Continuous Benchmarking of Numerical Algorithms Implemented in M++ via Gitlab CI/CD and Google Benchmark*

Niklas Baumgarten[1] and Daniele Corallo[2]

[1]Department of Mathematics, Heidelberg University
[2]Department of Mathematics, Karlsruhe Institute of Technology

Abstract

We present an automated framework for benchmarking numerical algorithms that solve partial differential equations under consistent and reproducible conditions using the parallel finite element software M++. This framework integrates GitLab CI/CD, Google Benchmark, and the HoreKa supercomputing system to enable continuous integration and benchmarking. By incorporating ongoing software development, the framework supports improving performance and reliability, which are vital for various scientific computing applications, including wave propagation, cardiovascular simulations, dislocation dynamics, and uncertainty quantification. These applications motivate the two benchmarking examples presented in this text. We further outline the benchmarking workflow as well as the use of a research database storing comprehensive performance data, facilitating reproducibility for future studies.

1 Introduction

Advancements in computational science and engineering nowadays often depend on the sustainable and robust development of open-source software. It has become a major carrier of knowledge and scientific methodologies, playing a key role in understanding physical, biological, technical, or economical phenomena. As complexity increases in the modeling of such phenomena, the supporting software systems must also grow in complexity, creating a multi-target optimization problem where usability, flexibility, performance, reliability, and maintainability must be balanced. Consequently, careful software engineering, supported by performance data collection and automated testing, has become a crucial part of the development process.

Here, we present our approach to this engineering task: an automated benchmarking system within the release cycle for M++, a high-performance computing (HPC) finite element method (FEM) software [8], developed over two decades by numerous researchers based on the initial works [26, 27]. Today, M++ is applied across a wide range of scientific fields, such as geoscience, where it is used for full waveform inversion in seismic imaging [10, 15, 22] and for simulating gas dynamics in carbon capturing [21]. It is also employed in material science to model nonlinear solid mechanics [9] and dislocation dynamics [23, 24, 25], as well as in life science to simulate cardiovascular processes [17, 18, 19].

M++ implements a wide range of numerical methods, e.g. standard Lagrange FEM, discontinuous Galerkin (DG) and Raviart-Thomas discretizations as well as a Message Passing Interface (MPI) based parallel linear algebra providing several preconditioners and Krylov subspace solvers [8]. Further, more recent features include implicit space-time discretizations for wave [11, 13, 14] and transport [28, 12] equations, interval arithmetic methods for computer-assisted existence proofs [31], and efficient time integration schemes combined with DG methods [20] for simulating time-dependent wave problems. The software also provides methodologies for large-scale uncertainty quantification (UQ), such as multilevel Monte Carlo, stochastic collocation and stochastic gradient descent methods [4, 5, 6, 7], to cope with the inherent uncertainties in the physical models and to impose an optimal control to the aforementioned applications.

*Proceedings of the 9th bwHPC-Symposium 2023. Mannheim Conference Series (MaConf). 2025. [CC BY 4.0]

Supporting such a wide range of applications and algorithms results in different requirements on the software architecture and the development process. It raises the questions on, when, how and where performance improvements should be considered and which new features might be pursued to further expand the application and decrease the runtime of the software. The discussion in [1] and the development process of *Ginkgo* [2] and *OpenCarp* [3] serve as inspiration to address these questions by including benchmarking and automated performance evaluation to the continuous integration/continuous deployment (CI/CD) pipeline of M++. As a result, we collect comprehensive data to make informed decisions on how to proceed in the development process, while ensuring that already achieved results are maintained or even improved.

In this text, we describe the details of this benchmarking approach, its integration within the CI/CD pipeline, and the HoreKa supercomputing system to enable fair comparisons using Google Benchmark as the evaluation tool. We present two benchmarking examples, each solving a PDE to outline the simplicity of the new set-up for the software.

2 Methodology

Researchers at different universities develop M++, e.g. at the Karlsruhe Institute of Technology (KIT) or Heidelberg University, all with different research projects and different requirements to the software. To streamline the development and to be able to always move forward towards a common, new and improved version of the software, we work within release cycles as described in [16]. We refer to Figure 1 at hand of which we explain the procedure and the involved technologies used to create a new version of the software.

