Archive of Numerical Software
vol. 5, no. 1, 2017, pages 63–94
DOI: 10.11588/ans.2017.1.29042

Asynchronous evaluation within parallel environments of coupled finite and boundary element schemes for the simulation of multiphysics problems.

Andreas Dedner[1] and Alastair J. Radcliffe[1]

[1]University of Warwick, Coventry, CV4 7AL, U.K.

Received: March 17th, 2016; **final revision:** October 31st, 2016; **published:** March 6th, 2017.

Abstract: A complete finite element (FEM) and boundary element (BEM) computational toolbox is presented, based on the Dune and Bem++ software packages respectively, for the efficient independent solution on parallel/multi-threaded multi-core computers of separate finite and boundary element systems. Each system has very different memory resource requirements, but can be coupled together within a common computer program for solving multi-physics PDE problems for computational sciences and engineering applications.

Examples of both direct (to a fixed-point) and indirect FEM-BEM iterative coupling, and their performance results with increasing core/thread count, drawn from electromagnetic scattering and fluid mechanics are presented as illustration of the wide scope of applications for this package.

1 Introduction

Finite element methods (FEM) and boundary element methods (BEM) have been around for many years, and due to the fundamental difference in their structures have typically been used for very different types of problem.

For unbounded computational problems, the classic example being an electromagnetic/acoustic wave scattering off an impenetrable obstacle, where a solution is required within a region of space (or even time) of unlimited extent, a finite element discretization of this whole region is impractical.

By requiring just the disretization of the boundaries of computational domains, however, (and hence the name) the boundary element method would avoid such a problem because the boundary of such an infinite region is just the finite surface of the scattering obstacle itself – the "other" boundary to the infinite region can be neglected provided suitable assumptions can be made about the solution at infinity Wrobel [2002].

This capability for supporting a solution in an unmeshed region, however, gives the BEM one of its principle drawbacks, namely that the solutions being "interpolated" in such regions are

assumed to be linear, and this limits its application to a relatively small class of problems where linear (or linearized) solutions are acceptable.

Finite elements, on the other hand, make no such assumptions on a solution, and thus give great flexibility for the simulation of all kinds of non-linear behaviour, but all, of course within a computational region of limited extent.

It should also be noted, that due to the implicitly localized nature of finite element interactions (giving sparse system matrices), parallelization for multi-core processing is relatively easy, with each core just computing the solution for a different physical region of the computational domain.

Because of the "dense" connections between the local values of the unknown solution at different locations in BEM discretizations, however, parallelization of BEM is much more challenging, and usually depends on an overall reduction in the amount of communication between these local values during the solution process through the use of "fast" techniques which can require a huge coding effort to implement.

Now, it is the complimentory, but very different, nature of the finite and boundary element methods which has led to the interest in their coupling together to tackle "mixed" problems Johnson and Nedelec [1980], Stephan [2004], Costabel [1987], Mund and Stephan [1997], with specific applications in electromagnetics Meddahi and Selgas [2003], including the wave scattering suggested above Hiptmair [2003], and fluids/elasticity Brink and Stephan [2001], Gatica and Heuer [2000], Meddahi and Sayas [2000], further references may be found in Radcliffe [2011, 2012].

These coupling schemes have typically involved the combining together of the discretizations afforded by the two methods to form one big system matrix to be inverted, however, the disparate but complementary nature of BEM and FEM means that the structure of this single system matrix can be very non-uniform, making its numerical inversion problematic.

Because the FEM only considers spacially local interations between the discretized values of the solution variables, the system matrices it creates are sparse, and the memory required for the solution of such problems may be distributed analogously to the spacial distribution of the variables themselves in what, following the definitions of Heroux et al. [2011], may be termed a "Mode 1" for the programming of modern multi-node computers with more than one core per node.

The BEM, however, at least in its traditional form, involves dense system matrices arising from the need for all values of the discretized unknown to know about all other values in the boundary integrals upon which the method is based.

