



SEBASTIAN GÖTSCHEL¹, ANTON SCHIELA, MARTIN
WEISER²

Kaskade 7 – a Flexible Finite Element Toolbox

¹  0000-0003-0287-2120
²  0000-0002-1071-0044

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: <http://www.zib.de>

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Kaskade 7 – a Flexible Finite Element Toolbox

Sebastian Götschel*, Anton Schiela†, Martin Weiser*

September 9, 2019

Abstract

Kaskade 7 is a finite element toolbox for the solution of stationary or transient systems of partial differential equations, aimed at supporting application-oriented research in numerical analysis and scientific computing. The library is written in C++ and is based on the DUNE interface. The code is independent of spatial dimension and works with different grid managers. An important feature is the mix-and-match approach to discretizing systems of PDEs with different ansatz and test spaces for all variables.

We describe the mathematical concepts behind the library as well as its structure, illustrating its use at several examples on the way.

Keywords: finite elements, generic programming, partial differential equations

MSC 2010: 65N30, 65M60, 65Y99, 68U20

1 Introduction

Application-oriented research in numerical analysis and scientific computing relies on the availability of research codes that on one hand are able to address a broad range of problems from academic examples for illustration purposes to complex PDE models arising from concrete applications in natural, engineering, and life sciences, and is on the other hand flexible enough to allow for the easy realization of solution algorithms and discretization schemes that have not been on the horizon at the time when the code structure has been designed. This purpose has been served for the last three decades by the KASKADE finite element codes developed at the Zuse Institute Berlin, dating back to (1).

KASKADE 7 (2), started in 2006, is the latest incarnation of these codes, which all share the same mathematical spirit (3) but differ vastly in functionality and software design aspects. It is a C++ library based on the Distributed and Unified Numerics Environment (DUNE) (4–7), and has been used for many diverse research projects ranging from development of numerical methods to real-life problems in various application fields (8–30). Code and documentation snapshots are freely available for noncommercial use at <http://www.zib.de/projects/kaskade7-finite-element-toolbox>.

In this paper, we describe the main ideas and components of the KASKADE 7 toolbox in Sec. 2, illustrating data structures and toolbox usage at short code

*Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, {goetschel | weiser}@zib.de

†Universität Bayreuth, 95440 Bayreuth, Germany, anton.schiela@uni-bayreuth.de

examples. Sec. 3 presents a few more specific components that, while contained in KASKADE 7, do not belong to the toolbox core, and serve as examples of how the code can be extended in different directions. The paper is not intended to be a complete overview, since many software aspects of finite elements are shared by most codes, and widely known. Instead, we focus on those aspects that are nonstandard, or particularly useful, or necessary for understanding the toolbox structure.

2 Toolbox Design

The requirements imposed on a scientific research code are manifold, and to some extent conflicting. In the following section, we therefore discuss the different design goals and their impact on code structure briefly, and prioritize them, which motivates the design decisions made for KASKADE 7. After that, we will describe the resulting library user interface and core structures in more detail. For more details, we refer to the tutorial examples illustrating the basic usage and building blocks of KASKADE 7 applications, and to the comparatively good – for a research code – and extensive class documentation generated via DOXYGEN.

2.1 Design goals

The targeted problems are mainly heterogeneous systems of elliptic and parabolic equations with second order differential operators in two or three space dimensions, including optimal control and inverse problems. The code addresses medium-sized problems and is not intended to solve extremely large scale problems.

Flexibility At the heart of algorithmic research in scientific computing is the continuous improvement of algorithms as well as the development of new approaches. Supporting the inclusion of new algorithmic components is therefore mandatory. This is best achieved by providing a library of loosely coupled components that interact through clear, sufficiently general, and preferably simple interfaces. One example are the DUNE libraries, which allow the use of very different grid managers, each with its own trade-offs on functionality and efficiency, through a single interface. In contrast, software frameworks tend to be rigid, making the implementation of algorithmic structures that are not explicitly supported rather difficult.

Consequently, KASKADE 7 is structured as a library, where suitable components – grid managers, FE discretizations, solvers, preconditioners, I/O – can be selected as needed and combined in many ways. Since the main program is completely in the realm of the library user, any part of the library can be selected or left out, or replaced by tailored components, and combined with application codes in many ways not envisaged during the design of KASKADE 7.

A more specific aspect of flexibility is the support for discretization of heterogeneous PDE systems with different finite element spaces for the different variables, scalar or vectorial. Examples are flow problems, mixed finite elements for solid mechanics, or optimal control problems. For the former, Taylor-Hood

elements are not provided as such, but can be immediately realized as a P2-P1 discretization of the Stokes or Navier-Stokes system.

An early design decision, driven by the requirement of flexibility in algorithmic development, has been to not support distributed memory systems. Instead, shared memory parallelism on multi-core compute servers is used, allowing for a smooth transition from an easily understandable and debuggable sequential programming model to an efficient multithreaded program, which exploits NUMA systems, mixed precision, and SIMD instructions where beneficial (22).

Correctness and generality KASKADE 7 aims at supporting correctness by employing modern C++ 17 features, in particular templates, for detection of programming errors at compile time. This is achieved by providing as much structural information about the problem as possible or practical to the compiler. For example, the vectorial dimension of variables in a PDE system is encoded in the type system. The structure of coefficient vectors in the 2D Stokes system, i.e. $((u_{0x}, u_{0y}), (u_{1x}, u_{1y}), \dots, [p_0, p_1, \dots])$, is not only convention but also known to and enforced by the compiler, which can prevent several kinds of wrong access to coefficients. Similarly, static information about symmetry of stiffness and mass matrices, or their block sparsity pattern in systems of PDEs, can be exploited for detecting errors at compile time.

The usage of C++ templates has the additional benefit of code genericity – the original motivation for templates. Many algorithms in KASKADE 7, from assembly of matrices and right hand sides to solvers and I/O routines, are faced with structurally heterogeneous problems that need to be treated uniformly. An algorithm able to cover both a scalar Poisson problem and the 2D Stokes system without exploiting their special structure will also cope with a mixed formulation of 3D solid mechanics. This – to some extent enforced – genericity is not only beneficial for functionality, but also for correctness. In fact, not only will an implementation debugged on a Stokes example most likely work correctly for Poisson and solid mechanics problems as well, but the need for thinking in more general problem structures often leads to implementations that are well thought-out in the first place. The common experience, formulated very pointedly, is, that once the code compiles, it is semantically correct.

The obvious drawback of the template approach is the one shared by almost all template-heavy libraries: the impact on compile times. To some extent, this is mitigated by providing explicit instantiations in a pre-compiled library and hiding the template definition in special header files, but templates depending on problem structure still need to be recompiled frequently. Thus, build times for the main program are relatively large.

Efficiency While providing high-performance implementations for all kinds of problems is not the main intention, an explicit design goal was not to prevent them. One of the main means for that is, again, C++ templates. Handing down static information on problem structure to the computation-intense algorithmic cores simplifies compiler optimizations such as loop unrolling, vectorization, or dead code elimination.

Functionality One design goal has been to provide a flexible infrastructure on top of which needed functionality can be built easily. In this way, complete-

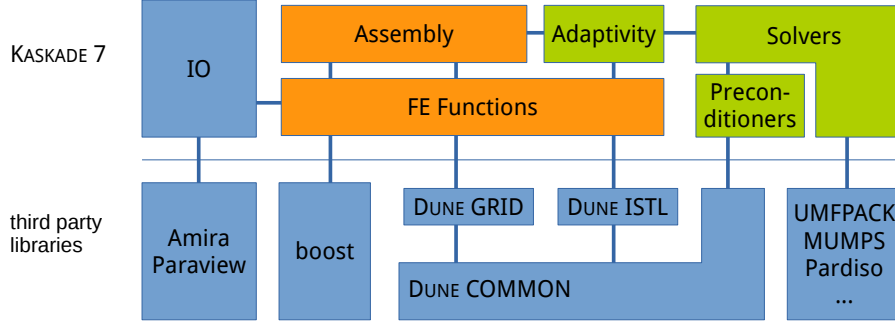


Figure 1: Collaboration and dependencies of the main KASKADE 7 parts.

ness in terms of functionality is a process rather than a design. Consequently, KASKADE 7 functionality has been constantly extended over the years as required by algorithmic and application-driven research, mostly without affecting the basic structure. Today, it features a rich set of discretizations, solvers, preconditioners, error estimators, boundary treatments, and input/output functionality. The top-level structure of the KASKADE 7 modules is illustrated in Fig. 1.

