackets. Aspect ratio: a) 1:10, b) 1:20

3.2 Approximate Inverses with Neural Network

In this approach we use a neural network prototype trained on function regression to map the FEM system matrix to its corresponding inverse and thus get a beneficial approximation of that inverse which we can use as smoother in multigrid methods or as preconditioner. The structure of the neural network is important to yield strong approximate inverses which are able to smooth the system or lead to converging methods when used e. g. in a Richardson iteration solver. On the other hand it provides a large design space in which we can keep balance between calculation time and accuracy. In [1] we show that fully-connected feed forward multilayer perceptrons are able to extrapolate coefficient matrices which are suitable SPAI-like preconditioners within defect correction methods. In this paper we expand the working system to anisotropic meshes.

To avoid storage problems for larger matrices we use the online-learning method plus only feed the non-zero entries of the system matrix to the neural network. The approximate inverse can be filtered to a sparse matrix thus the assembly of the preconditioner is one pass of the neural network in addition to the application,

which is a sparse-matrix-vector-multiplication. With sophisticated matrix formats this performs in parallel and efficiently on modern hardware accelerators. This perfectly couples with FEAT3, which is also used to generate the training data tensor. We randomly shift the inner nodes in order to get a training dataset with the system matrices and associated inverses. This procedure bases on r-adaption techniques we want to use during the CFD simulation, with which the node shift is used to minimize the error.

3.3 Neural Networks for anisotropic meshes

To measure the quality of the approximate inverse out of neural networks we use the modified Richardson iteration and compare the number of solver iterations with the conventional damped Jacobi as well as the Gauß-Seidel method. Figure 2 displays the results for different aspect ratios on disturbed meshes. For higher anisotropies, e. g. 1:10 (see fig. 2b), the common methods collapse and the Jacobi method reaches the maximum iteration number. The low number of iterations even for finer meshes gives strong evidence that neural networks can generate valuable preconditioners. The number of iterations raise just slightly for different refinements and the behavior depends on the training parameter of the neural network.

	dim	Jac (0.7)	GS (1.0)	NN		dim	Jac (0.7)	GS (1.0)	NN
a)	25	422	147	26	b)	25	1101	385	22
	121	1955	683	39		121	5036	1762	32
	529	8622	3017	64		529	10000	7939	37

Fig. 2 Number of iterations for damped Jacobi (Jac), Gauß-Seidel and Richardson iteration with neural networks. Aspect ratio 1:3 (left) and 1:10 (right)

4 Conclusion and future work

We showed that it is possible to combine methods of Machine Learning, which empowers several scientific fields and industry, with the numerical treatment of solving PDEs. Regarding real world CFD simulation, large problem sizes and difficulties like anisotropies arise. The presented Finite Element Analysis Toolbox 3 is specially tailored to solve such problems with respect to performance, hardware efficiency as well as a high accuracy. FEAT3 offers the ability to future-oriented UCHPC along with a easy-to-use framework for academic researchers. On the one hand Machine Learning fits perfectly into this gap of using modern hardware, since chip vendors specially adjust their portfolio to satisfy the fast-growing AI-market. And we demonstrated in a small test scenario, that the designed Machine Learning-based preconditioner

with a Richardson iteration as solver maintained low numbers of iteration even for anisotropies with such a high aspect ratio that common solvers fail on the other hand. This is an evidence that Machine Learning techniques can perfectly empower and amplify current numerical PDE solving methods.

One of the most important enhancements in future work will be the extension of the neural network to real applications including large problem sizes.

References

1. Ruelmann, H., Geveler, M., Turek, S.: On the Prospects of Using Machine Learning for the Numerical Simulation of PDEs: Training Neural Networks to Assemble Approximate Inverses, ECCOMAS Newsletter June 2018, pp. 27 – 32, 2018.
2. Turek, S.: Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach, vol. 6. Springer, 1999
3. Geveler, M., Ribbrock, D., Ruelmann, H., Donner, D., Höppke, C., Schneider, D., Tomaschewski, D., Turek, S.: The ICARUS white paper: A scalable, energy-efficient, solar-powered HPC center based on low power GPUs, UcHPC'16 at Euro-Par'16, Grenoble, 2016
4. Geveler, M., Reuter, B., Aizinger, V., Göddeke, D., Turek, S.: Energy efficiency of the simulation of three-dimensional coastal ocean circulation on modern commodity and mobile processors-A case study based on the Haswell and Cortex-A15 microarchitectures, LNCS, ISC'16, Computer Science-Research and Development, 1-10, Workshop on Energy-Aware HPC, Springer, doi = 10.1007/s00450-016-0324-5, 2016
5. M. Geveler, D. Ribbrock, D. Goeddeke, P. Zajac, S. Turek: Efficient Finite Element Geometric Multigrid Solvers for Unstructured Grids on Graphics Processing Units; in P. Ivanyi, B.H.V. Topping, (Editors), "Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering", Civil-Comp Press, Stirlingshire, UK, Paper 22, doi:10.4203/ccp.95.22, 2011
6. M. Geveler, D. Ribbrock, D. Goddeke, P. Zajac, S. Turek: Towards a complete FEM-based simulation toolkit on GPUs: Unstructured grid finite element geometric multigrid solvers with strong smoothers based on sparse approximate inverses; Computers and Fluids, Vol 80, 2013, pp. 327-332, doi:10.1016/j.compfluid.2012.01.025
7. S. Turek, D. Göddeke, C. Becker, S.H.M. Buijssen, H. Wobker: FEAST – realization of hardware-oriented numerics for HPC simulations with finite elements; Concurrency and Computation: Practice and Experience, 2010, Volume 22, Issue 16, doi:10.1002/cpe.1584
8. D. Göddeke: Fast and Accurate Finite-Element Multigrid Solvers for PDE Simulations on GPU Clusters; PhD thesis, Lehrstuhl für angewandte Mathematik und Numerik, Fakultät für Mathematik, Technische Universität Dortmund, 2010, doi:10.17877/DE290R-8758
9. D. van Dyk, M. Geveler, S. Mallach, D. Ribbrock, D. Göddeke, C. Gutwenger: HONEI: A collection of libraries for numerical computations targeting multiple processor architectures; Computer Physics Communications, Volume 180, Issue 12, 2009, pp. 2534-2543, doi:10.1016/j.cpc.2009.04.018
10. S. Turek, C. Becker, S. Kilian: Hardware-oriented numerics and concepts for PDE software; Future Generation Computer Systems 22 (2006) 217–238, doi:10.1016/j.future.2003.09.007
11. P.G. Ciarlet: The Finite Element Method for Elliptic Problems; North-Holland, 1978, doi.org:10.1137/1.9780898719208