It is for this reason that there are sparse and dense regions within the system matrix when FEM and BEM discretizations are combined at the (more fundamental) system matrix level.

The alternative to such a coupling of the system matrices, is to somehow couple the solutions their separate inversions would generate instead. This allows different optimized preconditioners and solvers to be used for the FEM and BEM parts separately.

These coupling methods may be referred to as "iterative", and as will be seen in the next section, involve repeated solution of the separate FEM and BEM problems within each iteration to provide a successive updating, from some initial guess, of the solution and/or its derivative, over the common boundary between the FEM and BEM regions until the updates change nothing and the solution is converged Lin et al. [1996].

Though not used here, these updates can be further defined through the use of relaxation parameters which specify, within a particular iteration, how the solution (or derivative) from one scheme should be used to set the Dirichlet and/or Neumann boundary conditions for the other at the next iteration. By allowing a "mix" of Dirichlet/Neumann data from the two coupled problems to be used as the boundary conditions for either at the next iteration, the convergence of the coupling scheme may be altered significantly.

For the examples presented here, though, such mixes are not used, and the data for setting the boundary condition of one problem will come exclusively from the solution of the other problem it is to be coupled to.

Now, while the sparse FEM systems work best on distributed memories, the dense BEM systems work best in shared memory environments, where multiple processes can access all the variable data that they will need over the entire discretized domain to perform their integrals. This involves a "Mode 2" style of programming Heroux et al. [2011], where the parallelization is achieved through the use of multi-threading on a single node with multiple cores.

Extreme-scale computing hardware is typically either specialized towards shared memory designs, with up to 64 cores (beyond which memory clashes become prohibitative), or distributed memory 1000+ cores, with the popular software libraries OpenMP and MPI being often used for the management of each type of memory respectively.

Thus one can say that any dense BEM implementation is well suited for a multithreading environment, for execution on a shared memory machine (as is the case for `Bem++`), while FEM applications are best written using MPI for distributed memory machines (as is the case with the Dune software packages). Thus a program that were to combine both FEM and BEM solves, would ideally have a hybrid or "bi-modal" memory architecture.

The memory architecture requirements of modern fast, data sparse BEM methods like the H–matrices available in `Bem++` (or other related "fast multipole" type methods) can, however, be significantly different, but will be considered for present purposes as closer to those of their dense forebears, than those of FEM.

As with numerical schemes themselves, the majority of literature on computational data structures has been connected with either the shared or the distributed memory architectures appropriate to the numerical scheme itself, rather than on hybrids. A good discussion on the data-structures and techniques needed to introduce ahared threading libraries into MPI models may be found in Heroux et al. [2011], which also has a few useful references therein.

Recently, approaches for combining the distributed and shared memory paradigms have been investigated – mostly aiming at extending an existing distributed memory model to benefit from the memory hierarchy and hardware structure of modern computer architectures. For example the distribution of the spatial grid for a finite element computation is carried out in two steps, first a coarser distribution over the nodes of the machine using MPI for data communication and in a second step subdividing these coarse patches into smaller chunks distributed over the cores of each node using a shared memory model, see Gaston et al. [2015], Ibanez et al. [2016], Olson et al. [2007] and references therein.

This parallelization strategy is also available in some Dune modules, for example for matrix free methods in Klöfkorn [2012]. This approach can be thought of as a *hierarchic hybrid* parallelization (or "hierarchic bimodal" to follow the "mode" terminology of Heroux et al. [2011]). It is restricted to single packages and does not require any global access to the whole set of degrees of freedom. This is an important difference to the challenges discussed in the following, where the `Bem++` library requires the whole grid to be available in a shared memory address space, while the Dune library and the linear solver packages used for solving the FEM problem rely on MPI for data exchange and are not even necessarily thread safe. We are not aware of any openly available publications in the area of computational PDEs employing a *non-hierarchic* hybrid parallelization approach to the use of distributed/shared memory concepts within the same program, and would hope that the present work might be among the first to openly address this issue.