2.2 Problem formulation

For stationary problems, KASKADE 7 addresses variational functionals of the type

$$\min_{u_i \in V_{a,i}} \int_{\Omega} f(x, u_1, \dots, u_{n_a}, \nabla u_1, \dots, \nabla u_{n_a}) dx + \int_{\partial\Omega} g(x, u_1, \dots, u_{n_a}) ds \quad (1)$$

for functions $u_i \in V_{a,i}$ defined on a domain $\Omega \subset \mathbb{R}^d$, as well as more general weak formulations of the type

$$\int_{\Omega} \left(\varphi f_i(x, u_1, \dots, u_{n_a}, \nabla u_{n_a}) + \varphi' \tilde{f}_i(x, u_1, \dots, u_{n_a}, \nabla u_{n_a}) \right) dx + \int_{\partial\Omega} \varphi g_i(x, u_1, \dots, u_{n_a}) ds = 0 \quad (2)$$

for all $i = 1, \dots, n_t$ and test functions $\varphi \in V_{t,i}$. Here, n_a is the number of variables in the system, and n_t the number of equations. Any of the variables and equations can be scalar or vectorial.

The problem definition for (1) consists of providing f , g , and their first and second directional derivatives, as a class adhering to a specific interface, whereas the problem definition for (2) provides f_i , \tilde{f}_i , and g_i . Both problem types use exactly the same interface, in the sense that an implementation for a variational functional (1) is a valid implementation for the first order necessary conditions interpreted as weak problem (2). The numbers n_a of variables and n_t of equations as well as their respective dimension $m_{a,j}$ and $m_{t,i}$ are specified as compile time constants.

A problem class defines two mandatory member classes, the `DomainCache` defining f and the `BoundaryCache` defining g . The domain cache provides mem-

ber functions `d0`, `d1`, and `d2` evaluating $f(\cdot)$, $f'(\cdot)v_i$, and $f''(\cdot)[v_i, w_j]$, respectively. The boundary cache is defined analogously. Both `DomainCache` and `BoundaryCache` can be derived from base classes providing default implementations for simple cases. These can, of course, be overwritten by the problem definition class. This gives greater flexibility than, e.g., requiring the user only to specify coefficient functions for fixed types of equations, but is slightly more involved. However, for various standard problems, helper classes for common differential operators are provided as building blocks.

As a guiding example, we consider the problem of linear elasticity, i.e. the variational functional

$$\min_{u \in H^1(\Omega)} \int_{\Omega} W(E) dx - \int_{\Sigma} z(s) u(s) ds$$

where u is a vector-valued displacement in $2D$ or $3D$, and $E = \frac{1}{2}(u_x + u_x^T)$. Dead load forces z acting on $\Sigma \subset \Omega$ (e.g., gravity), or the boundary $\Sigma \subset \partial\Omega$ lead to a nontrivial displacement. The elastic energy $W = \frac{\lambda}{2}\text{tr}(E)^2 + \mu(E : E)$ is implemented by the convenience class `LameNavier`, which makes use of helper classes defining the St. Venant-Kirchhoff material law as well as the linearized Green-Lagrange tensor E . Various other material laws, including material parameters for some commonly used materials, as well as, e.g., membrane models for cardiac electrophysiology, are provided by KASKADE 7. The essential parts of the problem class and the `DomainCache` are shown in Listing 1.

```

template <class VarSet>
class ElasticityFunctional: public Kaskade::FunctionalBase<VariationalFunctional>
{
public:
    using Scalar = double;
    static int const dim = AnsatzVars::Grid::dimension;
    using ElasticEnergy = Kaskade::Elastomechanics::LameNavier<dim,Scalar>;
    //...

    class DomainCache
    {
    public:
        //...
        template <class Position, class Evaluators>
        void evaluateAt(Position const& x, Evaluators const& evaluators)
        {
            energy.setLinearizationPoint(
                at_c<u_idx>(vars.data).derivative(at_c<u.Space_idx>(evaluators)));
        }

        Scalar d0() const
        { return energy.d0(); }

        template<int row>
        Vector d1 (VariationalArg<Scalar,dim> const& arg) const
        { return energy.d1(arg); }

        template<int row, int col>
        Matrix d2 (VariationalArg<Scalar,dim> const& argTest,
                    VariationalArg<Scalar,dim> const& argAnsatz) const

```

```

    { return energy.d2(argTest,argAnsatz); }

private:
    typename AnsatzVars::VariableSet const& vars;
    ElasticEnergy energy;
};
};

```

Listing 1: Variational functional/DomainCache for elasticity example

We prescribe three kinds of boundary conditions for this example as shown in Listing 2: on the top face of the unit cube domain, natural boundary conditions are set, on the bottom we apply a given normal stress, and the side faces are fixed at their position (homogeneous Dirichlet conditions). The distinction between boundary regions is provided by the DUNE grid interface.

```

class BoundaryCache : public CacheBase<ElasticityFunctional,BoundaryCache>
{
public:
    //...
    void moveTo(Faceliterator const& face)
    {
        switch (face->boundarySegmentId())
        {
            case 0: // top face: homogeneous Neumann, zero normal stress
                alpha = 0;
                beta = 0;
                break;
            case 1: // bottom face: inhomogeneous Neumann, given normal stress
                alpha = 0;
                beta = up;
                break;
            case 2: // side face: essential boundary conditions
                alpha = 1e14; // requires large penalty for hard materials
                beta = 0;
        }
    }

    template <class Evaluators>
    void evaluateAt(Dune::FieldVector<ctype,dim-1> const& x,
                  Evaluators const& evaluators)
    { u = component<u_idx>(vars).value(at_c<u_Space_idx>(evaluators)); }

    Scalar d0() const
    { return alpha*(u*u) - beta*u; }

    template<int row>
    Vector d1(VariationalArg<Scalar,dim,dim> const& argTest) const
    { return 2*alpha*(u*argTest.value) - beta*argTest.value; }

    template<int row, int col>
    Matrix d2(VariationalArg<Scalar,dim,dim> const &argTest,
              VariationalArg<Scalar,dim,dim> const &argAnsatz) const
    { return 2*alpha*(argTest.value*argAnsatz.value); }

```



```

private:
  typename AnsatzVars::VariableSet const& vars;
  Vector u, beta;
  Scalar alpha;
};

```

Listing 2: BoundaryCache for elasticity example

Boundary conditions are in general formulated as Robin-type conditions, with the limit case of Dirichlet conditions incorporated by a simple penalty formulation or via Nitsche’s method (31).

2.3 At the core: finite element function spaces

At the very heart of finite element codes is the mesh, representing a triangulation $\mathcal{T} = \{T_i \mid i = 1, \dots, m\}$ of the computational domain Ω . On \mathcal{T} , finite element spaces are defined. The mesh cells T_i are images of a reference cell T_{ref} under the smooth bijective (and usually affine) coordinate transform $x_T(\xi)$ for $\xi \in T_{\text{ref}}$. As the actual cell T is clear from the context, we will omit the subscript T in the coordinate transform. Currently, the use of simplicial and quadrilateral reference cells is supported in KASKADE 7.

Representation concept Finite element functions are stitched together from linear combinations of *shape functions* ϕ_j defined on T_{ref} . In KASKADE 7, different scalar and vector-valued shape function sets $\Phi = \{\phi_j \mid j = 1, \dots, n_\phi\}$ are defined. Scalar examples are the ubiquitous complete Lagrange polynomials $\Phi_p^L \subset \mathbb{P}_p$ of arbitrary order $p \geq 0$ on barycentric Lobatto nodes $\xi_j \in T_{\text{ref}}$ (32), such that $\phi_j(\xi_k) = \delta_{jk}$, or symmetric hierarchical polynomial bases Φ_p^H with $\mathbb{P}_p \subset \text{span} \bigcup_{k=0}^p \Phi_k^H \subset \mathbb{P}_{p+1}$ (33) allowing the construction of hierarchical error estimators. An example of vectorial shape functions are those for edge elements (34).