The starting point for a new software release (left side of Figure 1) typically involves programming a new algorithm, designing a computational model, or enhancing existing code. Researchers with access to the Git repository[1] can make changes, which are automatically built by the Gitlab CI pipeline with various configurations within Docker containers. These containers run on a Kubernetes cluster at the Department of Mathematics at KIT, executing a rigorous and large set of tests to verify the correctness of the code (cf. top left of Figure 1). Typical tests check for consistency of the implementation based on mathematical theory, execute convergence experiments with respect to a certain discretization parameter, but also verify the integration of the core M++ library within projects depending upon it.

Release candidates of M++ which have passed the build and verification stage can be deployed in a next step on the HoreKa supercomputing system (cf. top right of Figure 1). Here, the pipeline executes HPC-experiments and benchmarks parallelized with Open MPI, typically in the range of 64 to 16384 processing units, to create the simulation data of interest in the form of HDF5[2] or VTU files, but also to collect performance and configuration records in JSON files. All benchmark data is collected within the Google benchmark framework and archived.

Subsequently, the data collected within the benchmarking stage is analyzed and post-processed via Python tools (cf. bottom right of Figure 1). This involves an automated comparison with previous release data, and preparation for a final publication with a persistent identifier via the RADAR[3] service. (cf. bottom left of Figure 1). With this final publication, new benchmarking results are available, serving as the reference for future developments and a new release cycle.

At the heart of this cycle is Gitlab CI/CD as the automation tool and the driver of this procedure. It takes over the configuration and execution of the aforementioned workflow by defining the utilized infrastructure and stages in YML files. As a result, all components are easy to maintain, replace or extend simply by changing the Gitlab CI/CD configuration. Under this development workflow, we currently create between three and five new releases every year depending on the implemented features.

With over 30 releases ([30] as the latest), each developed with increasing levels of scrutiny, this workflow has driven major improvements in usability, reliability and ensuring reproducibility of results, which is crucial for the scientific method. To highlight the continuous advancements in quality control, we present two benchmarking examples in the following section.

[1]`https://gitlab.kit.edu/kit/mpp/mpp/`

[2]availbale in upcoming release

[3]`https://radar.products.fiz-karlsruhe.de/en`

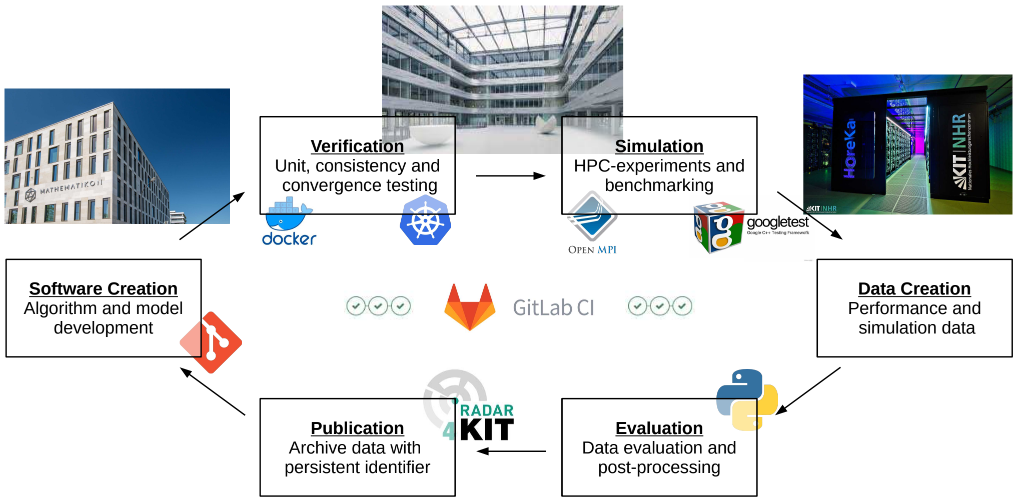
Verification
Unit, consistency and convergence testing
Simulation
HPC-experiments and benchmarking
Software Creation
Algorithm and model development
Data Creation
Performance and simulation data
Publication
Archive data with persistent identifier
Evaluation
Data evaluation and post-processing
GitLab CI
docker
Open MPI
googletest
RADAR 4KIT
HoreKa
KIT NHR

Figure 1: Release cycle of M++ with automated testing, benchmarking and performance data collection. Figure inspired by [16] and adapted from [4].