This paper will present the internal structure and design decisions taken for a software toolbox that intertwines the asynchronous evaluation (across a parallel/multithreaded environment) of fem and bem constituent numerical kernels/solvers with disparate but complementary computing resource requirements for the development of efficient integration algorithms for the simulation of coupled partial differential equations arising in multiphysics applications ranging from electromagnetic wave scattering to electrically charged droplet deformations where the underlying

processes have disparate computing resource requirements that can be exploited effectively in this way.

In particular, we show the construction and management of special shared memory spaces, on a single node, that allow the running of a multi-threaded bem solver within an MPI application using all the cores of the node in which the shared memory sits. The management essentially involves granting ownership of different regions of the shared memory to different processes, or "ranks", to avoid access clashes if two different ranks try to write to the same memory address at the same time. Multiple reading from the same address does not pose the same problems.

Finally, this paper is intended as an introduction and guide to the associated software for use in new hybrid FEM/BEM application developments. While limited investigation and discussion of computational efficiencies is provided, this is from a very experimental "try it and see" approach comprehensible to the largest possible audience, and no formal symbolic analysis of the coupling methods used is presented. However, the reader is directed to Sayas [2009, 2013] and the references therein for a formal analysis of coupling methods similar (or identical) to those used here.

2 Coupling Methods

Consider an equation for an unknown u, valid inside an interior domain, $\Omega \subset \mathbb{R}^3$, satisfying

$$A\,u = f - Bv \tag{1}$$

arising from the discretization of a partial differential equation by the finite element method. Similarly, let the function v be a discrete solution defined on the boundary, $\Gamma \equiv \partial\Omega \subset \mathbb{R}^3$, satisfying

$$D\,v = g - Cu \tag{2}$$

arising again from a discretized partial differential equation using either finite or boundary elements (or both if this surface solution may itself be decomposed into two distinct surface solutions). Any BEM solution would of course be equivalent to solving the corresponding volume problem in the exterior $\Omega^\infty \equiv \mathbb{R}^3 \backslash \Omega$.

The full coupled problem can be written as a (block) matrix vector problem, where A, D arise from a discretized weak formulation of partial differential equations defining a function in the domain Ω and on Γ, respectively. The terms B, C provide coupling between the problem in the domain and on the boundary. The term B provides, for example, Neumann information using the surface solution as flux and C might take Dirichlet or Neumann traces from the domain solution to force the surface problem. Note that the terms A, B, D, C could refer to non-linear operators although we will mostly be focusing on linear operators in the following.

The combined bulk-surface problem thus results in the matrix system

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \tag{3}$$

For the coupling problems discussed here, it is possible to combine the system matrices from both to form a large matrix as shown above. In the case of FEM and BEM coupling this larger matrix will have both sparse and dense regions corresponding to the two different methods used in its construction. Consequently it is crucial for an efficient solution of the coupled system to not treat it as a single matrix, but to make use of its block structure. Furthermore, the methods discussed here allow us to treat the solvers for the different parts of the system as "black boxes", allowing us to use optimal solvers for both the surface and the bulk part of the problem. Thus we can use the efficient methods available in Dune for solving FEM problems (including multigrid and direct solvers) and the optimal methods for the BEM problems available in `Bem++` (including

multipole and H-matrix methods). An efficient solution method is based, for example, on a block-preconditioned GMRes iterative solver Feischl et al. [2015]. We discuss a similar approach below. First we describe the idea based on a simple iterative updating of each scheme's solution allowing convergence to a fixed point solution.