The restriction $\varphi_i|_T$ of finite element *basis functions* φ_i to a cell T are defined in terms of the shape functions ϕ_j as

$$\varphi_i|_T(x) = \sum_j K_{T,ij} \psi_T(x, \phi_j(\xi(x))), \quad x \in T.$$

The value of vectorial basis functions often depends on the coordinate mapping $x(\xi)$. This dependence ψ_T is implemented as a special converter class for each FE space. For scalar basis functions this is usually just the identity. Finally, the linear combination of shape function values by K_T is realized by a special class for each defined FE space for performance reasons: In many cases, K_T has a special structure that can be exploited statically. For example, it is just the identity for Lagrange finite elements, and a diagonal matrix with entries ± 1 for hierarchical ansatz functions.

This factorized definition of FE basis functions φ_i allows to reuse generic shape function implementations. As an example, nonconforming elements for biharmonic problems can immediately be defined in terms of standard Lagrange shape functions.

Example 2.1 (Morley elements). A minimal, nonconforming discretization of 2D biharmonic problems with piecewise quadratic basis functions on triangular

grids can be defined by requesting continuity at grid vertices and continuity of normal derivatives at edge midpoints (35). This yields six interpolation conditions for the linear combination of shape functions. From these conditions, the combiner matrix K_T can be computed. For two reasons, K_T depends on the actual cell T : derivatives are affected by the mapping from reference to actual cell, and the normal derivatives need to be oriented, e.g., pointing towards the cell with smaller index. Listing 3 shows the implementation of the combiner's constructor, which computes K_T by first creating its inverse, i.e., evaluating values and derivatives of Lagrangian shape functions on vertices and edge midpoints, respectively, and then inverting it. While this computation is certainly not the most efficient, it is simple, straightforward, and a rather general blueprint for other finite elements.

```

Combiner(Cell const& cell, GridView const& gv, IndexSet const& is) {
    Converter psi(cell);
    auto const& sfs = lagrangeShapeFunctionSet<ctype,dim,Scalar>(cell.type(),2);
    auto const& Tref = Dune::ReferenceElements<ctype,dim>::simplex();

    // Evaluate oriented normal derivatives  $\pm\phi'(\xi)\xi'(x)n$  of ansatz functions
    // on all edge midpoints.
    for (auto edge=gv.ibegin(cell); edge!=gv.iend(cell); ++edge) {
        auto n = edge->centerUnitOuterNormal();
        int j = edge->indexInside(); // cell-local edge index
        auto xi = Tref.position(j,1); // edge midpoint in cell-local coordinates
        psi.setLocalPosition(xi);
        auto sign = (edge->neighbor() && is.index(edge->outside()) < is.index(cell)) ?
            -1.0: 1.0; // for orienting the normal derivative

        // Compute and enter all ansatz functions' oriented normal derivatives.
        for (int i=0; i<sfs.size(); ++i) {
            VariationalArg<Scalar,dim,1> dsf(0,sfs[i].evaluateDerivative(xi));
            VariationalArg<Scalar,dim,1> daf = psi.global(dsf);
            K[j][i] = sign * (daf.derivative[0] * n);
        }
    }

    // Evaluate shape function values  $\phi(\xi)$  at the vertices
    // (no converter needed for values).
    for (int j=0; j<3; ++j) {
        auto xi = Tref.position(j,2); // vertex position in cell-local coordinates

        // Compute and enter all shape function's values.
        for (int i=0; i<sfs.size(); ++i)
            K[j+3][i] = sfs[i].evaluateFunction(xi);
    }

    K.invert();
}

```

Listing 3: Constructor of the Combiner class for Morley elements.

Fig. 2 shows a simulation result using Morley elements. A rectangular piece of denim cloth is fixed on two boundaries and deforms under gravity. We use a 2D shell model derived from the 3D material law, and a damped Newton

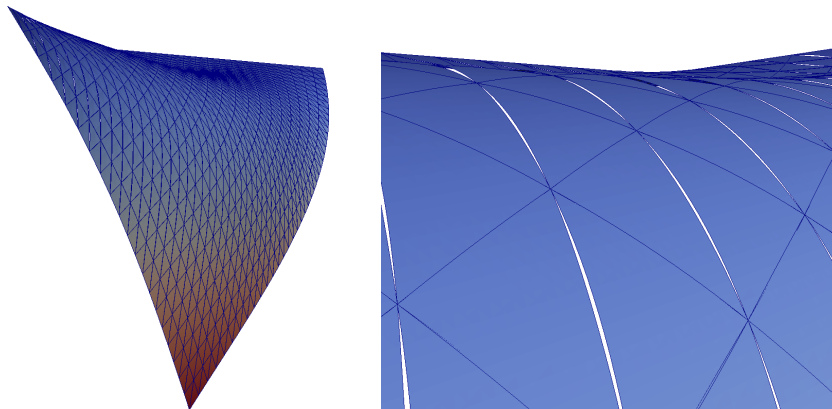


Figure 2: Cloth simulation using Morley elements. Left: a $0.2 \times 0.1 \text{ m}^2$ piece of denim fixed on two boundaries deforming under gravity. Right: zooming in on the top left part on a deliberately coarse grid shows the nonconforming deformation.

method for minimization of the fourth order shell energy.

Mesh refinement FE functions are given in terms of their coefficient vectors with respect to a basis $(\varphi_i)_{i=1,\dots,N}$, with degrees of freedom associated to mesh vertices, edges, faces, or cells depending on the FE space. Whenever the mesh is modified, e.g., by adaptive refinement or coarsening, the basis is changed, rendering the coefficient vectors useless. In order to keep the FE function despite the mesh modification, the coefficient vectors need to be transferred to the new basis – a process easy to get wrong if several FE functions of different type are around. To keep track of them, the callback mechanism provided by Boost.Signals2 (36) is used. First, a `GridManager` class assumes ownership of the grid – one of the grid implementations with DUNE interface – and provides read-only access to it as well as methods for modifying operations. Thus, the grid manager controls the state of the grid and can inform registered objects via a callback about upcoming mesh modifications and their completion, see Fig. 3.

Finite element spaces register themselves at the grid manager. On mesh modification, they store sufficient information about the old grid state to compile a basis change matrix on completion of the mesh modification. This basis change is forwarded to all objects registered at the space via a second callback. Finite element functions register themselves at their spaces, and apply the basis transform to their coefficient vectors. This automatic transfer of finite element functions to refined or coarsened meshes has proven to be extremely convenient, and eliminates a substantial source of bugs.

Mesh refinement is usually controlled by error estimators. KASKADE 7 provides hierarchical (1) and embedded error estimators, as well as gradient averaging and an error estimator in the flavor of Bank and Weiser (37). Moreover, various refinement criteria are provided to specify which cells to mark for refinement. The embedded error estimator is particularly easy to use. Having computed an approximate solution `sol`, the FE function is projected onto the

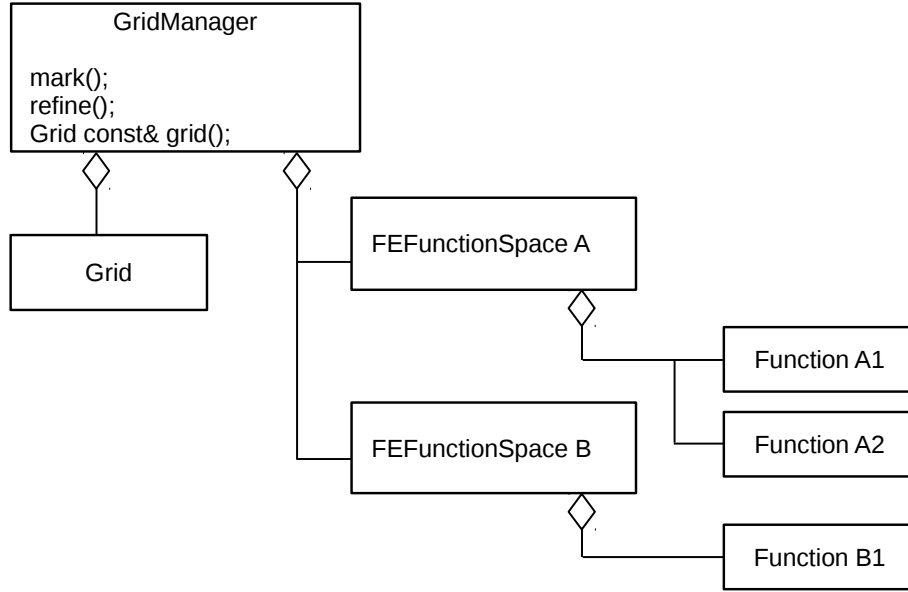


Figure 3: Callback collaboration diagram for mesh refinement and coarsening.