3 Examples

Defining a new benchmark is as simple as defining a new C++ function and registering it withing the Google benchmark framework (cf. Listing 1). The following two subsections will introduce the definition of the `EllipticSubsurfaceDiffusion3D` and the `AcousticGaussHatAndRicker2D` benchmark, which run among many others on the HoreKa supercomputing system and serve here as examples. While research software quite often favors specification over generalization, and therefore, tight integration of features over usability, we want to emphasize with these examples the user accessibility for running PDE related applications.

```
BENCHMARK(EllipticSubsurfaceDiffusion3D);                 // Registers 3D diffusion problem as benchmark

BENCHMARK(AcousticGaussHatAndRicker2D);                   // Registers 2D acoustic problem as benchmark

// ...                                                    // Register further benchmarks

int main(int argc, char **argv) {
 MppTest mppTest = MppTestBuilder(argc, argv).WithPPM(); // MppTest is wrapper around Google benchmark
 return mppTest.RUN_ALL_MPP_BENCHMARKS(argc, argv);      // Runs all benchmarks
}
```

Listing 1: Mockup main file collecting several PDE solver benchmarks

3.1 An Elliptic Subsurface Diffusion Benchmark in 3D

Firstly, we consider a 3D elliptic subsurface diffusion problem on a unit cube $\mathcal{D} = (0,1)^3$. Given a spatially dependent permeability $\boldsymbol{\kappa}\colon \mathcal{D} \to \mathbb{R}$, homogeneous Dirichlet boundary conditions on the bottom face $\Gamma_{\mathrm{D}} = \{\mathbf{x} \in \partial\mathcal{D}\colon x_2 = 0\}$ and an inflow Neumann boundary condition on the top face $\Gamma_{\mathrm{N}} = \{\mathbf{x} \in \partial\mathcal{D}\colon x_2 = 1\}$, the computational task is to find the pressure head $\mathbf{u}\colon \mathcal{D} \to \mathbb{R}$, such that

$$\begin{cases} -\operatorname{div}(\boldsymbol{\kappa}(\mathbf{x})\nabla\mathbf{u}(\mathbf{x})) & = & 0 & \text{on} & \mathcal{D} \\ \mathbf{n}\cdot\nabla\mathbf{u}(\mathbf{x}) & = & -1 & \text{on} & \Gamma_{\mathrm{N}} \\ \mathbf{u}(\mathbf{x}) & = & 0 & \text{on} & \Gamma_{\mathrm{D}} \end{cases} .$$

The permeability function $\boldsymbol{\kappa}$ takes either the value one or one-hundred at 16 distinct locations (cf. [29] for definition). In M++ this problem can be implemented as a benchmark as shown in Listing 2. The

EllipticPDESolver class can be configured in various ways to solve the problem including different ansatz spaces for the FEM and various preconditioned linear solvers. In its default configuration, the benchmark uses a GMRES solver with an incomplete LU preconditioner for solving the linear system arising by using a standard Lagrange FEM of degree one. The benchmarking functionality in Listing 2 solves the PDE several times to collect statistically robust timing data by using the wrapper macros BENCH_START_TIMING() and BENCH_END_TIMING(state), accounting for the run time imbalance of different MPI ranks.

```
static void EllipticSubsurfaceDiffusion3D(benchmark::State &state) {
 const int level = 4;                                                  // Discretization level
 EllipticPDESolver pdeSolver();                                        // Creates default solver

 auto pdeProblem = EllipticPDEProblemBuilder("Cube")                   // Defines problem on unit cube
  .WithDirichletBoundary([](const Point &x) { return (x[1] == 0); }) // Defines Dirichlet boundary
  .WithNeumannBoundary([](const Point &x){ return (x[1] == 1); })     // Defines Neumann boundary
  .WithPermeability([](const Point &x) { return Permeability(x); })  // Confer repository for def.
  .WithSolution([](const Point &x) { return 0.0; })                   // Solution on Dirichlet boundary
  .WithFlux([](const Point &x) { return {0.0, -1.0, 0.0}; })          // Flux on Neumann boundary
  .WithLoad([](const Point &x) { return 0.0; })                       // Definition of right-hand-side
  .WithName("EllipticSubsurfaceDiffusion3D")                          // Name of problem
  .BuildShared();                                                     // Creates problem shared pointer

 for(auto _ : state) {                                                 // Multiple benchmark runs
  BENCH_START_TIMING();                                                // Starts timing for benchmark
  benchmark::DoNotOptimize(pdeSolver.Run(pdeProblem, level));          // Solves the problem on level
  BENCH_END_TIMING(state);                                             // Ends timing for benchmark
 }
}
```