Assume in the following that the matrices on the diagonal result from a bounded, positive bilinear form so that A and D are invertible, and system (3) may be preconditioned by the block diagonal matrix

$$P = \begin{pmatrix} A^{-1} & 0 \\ 0 & D^{-1} \end{pmatrix} \tag{4}$$

to give the equivalent system

$$\begin{pmatrix} A^{-1} & 0 \\ 0 & D^{-1} \end{pmatrix} \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} A^{-1} & 0 \\ 0 & D^{-1} \end{pmatrix} \begin{pmatrix} f \\ g \end{pmatrix} \tag{5}$$

which may be simplified to

$$\begin{pmatrix} I & A^{-1}B \\ D^{-1}C & I \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} A^{-1}f \\ D^{-1}g \end{pmatrix} \tag{6}$$

Now, one of the simplest iterative schemes one can devise for solving $A\,\mathbf{x} = \mathbf{b}$ is that of Richardson with a relaxation parameter of unity

$$\mathbf{x}^{k+1} = \mathbf{x}^k - A\,\mathbf{x}^k + \mathbf{b} \tag{7}$$

Applying this to (6) with $\mathbf{x} = (u, v)^T$ gives

$$\begin{aligned} \begin{pmatrix} u^{k+1} \\ v^{k+1} \end{pmatrix} &= \begin{pmatrix} u^k \\ v^k \end{pmatrix} - \begin{pmatrix} I & A^{-1}B \\ D^{-1}C & I \end{pmatrix} \begin{pmatrix} u^k \\ v^k \end{pmatrix} + \begin{pmatrix} A^{-1}f \\ D^{-1}g \end{pmatrix} && (8) \\ &= \begin{pmatrix} u^k - u^k - A^{-1}Bv^k \\ v^k - D^{-1}Cu^k - v^k \end{pmatrix} + \begin{pmatrix} A^{-1}f \\ D^{-1}g \end{pmatrix} && (9) \\ &= \begin{pmatrix} A^{-1}\left(f - Bv^k\right) \\ D^{-1}\left(g - Cu^k\right) \end{pmatrix} && (10) \end{aligned}$$

and our first, block Jacobi style iterative coupling solver, to be initiated from an initial guess u^0, v^0 and computed for integer $k > 0$:

$$v^{k+1} = D^{-1}(g - Cu^k)\,, \qquad u^{k+1} = A^{-1}(f - Bv^k) \tag{11}$$

A very simple variant of this gives a second, block Gauss-Seidel style method to solve the coupled problem:

$$v^{k+1} = D^{-1}(g - Cu^k)\,, \qquad u^{k+1} = A^{-1}(f - Bv^{k+1}) \tag{12}$$

It should be noted that very simple Richardson iterative solvers like these were considered in the early days of domain decompositon methods and may have considerable limitations regarding their efficiencies and even convergence, which could be critical or slow for some scenarios Toselli and Widlund [2005].

The preconditioner P in equation (4) used in the two simple iterative schemes can be modified by using suitable preconditioners instead of the exact inverse matrices. So for example A^{-1} can be replaced by an algebraic multigrid preconditioner as available in Dune-Istl. If a Krylov type method is used to invert A then it is possible to fine-tune stopping criteria in each step of the Richardson iteration to improve efficiency of each step, but at the cost of perhaps a slight increase in the number of iterations. This type of efficiency tuning has not been investigated here but could be added to the package.

The final approach discussed here is more involved, but more robust, and often leads to a reduction in the number of iterations required. It makes use of a GMRes method and a reformulation of problem (3) eliminating either v or u: First the second row of (3) gives

$$v = D^{-1}g - D^{-1}Cu \tag{13}$$

which may be substituted into the first row of (3) to give

$$Au + BD^{-1}g - BD^{-1}Cu = f \tag{14}$$

Multiplying through by A^{-1} we have

$$u - A^{-1}BD^{-1}Cu = A^{-1}f - A^{-1}BD^{-1}g \tag{15}$$

Thus solving the coupled system

$$M\,u = b \tag{16}$$

where

$$M = I - A^{-1}BD^{-1}C \tag{17}$$
$$b = A^{-1}\left(f - BD^{-1}g\right) \tag{18}$$

is then completely equivalent to solving system (3). The problem can now be solved using a Krylov iterative solver, for example GMRes. These methods only require computing the matrix vector product $M\,u$.

Of course, it would also have been possible to perform the eliminations the other way around to obtain a combined system $M\,v = b$; solving for the surface variable v instead, with possibly different memory requirements. A third option would be to apply the Krylov method directly to the full block system 6. The framework presented here is flexible enough to allow for an easy implementation of these alternative coupling methods.