the polynomial ansatz subspace of one order lower to get an error approximation. The hierarchic projection is defined by the shape function sets on each cell accessible through the `variableSet`.

```

auto err = sol;
projectHierarchically(variableSet,err);
err -= sol;
  
```

With this and a refinement criterion, the `EmbeddedErrorEstimator` class handles the refinement, e.g.,

```

EmbeddedErrorEstimator<VariableSetDescription> estimator(gridManager,varSetDesc);
FixedFractionCriterion marker(0.1);
estimator.setTolerances(tol).setCriterion(marker);
bool accurateEnough = estimator.estimate(err,sol);
  
```

The flexibility of KASKADE 7 allows to quickly implement own error estimators, e.g., for goal-oriented error estimation in optimal control problems (25).

2.4 Assembly

One of the main tasks in FE computation is the assembly of right hand sides and stiffness matrices for variational problems and weak formulations of PDE systems as formulated in (1) and (2). The assembler creates a right hand side as a heterogeneous container of coefficient vectors, i.e. of type

```

boost::fusion::vector<Dune::BlockVector<Dune::FieldVector<double,mt1>>,
...,
Dune::BlockVector<Dune::FieldVector<double,mtnt>>>>.
  
```

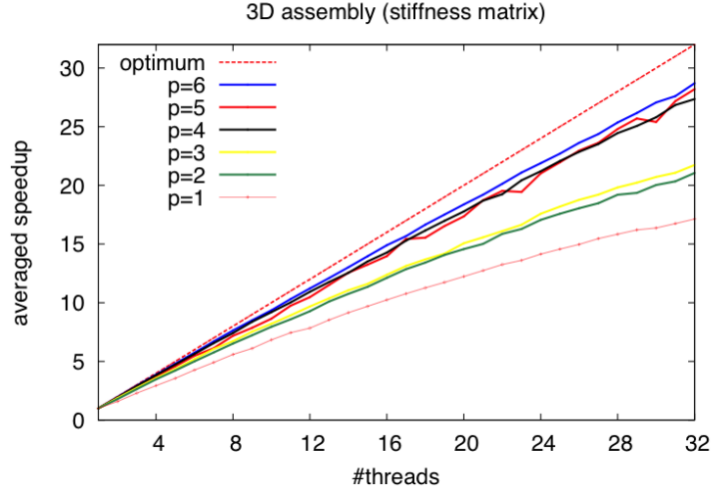


Figure 4: Strong scaling of stiffness matrix assembly for the Poisson problem on a unit cube, computed on a 8-socket compute server with AMD Opteron 8384. For ansatz order $p \geq 4$, the parallel efficiency reaches reasonable 85% even when all 32 cores are used.

Here, $mt_1 \dots mt_{nt}$ are the equations' dimensions $m_{t,1}, \dots, m_{t,n_t}$. The Boost.Fusion meta programming library (36) allows to translate the static information provided to the assembler into the right hand side container type in a completely generic way.

Stiffness matrices for systems of PDEs are $n_a \times n_t$ block matrices with sparse blocks A^{ij} , where the block entries are $m_{t,i} \times m_{a,j}$ matrices. Of course, not all blocks are present, e.g., there is no p - p block in the Stokes system, and the whole block matrix may be symmetric, e.g., in optimization problems. Thus, stiffness matrices are represented in KASKADE 7 as heterogeneous lists of actually present blocks, omitting those that are just transposed copies of other blocks due to symmetry. As an example, the type of the Stokes stiffness matrix is essentially

```
boost::fusion::vector<NumaBCRSMMatrix<Dune::FieldMatrix<double,dim,dim>>,
    NumaBCRSMMatrix<Dune::FieldMatrix<double,1,dim>>>>.
```

Again, this list of sparse matrices can be extracted generically from the compile-time information about the PDE system provided by the library user. The `NumaBCRSMMatrix<Entry>` type is a compressed sparse row storage with potentially matrix-valued entries of type `Entry`, which distributes its rows in blocks over the available NUMA memory banks. The grid cells are split in ranges of roughly equal size, and each range is worked on by one thread. Local stiffness matrices are computed and stored in thread-local buffers that fit into the second level caches. When filled, the buffers' contents are scattered to the global stiffness matrix, using individual locking of several matrix row blocks.

With this approach, the assembly of stiffness matrices scales reasonably well, see Fig. 4. The simple constant-coefficient Poisson problem has a particularly low arithmetic density, in particular for low finite element ansatz order. With growing order, the number of quadrature points treated locally increases, and

with it the arithmetic intensity. Consequently, higher orders lead to better scalability.

2.5 Solvers

The tools described above enable to formulate given infinite dimensional problem of the form (1) and (2) as a finite dimensional problems in \mathbb{R}^n . If the problem is linear, then after assembly, linear solvers can be applied. Otherwise, nonlinear solution algorithms (e.g., Newton's method) have to be used, which typically turn the nonlinear problem into a sequence of linearized problems, each of which has to be assembled and solved.

For the solution of the resulting linear systems, KASKADE 7 contains wrappers for some well-known direct solvers (MUMPS¹, UMFPACK², SUPERLU³) adhering to the DUNE interface:

```
class InverseLinearOperator : public Dune::LinearSolver<Domain,Range>;
```

For larger problems, KASKADE 7 not only allows to use iterative solvers and preconditioners as provided by DUNE (CG, BiCGStab, MINRES, GMRES), but provides own functionality for the solution of linear systems. This includes solvers like an Uzawa solver for saddle point systems and preconditioned conjugate gradients (PCG) `Pcg` with several termination criteria, but also preconditioners. There, both stationary iterations (e.g., `JacobiPreconditioner`) and multigrid methods for arbitrary finite element order on simplicial grids are available. There are general multiplicative V-cycles and additive multigrid methods, as well as several convenience functions for defining concrete preconditioners by specifying particular ingredients like smoothers and coarse grid preconditioners. Solving the elasticity example from above using PCG with a BPX preconditioner (named after Bramble, Pasciak, and Xu (38)) is shown in Listing 4. There, after defining some data types, the linear operator formed by the assembled stiffness matrix `A` is wrapped as a symmetric linear operator `sa` with a dual pairing `dp`. Using an absolute energy error termination criterion, the system with given right-hand side `rhs` is then solved and the solution stored in the variable `u`. Statistics about the solution, like number of iterations and convergence rate, are returned in the variable `res`.

```
using X = Dune::BlockVector<Dune::FieldVector<double,dim>>;
using Matrix = NumaBCRSMatrix<Dune::FieldMatrix<double,dim,dim>>;
using LinOp = Dune::MatrixAdapter<Matrix,X,X>;
SymmetricLinearOperatorWrapper<X,X> sa(LinOp(A),dp);
PCGEnergyErrorTerminationCriterion<double> term(atol,maxit);
std::unique_ptr<SymmetricPreconditioner<X,X>> mg;
mg = moveUnique(makeBPX(Amat,gridManager));
Dune::InverseOperatorResult res;
Pcg<X,X> pcg(sa,*mg,term);
pcg.apply(u,component<0>(rhs),res);
```

Listing 4: PCG with BPX preconditioner.

¹<http://mumps.enseeiht.fr/>

²<http://faculty.cse.tamu.edu/davis/suitesparse.html>

³<https://portal.nersc.gov/project/sparse/superlu/>

More advanced methods are available as well, like overlapping Schwarz smoothers with low/mixed precision storage of matrix entries (22), and can be constructed in an equally easy fashion using the `makePBPX` helper function.