Listing 2: Mockup implementation of a 3D elliptic subsurface diffusion benchmark in M++

3.2 An Acoustic Wave Propagation Benchmark in 2D

The PDE for the second benchmark example is an acoustic wave equation where we search a pressure and velocity component $(p, \mathbf{v})\colon [0,T] \times \mathcal{D} \to \mathbb{R} \times \mathbb{R}^2$ on a unit square $\mathcal{D} = (0,1)^2$ and with end time $T = 1$, such that we satisfy

$$\begin{cases} \partial_t \mathbf{v}(t,\mathbf{x}) - \nabla p(t,\mathbf{x}) &= \mathbf{f}(t,\mathbf{x}) & \text{on} & (0,T] \times \mathcal{D} \\ \partial_t p(t,\mathbf{x}) - \operatorname{div}(\mathbf{v}(t,\mathbf{x})) &= g(t,\mathbf{x}) & \text{on} & (0,T] \times \mathcal{D} \\ \mathbf{n} \cdot \mathbf{v}(t,\mathbf{x}) &= 0 & \text{on} & (0,T] \times \partial\mathcal{D} \\ \mathbf{v}(0,\mathbf{x}) &= 0 & \text{on} & \mathcal{D} \\ p(0,\mathbf{x}) &= 0 & \text{on} & \mathcal{D} \end{cases}$$

with homogeneous boundary and initial conditions. The right-hand side is given by a constant $\mathbf{f} \equiv \mathbf{0}$ and a separated $g(t,\mathbf{x}) = g_1(t)\, g_2(\mathbf{x})$. Here, $g_1(t)$ is a Ricker wavelet,

$$g_1(t) = 10\left(1 - \left(\tfrac{t}{a}\right)^2\right)\exp\left(-\tfrac{t^2}{2a^2}\right) \quad \text{with} \quad a = \tfrac{\pi}{10}, \quad t \in [0,1].$$

while $g_2(\mathbf{x})$ is a nascent delta function centered at $\mathbf{c} = (0.5, 0.75)^\top$ and with a diameter of $w = 0.1$. The nascent delta is multiplied with $\overline{g}_2$ such that $\|g_2\|_{\mathrm{L}^1(\mathcal{D})} = 1$, thereby it is given by

$$g_2(\mathbf{x}) = \begin{cases} \overline{g}_2 \exp\left(-\left(1 - \|\frac{\mathbf{x}-\mathbf{c}}{w}\|_2^2\right)^{-1}\right), & \|\mathbf{x}-\mathbf{c}\|_2 < w \\ 0, & \|\mathbf{x}-\mathbf{c}\|_2 \geq w \end{cases} \qquad \mathbf{x} \in \mathcal{D}.$$

A similar example is also used in [4, 5] where this wave propagates through randomly distributed material. As a result, this benchmark ensures the reproducibility of these studies and scaling results.

For an illustration of the pressure component p and its temporal development, we refer to Figure 2, for the implemented benchmark to Listing 3. Here, a DG discretization of degree two is used in combination with an implicit midpoint rule. The linear system in each time step is solved with a GMRES solver preconditioned by a point-block Jacobi iteration by default. The relation between the mesh width h and the time-step size τ is chosen as $\tau = C_{\mathrm{CFL}} h$ with $C_{\mathrm{CFL}} = 0.25$.