Note that the first two coupling methods are included primarily as a simple demonstration on how to implement coupling schemes within the current framework. These methods are well known not to be very robust and efficient but are included to show the appropriate code structure and, due to their simplicity, are intended as a starting point for user developments of coupling schemes/solvers more suited to their particular problems.

Indeed, the use of an appropriate coupling scheme can be crucial to the success of the FEM–BEM coupling as a whole, thus, for the scattering wave case seen in section 6.2, for example, it is suggested the reader try the Jacobi and Gauss-Seidel methods (which do not converge at all without additional relaxation), before restoring the original GMRes coupling scheme more suited to the problem. In the simpler FEM-FEM coupling example found in section 6.1, on the other hand, these two methods *will* converge within a reasonable number of iterations.

3 Software Dependencies

The software, version 2.3, to be presented here for the use, and possibly coupling, of FEM and BEM within the same parallel program is based on the Dune 2.3 and `Bem++` software packages, which we now examine in a little more detail.

Dune stands for the "Distributed and Unified Numerics Environment", and is a modular toolbox for solving partial differential equations (PDEs) with grid-based methods Dune [2012]. It supports the easy implementation of methods like finite elements (FE), and finite volumes (FV).

DUNE is free software licensed under the GPL (version 2) with a runtime exception, thus it is possible to use Dune even in proprietary software.

The underlying idea of Dune is to create slim interfaces allowing an efficient use of legacy and/or new libraries. Modern C++ programming techniques enable very different implementations of the same concept (e.g: grids, solvers, ...) using a common interface at a very low overhead. Thus Dune ensures efficiency in scientific computations and supports high-performance computing applications Dune [2012].

Dune applications are designed and written to use pre-existing packages, or "modules", which can be bolted together to provide whatever functionality the developer will need in their own program (also written as a module). This allows easy sharing of computer code with a "black-box" mentality that anyone using a module should not need to know how it actually works.

Essential to most Dune programs are the five core modules, Dune-Common, Dune-Geometry, Dune-Grid and Dune-Istl, which provide the basic housekeeping required of any program handling meshes.

Based on these core modules, the Dune-Fem module provides all the basis function definitions, matrix assembly and problem definition routines required for the creation of a complete finite element program. If an appropriate discretization module, or "grid-manager", is also used (see next) then parallel assembly and solution of the resulting problems is easily possible for the novice user, with minimal knowledge. The parallelization is based on calls to routines drawn from the Message Passing Interface (MPI) libraries, with a good computational scaling dune-fem howto [2015].

To enable the efficient parallelization of the FEM part of the coupled FEM-BEM solver, the grid manager chosen is Dune-ALUGrid Alkaemper et al. [2016], which allows the construction and partitioning across multiple processors of any three-dimensional volume and surface meshes the user might specify for the separate FEM and BEM sub-problems. Importantly, it also allows a user defined specification of the partitioning, so that optimal division of the given meshes is possible, with the same partitioning being used for both volume and surface meshes where required to facilitate data exchange.

The last of the Dune modules required may be thought of as almost literally sticking the whole thing together. With FEM and BEM solutions required on both volume and surface meshes in the various examples that will be presented, it is important to have a means of communicating solution data from one computational domain to another in order that it may be used to update the respective solutions there. The Dune-Grid-Glue module Bastian et al. [2010], provides just the simple yet effective means required for transferring various numerical values between the grids (even partitioned and non-matching grids).

Outside of the Dune environment, and to provide the necessary boundary element routines the `Bem++` library is used; an open-source Galerkin boundary element library that handles Laplace, Helmholtz and Maxwell problems on bounded and unbounded domains in three space dimensions Smigaj et al. [2015]. Owing to build compatibility issues, at present the installation script, see appendix §B, downloads a certain git hash of `Bem++` which resolves to the 3.0.3 tag. When the `Bem++` compatible with Dune 2.4 becomes available, this restriction is intended to be lifted, with a 2.4 release of the Dune-Fem-Bem-Toolbox.