Solvers for nonlinear problems can build upon this functionality in several ways. Simple algorithms, like a local Newton method can be implemented in an ad-hoc fashion. For more sophisticated methods, which combine, e.g., globalization, inexact linear solvers, and adaptivity, it is helpful to access KASKADE 7 functionality via an abstract interface. For example, a composite step method (16) for the solution of nonlinear optimal control problems has been implemented within the C++-library `Spacy`⁴ that provides such an interface.

2.6 I/O

Several grid types provided by the DUNE interface can be used in KASKADE 7. Simplicial and cartesian cell types in 1D, 2D and 3D are supported, e.g., triangles, tetrahedra, quadrilaterals and hexahedra. Grid generation for simple shapes like L-shaped domains or Platonic solids is available for testing purposes, while more interesting grids can be imported from several grid file formats, e.g., from `.poly` files generated by TRIANGLE (39) in 2D, VTK files (40), or from Hypermesh files exported by AMIRA (41). Moreover, the DUNE Grid Format (DGF) allows importing meshes in various other formats, like GMSH (42).

Visualization of solutions is supported by output to VTK files, in conforming or nonconforming mode depending on the FE function's continuity (see Fig. 2), for further postprocessing in ParaView (43), as well as in the AMIRA file format. Additionally, mesh and solution can be written to files for later visualization using GNUPLOT. Basic checkpoint/restart capabilities especially for time-dependent problems are given by the option to read VTK files and extract meshes and FE functions in order to continue computations from such a snapshot.

2.7 Instationary problems

KASKADE 7 provides two possibilities to handle time-dependent problems of the form $B(u)\dot{u} = f(u)$: an extrapolated linearly implicit Euler method (LIMEX) (44), i.e., a method of time layers approach (Rothe's method), and spectral deferred corrections (SDC) (45) with a method of lines discretization. For the former, given an evolution equation Equation eq, the corresponding integration loop including estimation of the time discretization error is shown in Listing 5.

```

Limex<Equation> limex(gridManager,eq,variableSet);
for (steps=0; !done && steps<maxSteps; ++steps) {
  do {
    dx = limex.step(x,dt,extrapolOrder,tolX);
    errors = limex.estimateError(/*...*/);
    // ... (choose optimal time step size)
  } while( error > tolT );
  x += dx ;
}

```

Listing 5: LIMEX time integration loop

⁴<https://spacy-dev.github.io/Spacy/>

For step computation, the stationary elliptic problem resulting from the linearly implicit Euler method have to be provided. This is done with help of the class `SemImplicitEulerStep`, and requires only an additional method `b2` in the `DomainCache` for the evaluation of B . SDC methods are further discussed in Sec. 3.

3 Specific Extensions

In this section, we discuss some non-core functionalities provided in KASKADE 7, which are specific for application domains or particular types of PDE problems. Besides illustrating the range of problems to which KASKADE 7 has been applied, they serve as examples of how the toolbox core has been extended.

3.1 Spectral deferred correction methods for time integration

Spectral deferred correction methods (SDC) (45) are fixed point iteration methods to solve ODE collocation systems. Each iteration, or *sweep*, consists of stepping through collocation nodes by a low-order scheme, yielding a high-order method. To be more precise, consider the reaction-diffusion system

$$B\dot{u} = Au + Bf(u)$$

coming from a method of lines finite element discretization of the underlying PDE, with mass matrix B and stiffness matrix A . With collocation time discretization on a time grid t_0, \dots, t_M , abbreviating $u(t_i) = u_i$, and applying a suitable quadrature rule, this system is transformed to

$$B(u_{i+1} - u_i) = \sum_{j=0}^M S_{ij}(Au_j + Bf(u_j)). \quad (3)$$

KASKADE 7 provides time grids for collocation (Lobatto, Gauss, and Radau points) derived from a common abstract base class `SDCTimeGrid`, together with Lagrange or Hermite polynomial interpolation, quadrature, and differentiation. As time-global coupling makes system (3) hard to solve, the quadrature coefficients S are approximated by a triangular matrix \hat{S} , e.g., using an Euler time stepping scheme or the LU factorization trick (46). Functions performing a single SDC sweep as well as helper methods computing optimized integration matrices are contained in KASKADE 7.

SDC integrators have been applied, e.g., to the electrical excitation of myocardial tissue, described by the monodomain equations (47):

$$\begin{aligned} C\chi\dot{u} &= \operatorname{div}(\sigma\nabla u) - I_{\text{ion}}(u, w) \\ \dot{w} &= f(u, w). \end{aligned}$$

This reaction-diffusion equation for the transmembrane voltage u , coupled to pointwise ODEs for the action of various ion channels, describes the propagation of activation and deactivation fronts traveling through the tissue. Activation initiates creation of tensile stress leading to contraction of the heart. Since the stretch induced by the contraction affects the behavior of stretch-activated

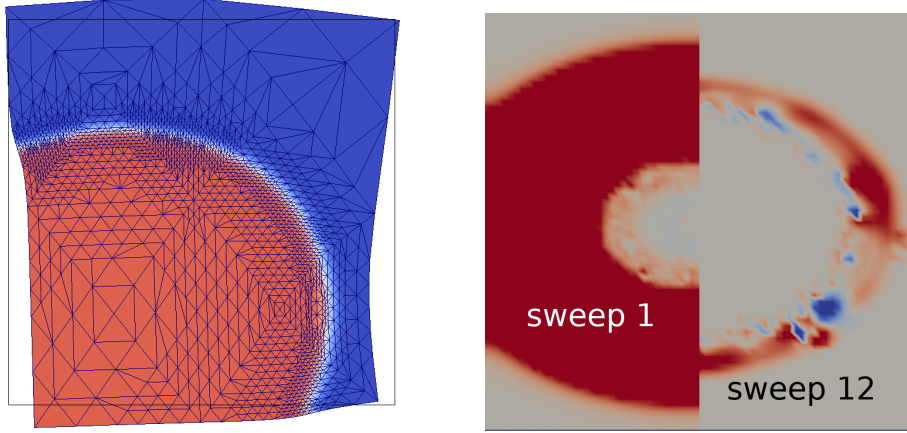


Figure 5: *Left*: Testcase for adaptive simulation of cardiac electrical excitation coupled to mechanical contraction. Stretch-activated ion channels affect the excitation front speed. *Right*: Comparison of effective support of different SDC sweeps for the same time step of expanding activation front.

ion channels, the front speed depends weakly on the mechanics. In situations where this effect cannot be neglected, electrophysiology and mechanics need to be solved jointly. Here, the SDC iteration can be nicely interleaved with mechano-electrical coupling as well as with mesh refinement, see Fig. 5, left, and (30).

In moving front systems like the monodomain equations, the converging SDC iteration leads not only to geometrically decreasing corrections, but also the effective spatial support of the corrections is shrinking, see Fig. 5, right. Restricting the computation of SDC sweeps to their effective support reduces the computational effort for SDC iterations significantly and realizes a certain kind of multirate integration.

Spectral deferred correction methods are also the basic ingredient of the Parallel-in-Time method PFASST (48), which has been used in KASKADE 7 to investigate the impact of MPI communication on hybrid Parareal methods (9).