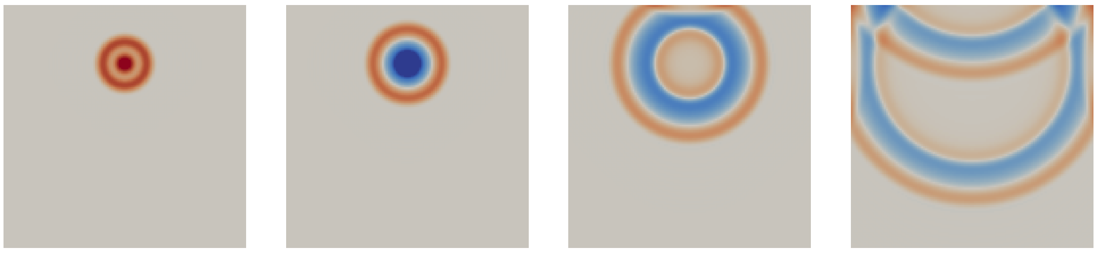
Figure 2: Pressure component p at different time steps

```
static void AcousticGaussHatAndRicker2D(benchmark::State &state) {
 const int level = 7;                                              // Discretization level
 auto pdeSolver = AcousticPDESolver(PDESolverConfig()              // Creates PDE solver with implicit
  .WithRkOrder(-2).WithDegree(2));                                 // midpoint rule and DG degree two

 const double startTime = 0.0;                                     // Start time of problem
 const double gaussWidth = 0.1;                                    // Width of GaussHat
 const double rickerDuration = 0.1;                                // Duration of Ricker wavelet
 const Point shotLocation = {0.5, 0.75};                           // Location of shot

 auto pdeProblem = AcousticPDEProblemBuilder("Square")             // Defines problem on unit square
  .WithForce([&](double t, const Point &x, COMPONENT comp) {      // Mockup for force term
    if (comp != COMPONENT::P0) { return 0.0; }                    // Homo. velocity right-hand-side
    return 10.0 * Ricker(t - startTime, rickerDuration)           // Defines right-hand-side for
       * GaussHat(dist(x, shotLocation), gaussWidth);             // Preassure component
  })
  .WithName("GaussHatAndRicker2D")                                 // Name of problem
  .WithEndTime(1.0)                                                // Defines end time of problem
  .WithCFL(0.25)                                                   // Defines CFL number of problem
  .BuildShared();                                                  // Creates problem shared pointer

 for(auto _ : state) {                                             // Multiple benchmark runs
  BENCH_START_TIMING();                                            // Starts timing for benchmark
  benchmark::DoNotOptimize(pdeSolver.Run(pdeProblem, level));      // Solves the problem on level
  BENCH_END_TIMING(state);                                         // Ends timing for benchmark
 }
}
```

Listing 3: Mockup implementation of a 2D acoustic wave propagation benchmark in M++

4 Evaluation and Conclusion

The final evaluation of the benchmarking results, e.g. of the examples presented in Section 3, is part of the automated workflow as outlined in Section 2 and includes three levels. The first two levels are executed on a weekly basis, and prior to each release of M++. This includes over 60 benchmarks testing critical functionalities (like Krylov subspace methods or mesh generation) running on a single node using 64 CPUs for up to 3 hours in total. The results are evaluated using Python tools that generate historical performance plots for comparison, such that any performance change is noticeable in the deviations of these plots.

These functionality benchmarks are accompanied by larger integration benchmarks, based on applications from prior work [8, 10, 4, 20], and cover complete 2D and 3D simulations finishing within 10 minutes each and requiring up to 128 GB of memory. Previously observed performance is then imposed with some tolerance to the slurm job scheduler, such that if a job exceeds its limits, it is terminated and marked as failed.