3.2 Optimal control of nonlinear problems

The KASKADE 7 toolbox has been used in a couple of projects concerned with optimal control of nonlinear PDEs. These are minimization problems of the following general form:

$$\min J(y, u) \text{ s.t. } A(y) - Bu = 0, \quad (4)$$

where J is a given objective functional, $A(y) - Bu = 0$ a nonlinear PDE with state y and control u . Typically, A is a nonlinear differential operator and the operator B models the influence of the control. Introducing a Lagrange parameter p a Lagrangian function $L(y, u, p) := J(y, u) + \langle p, A(y) - Bu \rangle$ can be

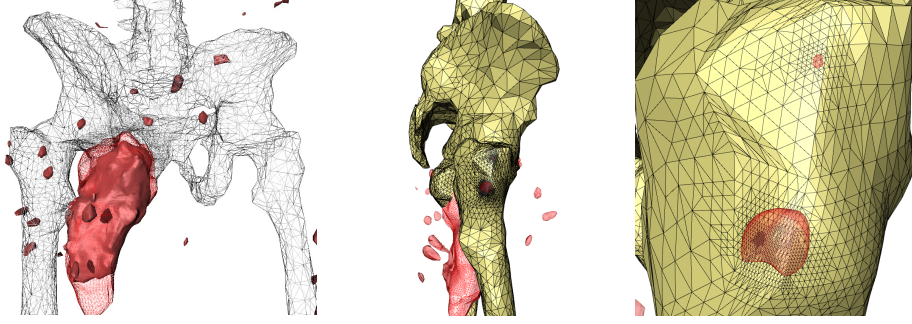


Figure 6: *Left:* Optimal temperature distribution, 43°C isothermal surface. *Middle and Right:* adaptively refined grid. Critical regions, where upper temperature bound is close, are refined.

defined, and KKT-conditions can (at least formally) be derived:

$$\begin{aligned} J_y(y, u) + A'(y)^*p &= 0 \\ J_u(y, u) - B^*p &= 0 \\ A(y) - Bu &= 0. \end{aligned} \tag{5}$$

This is a system of operator equations, which can be rewritten in the form (2). Discretization of this system and its linearization can be performed in KASKADE 7 after suitable finite element spaces for y , u , and p have been chosen. A function space oriented SQP-method for the solution of such problems has been proposed in (16) and applied to various nonlinear optimal control problems. This method behaves like a Newton method locally, but its globalization scheme encourages functional descent and this yields more robust convergence behaviour towards local minimizers of (4) than a damped Newton method, applied to (5), which does not distinguish local minimizers from other stationary points.

As a specific application we discuss a hyperthermia treatment planning problem (8). In deep regional hyperthermia a tumor is heated by a microwave applicator. The corresponding planning problem is to find optimal antenna parameters which result in maximal heating of the tumor, while not exceeding a specified maximal temperature in healthy tissue. In this context the control u are the antenna parameters and the state y is the temperature distribution inside the tissue. The PDE constraint consists of a combination of the time-harmonic Maxwell equation for the microwave antenna fields, coupled to a nonlinear heat equation. The required upper bounds on the temperature are tackled by an interior point path-following approach (21; 49) that also includes adaptive grid refinement, that is specifically tailored to state constrained problems, see Fig. 6.

3.3 Contact problems

A recent addition to KASKADE 7 is the treatment of frictionless multi-body contact problems. The usual linearized nonpenetration constraint

$$n^T(u(x) - u(x')) \leq g(x)$$

for the displacement u along the outer normal n at x with potential contact partner x' and gap g is discretized by a segment-to-segment approach with an arbitrary

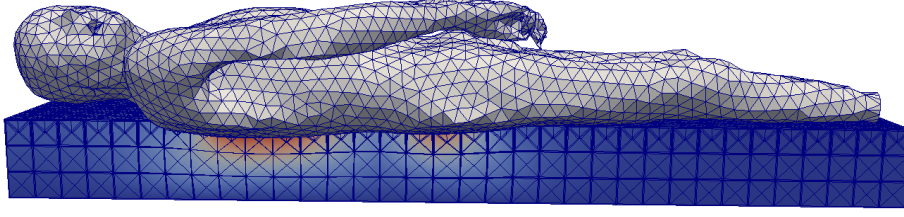


Figure 7: Contact between corpse and mattress for investigating the impact of corpse positioning and contact area on cooling.

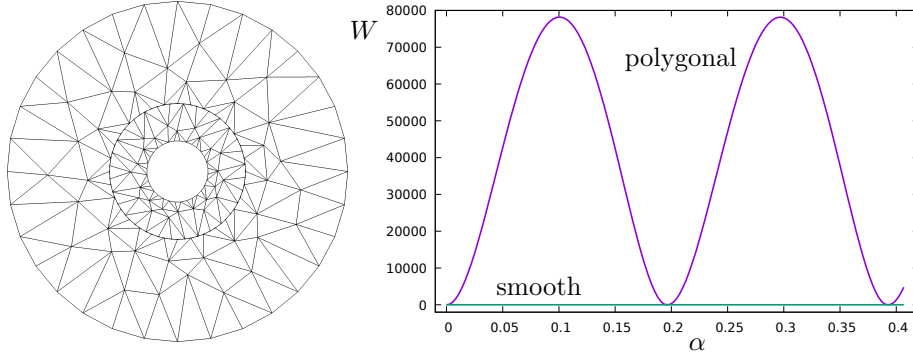


Figure 8: Two concentric annuli of total diameter 8, coarsely discretized as regular polygons, rotating against each other. *Left:* Coarse FE discretization with 32 vertices around the perimeter. The contact boundary vertices are located on two circles with 10^{-4} difference in radius. The inner annulus is rotated by half an edge length. *Right:* Deformation energy W with gap function computed on polygonal boundary (top) and on G^1 -continuous interpolation (bottom) versus rotation angle α . The lower energy line is below 10^{-17} .

bitrary number of samples per boundary face. Contact partners are found by ray-volume intersection queries in a spatial search tree provided by Boost.Geometry. This overconstrained contact formulation leads to quadratic programs of the form

$$\min_x \frac{1}{2} x^T A x + c^T x \quad \text{s.t.} \quad Bx \leq b$$

to be solved. As a contact solver, a nonlinear multigrid method with primal grid hierarchy as outlined in Sec. 2.5 is used in combination with overlapping block-QP solvers for smoothing on every level.

One application is in estimating the time of death in forensic medicine. The contact area between corpse and support (Fig. 7) affects the heat transfer into the environment, and therefore the cooling of the corpse. This in turn has an impact on rectal temperature readings and the estimated time since death (26).

Surface-to-surface discretizations are prone to contact locking (Fig. 8), for which the computation of gap functions based on G^1 -continuous surface interpolation on quadrilaterals has been proposed as remedy (50). KASKADE 7 provides a G^1 -continuous interpolation on triangles (51), and extends this smoothly into the volume. Feature edges can be specified or be detected automatically by angle thresholding and allow to include sharp curves separating smooth surface

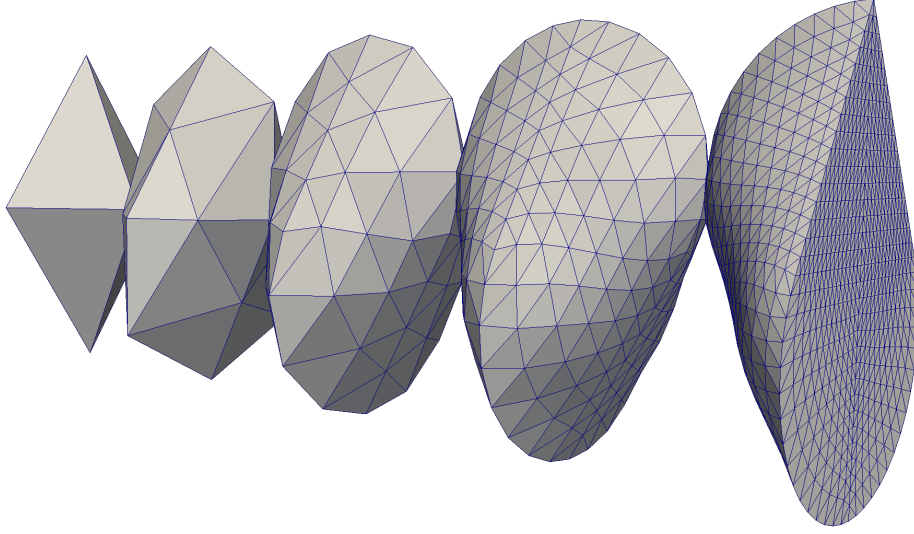


Figure 9: A stretched octahedron refined towards an oloid-like shape with G^1 -continuous surface except for the two feature curves. Four feature edges and normals at the four endpoints of the feature curves have been specified.

patches, see Fig. 9 for an illustrative example. As shown in Fig. 8, right, the boundary smoothing removes surface locking completely.

3.4 Lossy compression of finite element functions

For sufficient accuracy, PDE systems often need to be discretized with a huge number of spatial degrees of freedom. Large scale PDE simulations thus involve tremendous amounts of data that need to be stored or transmitted to other compute nodes, such that communication bandwidth and storage space need to be considered to obtain fast and efficient methods. To deal with these challenges, KASKADE 7 contains methods for lossy compression of finite element solutions with low computational overhead. It makes use of the grid hierarchy coming from uniform or adaptive mesh refinement and is based on transforming the finite element solution from the nodal basis to the hierarchical basis (52), or other wavelet bases, together with error-controlled quantization of the coefficients (12; 29). A priori error estimates for compression factors (29) show that asymptotically 2.96 bits/value (in 2D, compression factor 21.6 compared to double precision, slightly higher compression factor in 3D) are sufficient to achieve a reconstruction error equal to L^∞ -interpolation error bounds for functions with sufficient regularity. Using these tools in KASKADE 7 is straightforward:

```
LossyStorage<Grid, CoefficientVector> lossyStorage(coarseLevel);
lossyStorage.setup(gridManager.grid());
```

sets up the compression routines, e.g., computes, if required, prolongation matrices between grid levels, starting from `coarseLevel`. Afterwards,

```
lossyStorage.encode(gridManager.grid(), coeffVector, compressedData, quantTol);
```

compresses data given in `coeffVector` to an output byte stream `compressedData` using quantization tolerance `quantTol`. Reconstruction is done calling the method `lossyStorage.decode` with similar arguments. Working directly on the coefficient vector `coeffVector` with its type given by the template parameter `CoefficientVector` uncouples compression from the KASKADE 7 data structures for finite element functions and allows its use in combination with other DUNE-based FE discretization modules, e.g., `dune-PDELab`⁵.

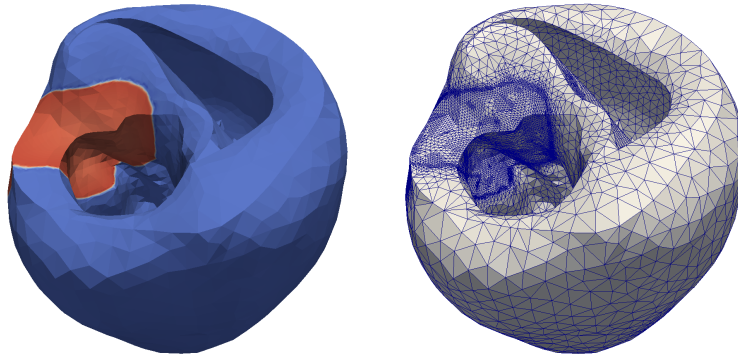


Figure 10: Reconstructed 3D solution 15.3 ms after excitation. Left: transmembrane voltage (compression factor 15.4, relative L^∞ -error 10^{-4}). Right: adaptive computational grid (403,192 vertices).

Figure 10 shows a snapshot of a 3D simulation of cardio-electrophysiology, with transmembrane voltage v and gating variable w governed by the Rogers-McCulloch variant of the monodomain equations (53), with the heart geometry from (54). A local, short excitation pulse of 1 ms leads to a depolarization wave traveling through the domain, requiring local adaptivity in time and space for efficient simulation. In this snapshot, the mesh consists of 2251 410 elements/403 192 vertices on 5 levels. Encoding of the 806 384 degrees of freedom took 4.9s, much less than the computation time required for the time step. We achieved a compression factor of 15.4 for a relative L^∞ -error of order 10^{-4} in the transmembrane voltage v and 10^{-2} in the gating variable w .

4 Conclusions and Outlook

KASKADE 7 is a versatile, efficient, and, most of all, flexible finite element library, meeting the needs of algorithmic research in numerical solution of PDEs and related problems. Its structure has proven to be flexible enough to meet virtually all requirements that showed up over the last 15 years, even if not anticipated in the design phase. The downside of the provided flexibility is a steeper learning curve than required for frameworks targeting higher abstraction levels.

The toolbox is continuously improved and extended. Present directions for extension, following current research and computing trends, are green computing by using reduced precision and lossy data compression, time-parallel and

⁵<https://dune-project.org/modules/dune-pdelab/>

space-time discretizations, and domain specific languages for supporting modern architectures.

Acknowledgment

Implementing a flexible FE toolbox is a major effort, and requires to involve several people. We are grateful for contributions by B. Erdmann, L. Lubkoll, M. Moldenhauer, R. Roitzsch, J. Schneck, L. Weimann, and F. Wende, as well as several master students and interns. To a significant fraction, KASKADE 7 has been developed within several third-party funded research projects. We acknowledge support in particular from DFG, BMBF, and ECMath, recent projects involved being DFG WE 2937/9-1, ECMath CH9, as well as BMBF 01IH16005D.

References

- [1] P. Deuffhard, P. Leinen, H. Yserentant, Concepts of an adaptive hierarchical finite element code, *IMPACT Comput. Sci. Engrg.* 1 (1989) 3–35.
- [2] S. Götschel, M. Weiser, A. Schiela, Solving optimal control problems with the Kaskade 7 finite element toolbox, in: A. Dedner, B. Flemisch, R. Klöforn (Eds.), *Advances in DUNE*, Springer, 2012, pp. 101–112.
- [3] P. Deuffhard, M. Weiser, *Adaptive numerical solution of PDEs*, de Gruyter, 2012.
- [4] M. Blatt, P. Bastian, The iterative solver template library, in: B. Kagström, E. Elmroth, J. Dongarra, J. Wasniewski (Eds.), *Applied Parallel Computing – State of the Art in Scientific Computing*, Springer, Berlin/Heidelberg, 2007, pp. 666–675.
- [5] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöforn, M. Ohlberger, O. Sander, A generic grid interface for parallel and adaptive scientific computing. Part I: Abstract framework, *Computing* 82 (2–3) (2008) 103–119.
- [6] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöforn, R. Kornhuber, M. Ohlberger, O. Sander, A generic grid interface for parallel and adaptive scientific computing. Part II: Implementation and tests in DUNE, *Computing* 82 (2–3) (2008) 121–138.
- [7] M. Blatt, A. Burchardt, A. Dedner, C. Engwer, J. Fahlke, B. Flemisch, C. Gersbacher, C. Gräser, F. Gruber, C. Grüninger, D. Kempf, R. Klöforn, T. Malkmus, S. Müthing, M. Nolte, M. Piatkowski, O. Sander, The Distributed and Unified Numerics Environment, Version 2.4, *Arch. Numer. Softw.* 4 (100) (2016) 13–29. doi:10.11588/ans.2016.100.26526. URL <http://dx.doi.org/10.11588/ans.2016.100.26526>
- [8] P. Deuffhard, A. Schiela, M. Weiser, Mathematical cancer therapy planning in deep regional hyperthermia, *Acta Numer.* 2 (2012) 307–378.
- [9] L. Fischer, S. Götschel, M. Weiser, Lossy data compression reduces communication time in hybrid time-parallel integrators, *Comput. Vis. Sci.* 19 (1) (2018) 19–30.