The last level is only executed on demand. This includes all job scripts used in [11, 5, 6, 7] which are stored in the Git repository for reproducibility. Although these large-scale HPC experiments—using up to 16,384 CPUs over 12 hours—are not run regularly due to their cost, they remain executable via a simple pipeline trigger. We refer to the pipeline configurations for a detailed overview of which job has which CPU-time and memory requirements in order to pass[1].

We view software development, maintenance, documentation, and performance analysis as integral to the scientific method, enhancing research robustness and credibility. This paper outlines our approach to

[1]https://gitlab.kit.edu/kit/mpp/mpp/

sustainable research software development, focusing on new benchmarking capabilities and automation. We present two simple benchmarking examples to highlight the workflow's flexibility and ease. With this framework we support future software development, ensuring result reproducibility and code reliability.

5 Acknowledgements

The associated software development to the benchmarking framework has started during the student project "Praktikum Softwareentwicklung" at the Karlsruhe Institute of Technology (KIT). Nico Fröhlich, Noah Hackenjos, Clara Maier, Tobias Merkel and Simon Wülker have implemented new ways for in- and output file handling, while the latter two extended their work on the outlined benchmarking framework. Their work was funded by the CRC 1173 on wave phenomena and by the research group of Christian Wieners.

We further acknowledge the technical support by Mathias Reichardt as well as by the National High-Performance Computing Center (NHR) at KIT. All benchmarks are performed on the HoreKa supercomputer funded by the Ministry of Science, Research and the Arts Baden-Württemberg and by the Federal Ministry of Education and Research.

References

[1] H. Anzt, F. Bach, S. Druskat, F. Löffler, A. Loewe, et al. An environment for sustainable research software in germany and beyond - current state, open challenges, and call for action. F1000Research, 9, 2020.

[2] H. Anzt, T. Cojean, G. Flegar, F. Göbel, T. Grützmacher, P. Nayak, T. Ribizel, Y. M. Tsai, and E. S. Quintana-Ortí. Ginkgo: a modern linear operator algebra framework for high performance computing. ACM Trans. Math. Software, 48(1):Art. 2, 33, 2022.

[3] F. Bach, J. Klar, A. Loewe, J. Sánchez, G. Seemann, Y.-L. Huang, and R. Ulrich. The openCARP CDE–concept for and implementation of a sustainable collaborative development environment for research software. arXiv preprint arXiv:2201.04434, 2022.

[4] N. Baumgarten. A Fully Parallelized and Budgeted Multi-level Monte Carlo Framework for Partial Differential Equations. PhD thesis, Karlsruher Institut für Technologie (KIT), 2023.

[5] N. Baumgarten, S. Krumscheid, and C. Wieners. A fully parallelized and budgeted multilevel monte carlo method and the application to acoustic waves. SIAM/ASA Journal on Uncertainty Quantification, 12(3):901–931, 2024.

[6] N. Baumgarten, R. Kutri, and R. Scheichl. A budgeted multi-level monte carlo method for full field estimates of multi-pde problems, 2025.

[7] N. Baumgarten and D. Schneiderhan. Multilevel stochastic gradient descent for optimal control under uncertainty. arXiv preprint arXiv:2506.02647, 2025.

[8] N. Baumgarten and C. Wieners. The parallel finite element system M++ with integrated multilevel preconditioning and multilevel Monte Carlo methods. Comput. Math. Appl., 81:391–406, 2021.

[9] H. R. Bayat, J. Krämer, L. Wunderlich, S. Wulfinghoff, S. Reese, B. Wohlmuth, and C. Wieners. Numerical evaluation of discontinuous and nonconforming finite element methods in nonlinear solid mechanics. Comput. Mech., 62(6):1413–1427, 2018.

[10] T. Bohlen, M. R. Fernandez, J. Ernesti, C. Rheinbay, A. Rieder, and C. Wieners. Visco-acoustic full waveform inversion: from a DG forward solver to a Newton-CG inverse solver, 2021.

[11] D. Corallo, W. Dörfler, and C. Wieners. Space-time discontinuous Galerkin methods for weak solutions of hyperbolic linear symmetric Friedrichs systems. Technical Report 1, 2023.

[12] D. Corallo and C. Wieners. A parallel adaptive space-time discontinuousgalerkin method for transport in porous media. 2025.