- [10] S. Götschel, N. Chamakuri, K. Kunisch, M. Weiser, Lossy compression in optimal control of cardiac defibrillation, *J. Sci. Comp.* 60 (1) (2014) 35–59.
- [11] S. Götschel, C. von Tycowicz, K. Polthier, M. Weiser, Reducing memory requirements in scientific computing and optimal control, in: *Multiple Shooting and Time Domain Decomposition Methods*, Springer, 2015, pp. 263–287.
- [12] S. Götschel, M. Weiser, Lossy compression for PDE-constrained optimization: Adaptive error control, *Comput. Optim. Appl.* 62 (2015) 131–155. doi:10.1007/s10589-014-9712-6.
- [13] S. Götschel, C. von Tycowicz, K. Polthier, M. Weiser, Reducing memory requirements in scientific computing and optimal control, in: T. Carraro, M. Geiger, S. Körkel, R. Rannacher (Eds.), *Multiple Shooting and Time Domain Decomposition Methods*, Springer, 2015, pp. 263–287.
- [14] L. Grüne, M. Schaller, A. Schiela, Sensitivity analysis of optimal control for a class of parabolic PDEs motivated by model predictive control, *SIAM J. Control Optim.* 57 (4) (2019) 2753–2774. doi:10.1137/18M1223083.
- [15] L. Lubkoll, A. Schiela, M. Weiser, An optimal control problem in polyconvex hyperelasticity, *SIAM J. Control Optim.* 52 (3) (2014) 1403–1422.
- [16] L. Lubkoll, A. Schiela, M. Weiser, An affine covariant composite step method for optimization with PDEs as equality constraints, *Optim. Meth. Softw.* 32 (5) (2017) 1132–1161.
- [17] J. Müller, S. Götschel, C. Maierhofer, M. Weiser, Determining the material parameters for the reconstruction of defects in carbon fiber reinforced polymers from data measured by flash thermography, in: *AIP Conference Proceedings*, 2017, p. 100006.
- [18] G. Müller, A. Schiela, On the control of time discretized dynamic contact problems, *Comput. Optim. Appl.* 68 (2) (2017) 243–287. doi:10.1007/s10589-017-9918-5.
- [19] S. Schenkl, H. Muggenthaler, M. Hubig, B. Erdmann, M. Weiser, S. Zachow, A. Heinrich, F. Güttler, U. Teichgräber, G. Mall, Automatic CT-based finite element model generation for temperature-based death time estimation: feasibility study and sensitivity analysis, *Int. J. Legal Med.* 131 (3) (2017) 699–712.
- [20] O. Schenk, A. Wächter, M. Weiser, Inertia revealing preconditioning for large-scale nonconvex constrained optimization, *SIAM J. Sci. Comput.* 31 (2) (2008) 939–960.
- [21] A. Schiela, M. Weiser, Barrier methods for a control problem from hyperthermia treatment planning, in: M. Diehl, F. Glineur, E. Jarlebring, W. Michiels (Eds.), *Recent Advances in Optimization and its Applications in Engineering*, Springer, 2010, pp. 419–428.
- [22] J. Schneck, M. Weiser, F. Wende, Impact of mixed precision and storage layout on additive Schwarz smoothers, Report 18-62, Zuse Institute Berlin (2018).

- [23] M. Weiser, Pointwise nonlinear scaling for reaction-diffusion equations, *Appl. Num. Math.* 59 (8) (2009) 1858–1869.
- [24] M. Weiser, Optimization and identification in regional hyperthermia, *Int. J. Appl. Electromagn. and Mech.* 30 (2009) 265–275.
- [25] M. Weiser, On goal-oriented adaptivity for elliptic optimal control problems, *Optim. Meth. Softw.* 28 (13) (2013) 969–992.
- [26] M. Weiser, B. Erdmann, S. Schenkl, H. Muggenthaler, M. Hubig, G. Mall, S. Zachow, Uncertainty in temperature-based determination of time of death, *Heat Mass Transf.* 54 (9) (2018) 2815–2826.
- [27] M. Weiser, Y. Freytag, B. Erdmann, M. Hubig, G. Mall, Optimal design of experiments for estimating the time of death in forensic medicine, *Inverse Probl.* 34 (12) (2018) 125005. doi:10.1088/1361-6420/aae7a5.
- [28] M. Weiser, T. Gänzler, A. Schiela, Control reduced primal interior point methods, *Comput. Optim. Appl.* 41 (1) (2008) 127–145.
- [29] M. Weiser, S. Götschel, State trajectory compression for optimal control with parabolic PDEs, *SIAM J. Sci. Comp.* 34 (1) (2012) A161–A184.
- [30] M. Weiser, S. Scacchi, Spectral deferred correction methods for adaptive electro-mechanical coupling in cardiac simulation, in: *Progress in Industrial Mathematics at ECMI 2014*, Springer, 2017, pp. 321–328.
- [31] J. Nitsche, Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind, *Abh. Math. Univ. Hamburg* 36 (1971) 9–15.
- [32] M. Blyth, C. Pozrikidis, A Lobatto interpolation grid over the triangle, *IMA J. Appl. Math.* (2005) 1–17.
- [33] G. Zumbusch, Symmetric hierarchical polynomials and the adaptive h-p-version, in: A. Il'in, L. Scott (Eds.), *Proc. of the Third Int. Conf. on Spectral and High Order Methods*, Houston Journal of Mathematics, 1996, pp. 529–540.
- [34] J.-C. Nédélec, Mixed finite elements in \mathbb{R}^3 , *Numer. Math.* 35 (3) (1980) 315–341.
- [35] L. Morley, The triangular equilibrium element in the solution of plate bending problems, *Aero. Quart.* 19 (2) (1968) 149–169.
- [36] B. Schöling, *The Boost C++ Libraries*, 2nd Edition, XML Press, 2014.
- [37] R. E. Bank, A. Weiser, Some a posteriori error estimators for elliptic partial differential equations, *Math. Comput.* 44 (170) (1985) 283–301.
URL <http://www.jstor.org/stable/2007953>
- [38] J. Bramble, J. Pasciak, J. Xu, Parallel multilevel preconditioners, *Math. Comp.* 55 (1990) 1–22.

- [39] J. Shewchuk, Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator, in: M. Lin, D. Manocha (Eds.), *Applied Computational Geometry: Towards Geometric Engineering*, Vol. 1148 of *Lecture Notes in Computer Science*, Springer, 1996, pp. 203–222.
- [40] L. Avila et al., *The VTK User’s Guide*, 11th Edition, Kitware, 2010.
- [41] D. Stalling, M. Westerhoff, H.-C. Hege, Amira: A highly interactive system for visual data analysis, in: C. Hansen, C. Johnson (Eds.), *The Visualization Handbook*, Elsevier, 2005, pp. 749–767.
- [42] C. Geuzaine, J.-F. Remacle, Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities, *Int. J. Numer. Meth. Eng.* 79 (11) (2009) 1309–1331. doi:10.1002/nme.2579.
- [43] U. Ayachit, *The ParaView Guide: A Parallel Visualization Application*, 2015.
- [44] P. Deuffhard, U. Nowak, Extrapolation integrators for quasilinear implicit ODEs, in: P. Deuffhard, B. Engquist (Eds.), *Large Scale Scientific Computing*, Birkhäuser, Boston, 1987, pp. 37–50.
- [45] A. Dutt, L. Greengard, V. Rokhlin, Spectral deferred correction methods for ordinary differential equations, *BIT Numer. Math.* 40 (2) (2000) 241–266.
- [46] M. Weiser, Faster SDC convergence on non-equidistant grids by DIRK sweeps, *BIT Numer. Math.* 55 (4) (2015) 1219–1241.
- [47] P. Colli Franzone, L. Pavarino, S. Scacchi, *Mathematical Cardiac Electrophysiology*, Springer, 2014.
- [48] M. Emmett, M. Minion, Toward an efficient parallel in time method for partial differential equations, *Comm. Appl. Math. Comp. Sci.* 7 (1) (2012) 105–132.
- [49] A. Schiela, A. Günther, An interior point algorithm with inexact step computation in function space for state constrained optimal control, *Numer. Math.* 119 (2) (2011) 373–407.
- [50] M. Puso, T. Laursen, A 3D contact smoothing method using Gregory patches, *Int. J. Numer. Meth. Eng.* 54 (2002) 1161–1194. doi:10.1002/nme.466.
- [51] B. Hamann, G. Farin, G. Nielson, A parametric triangular patch based on generalized conics, in: G. Farin (Ed.), *NURBS for Curve and Surface Design*, SIAM, 1991, pp. 75–85.
- [52] H. Yserentant, On the Multi-Level Splitting of Finite Element spaces, *Numer. Math.* 49 (1986) 379–412.
- [53] J. M. Rogers, A. D. McCulloch, A collocation-Galerkin finite element model of cardiac action potential propagation, *IEEE Trans. Biomed. Eng.* 41 (1994) 743–757.

- [54] D. A. Hooks, M. L. Trew, Construction and validation of a plunge electrode array for three-dimensional determination of conductivity in the heart., *IEEE Trans. Biomed. Eng.* 55 (2 Pt 1) (2008) 626–635. doi:10.1109/TBME.2007.903705.