[13] W. Dörfler, S. Findeisen, and C. Wieners. Space-time discontinuous Galerkin discretizations for linear first-order hyperbolic evolution systems. Comput. Methods Appl. Math., 16(3):409–428, 2016.

[14] W. Dörfler, S. Findeisen, C. Wieners, and D. Ziegler. Parallel adaptive discontinuous Galerkin discretizations in space and time for linear elastic and acoustic waves. 25:61–88, [2019] ©2019.

[15] J. Ernesti and C. Wieners. A space-time discontinuous Petrov-Galerkin method for acoustic waves. 25:89–115, [2019] ©2019.

[16] D. Farley. Continuous Delivery Pipelines - How to Build Better Software Faster. Pearson Education, 2020.

[17] J. Fröhlich, T. Gerach, J. Krauß, A. Loewe, L. Stengel, and C. Wieners. Numerical evaluation of elasto-mechanical and visco-elastic electro-mechanical models of the human heart. GAMM-Mitteilungen, 46(3-4):e202370010, 2023.

[18] J. Fröhlich. A segregated finite element method for cardiac elastodynamics in a fully coupled human heart model. PhD thesis, Karlsruher Institut für Technologie (KIT), 2022.

[19] T. Gerach, S. Schuler, J. Fröhlich, L. Lindner, C. Wieners, and A. Loewe. Electro-mechanical whole-heart digital twins a fully coupled multi-physics approach. Mathematics, 9(11):1247, 2021.

[20] M. Hochbruck, T. Pažur, A. Schulz, E. Thawinan, and C. Wieners. Efficient time integration for discontinuous Galerkin approximations of linear wave equations [Plenary lecture presented at the 83rd Annual GAMM Conference, Darmstadt, 26th–30th March, 2012]. ZAMM Z. Angew. Math. Mech., 95(3):237–259, 2015.

[21] M. M. Knodel, S. Kräutle, and P. Knabner. Global implicit solver for multiphase multicomponent flow in porous media with multiple gas components and general reactions: global implicit solver for multiple gas components. Comput. Geosci., 26(3):697–724, 2022.

[22] C. Rheinbay. All-At-Once and Reduced Solvers for Visco-Acoustic Full Waveform Inversion. PhD thesis, Dissertation, Karlsruhe, Karlsruher Institut für Technologie (KIT), 2023, 2023.

[23] K. Schulz, L. Wagner, and C. Wieners. A mesoscale continuum approach of dislocation dynamics and the approximation by a Runge-Kutta discontinuous Galerkin method. International Journal of Plasticity, 120:248–261, 2019.

[24] E. Thawinan. Numerical approximation of higher-dimensional Continuum Dislocation Dynamics theory in single crystal plasticity. PhD thesis, 2015.

[25] L. Wagner. A discontinuous Galerkin method for continuum dislocation dynamics in a fully-coupled elastoplasticity model. PhD thesis, Karlsruher Institut für Technologie (KIT), 2019.

[26] C. Wieners. Distributed point objects. A new concept for parallel finite elements. In Domain decomposition methods in science and engineering, volume 40 of Lect. Notes Comput. Sci. Eng., pages 175–182. Springer, Berlin, 2005.

[27] C. Wieners. A geometric data structure for parallel finite elements and the application to multigrid methods with block smoothing. Comput. Vis. Sci., 13(4):161–175, 2010.

[28] C. Wieners. A space-time discontinuous galerkin discretization for the linear transport equation. Computers & Mathematics with Applications, 152:294–307, 2023.

[29] C. Wieners, D. Corallo, D. Schneiderhan, L. Stengel, H. D. N. Pham, and N. Baumgarten. Mpp 3.4.1, 2024.

[30] C. Wieners, D. Corallo, D. Schneiderhan, L. Stengel, H. D. N. Pham, and N. Baumgarten. Mpp 3.5.0, 2025.

[31] J. M. Wunderlich. Computer-assisted Existence Proofs for Navier-Stokes Equations on an Unbounded Strip with Obstacle. PhD thesis, Karlsruher Institut für Technologie (KIT), 2022.