The package BEM++ is itself built extensively using routines drawn from the `Eigen` C++ template library for linear algebra with matrices, vectors, numerical solvers and related algorithms Guennebaud et al. [2010]. Eigen is a template library defined in the headers and doesn't have any dependencies other than the C++ standard library. Parallelization in `Bem++` is achieved using multi-threading based on Intel's Threading Building Block TBB [2014]. For the grid data structure `Bem++` relies on the DUNE core modules, so data exchange between the two packages is achievable without any reorganization of data and thus is highly efficient.

4 Implementing the Coupled Problem

The implementation discussed in this paper is based on the approach adopted in the "howto" module of DUNE-FEM dune-fem howto [2015]: both the bulk and the surface problems use a `Model` class describing the functions required for the underlying partial differential equations. This is passed to a `Scheme` class which sets up the required types for the discrete spaces, creates the glue objects that will be needed for any coupling and selects the solvers to be used in addition to also holding the actual solution.

The creation of the glue objects is done automatically by the scheme constructors, with every second scheme defined (within the `main.cc` program file) creating a glue object between its mesh and that of the previously declared scheme. Thus, at least for the present software release, minor attention needs to be paid to the order in which the schemes are declared.

In its original form within DUNE-FEM-HOWTO, the `Scheme` class implements two methods `prepare` and `solve`, which setup the right hand side vector and assembles and solves the system, respectively.

For the DUNE-FEM-BEM-TOOLBOX module the original scheme class used in the DUNE-FEM-HOWTO examples is extended by the additional method `couple` which processes the data communication between two schemes – one "bulk" scheme for the inside region, and one "surface" scheme for the surface of that region.

Thus, in our module we provide two such scheme classes, one for FEM discretized elliptic problems defined either on a surface mesh or on a volume mesh, and a second for exterior BEM problems defined on a surface grid only. In the case of the BEM problem the `Model` only provides the static information about which integral operators to use for the BEM problem. Additional parameters, for example wave numbers, can be set through run time parameters as described below.

For FEM problems systems of non-linear equations of the form

$$\int_\omega D(x, u(x, \nabla_\omega u(x)) \cdot \nabla_\omega \varphi(x) + m(x, u(x), \nabla_\omega u(x))\varphi(x) = \int_\omega f(x)\varphi(x)$$

can be defined through the `Model` class. Here $\omega = \Omega$ or $\omega = \Gamma$ is the volume or the surface with $\nabla_\Omega = \nabla$ and ∇_Γ denoting the surface gradient. Although non-linear elliptic problems can be described by the model class and handled by the schemes we will mainly focus on linear problems in the following.

In this version of the module only a linear coupling between the two schemes is considered. This is to be extended in future releases. The surface solution is assumed to be coupled to the volume problem using Dirichlet or Neumann conditions taking the form

$$\nabla u \cdot n + \alpha_\Omega u = \alpha_\Gamma v \,.$$

Here α_Ω and α_Γ are functions defined on the coupling surface.

Note that for a surface FEM problem v are the values of the quantities defined on the surface while for a BEM problem v will actually be an approximation of the normal derivative of the far

field solution. Since the $\alpha_\Omega u$ contribution of the coupling will be part of the matrix A, the actual coupling matrix is

$$< Bv, \varphi >:= \int_\Gamma \alpha_\Gamma v\varphi \tag{19}$$

The main part of the bulk problem is of the form

$$< Au, \varphi >:= \int_\Omega D(x,u,\nabla u)\cdot\nabla\varphi + m(x,u,\nabla u)\varphi + \int_\Gamma \alpha_\Omega u\varphi \tag{20}$$

and described by a single `Model` class consisting of four methods:

C++ code

```
template< class Entity, class Point >
void source ( const Entity &entity, const Point &hatx,
              const RangeType &value, const JacobianRangeType &gradient,
              RangeType &flux ) const
template< class Entity, class Point >
void linSource ( const RangeType& uBar,
                 const Entity &entity, const Point &hatx,
                 const RangeType &value, const JacobianRangeType &gradient,
                 RangeType &flux ) const
template< class Entity, class Point >
void diffusiveFlux ( const Entity &entity, const Point &hatx,
                     const RangeType &value, const JacobianRangeType &gradient,
                     JacobianRangeType &flux ) const
template< class Entity, class Point >
void linDiffusiveFlux ( const RangeType& uBar, const JacobianRangeType& gradientBar,
                        const Entity &entity, const Point &hatx,
                        const RangeType &value, const JacobianRangeType &gradient,
                        JacobianRangeType &flux ) const
```

The methods provide the implementation of $m(x,u,\nabla u)$, $D(x,u,\nabla u)$ and their linearization around $\bar{u}$ on an entity and local coordinate $\hat{x}$. Addition methods on the `Model` describe boundary conditions and a forcing term.

These `Model` classes are used for the implementation of both bulk ($\omega = \Omega$) and for surface ($\omega = \Gamma$) finite element methods. The lower diagonal block D representing the main part of the boundary pde is of the same form as A given above and the coupling occurs through the forcing term:

$$\int_\Gamma D(x, v(x,\nabla_\Gamma v(x))\cdot\nabla_\Gamma\varphi(x) + m(x,v(x),\nabla_\Gamma v(x))\varphi(x) = \int_\Gamma \Big(\beta_0 u + \beta_1 \nabla u\cdot n\Big)$$

With suitable function β_0, β_1 defined on the coupling surface. Therefore,

$$< Cu, \varphi >:= \int_\Gamma \Big(\beta_0 u + \beta_1\nabla u\cdot n\Big)\varphi \tag{21}$$

However, user defined parameters $\alpha_\Omega, \alpha_\Gamma, \beta_0, \beta_1$ are intended for future releases, and all the examples presented below will effectively have them set to one where they are present.

For describing problems to be discetized by the boundary element method a similar model class needs to be implemented. We provide a model for a far field Laplace and Helmholtz problem. For example, to solve a complex valued Helmholtz equation of the form

$$\begin{aligned} \triangle u + \omega u &= u_{\text{inc}} && \text{in } R^3 \setminus \Omega \\ u &= g && \text{on } \partial\Omega \end{aligned}$$

the model class has the following form:

C++ code

```
// FS: The complex function space, GP: the grid part used
template <class FS, class GP>
struct HelmholtzModel
{
  HelmholtzModel(double omega) ;
  struct IncidentData {
    void evaluate( const DomainType& x, RangeType& res ) const;
  };
  struct DirichletData {
    void evaluate( const DomainType& x, RangeType& res ) const;
  };
};
```

The two sub-structures are used to implement the Dirichlet data g and the incident wave u_{inc} in global coordinates.

A model class together with a grid is used to initilize a `Scheme` class which sets up the required types for the discrete spaces and solvers and in addition also holds the solution. The `Scheme` classes provide two methods `prepare` and `solve`, which sets up the right hand side vector and assembles and solves the system, respectively. In this module two such scheme classes are implemented: `FemScheme` for solving non-linear fem problems using the classes provided in the Dune-Fem module, and the `BemScheme` providing the bindings to `Bem++`.

C++ code

```
template < class Model, SolverType solver=istl >
struct FemScheme {
  FemScheme(const typename Model::GridPartType &gridpart, const Model& model);
  void prepare();
  void solve();
};
```

In addition the scheme class provides methods for accessing the discrete function spaces and the discrete function storing the computed solution. The `SolverType` template argument can be used to specify the solver backend to be used. While in Dune-Fem the bindings for a number of solver packages are available (Dune-Istl, `PETSc`, `UMFPACK`, `Eigen`) we use Dune-Istl for all our finite element computations. The `BemScheme` class has a very similar structure but without the option of choosing a solver backend since we use the H-matrix implementations available in the `Bem++` package.

Both our coupled problems given by (1) and (2) consist each of three terms: $A\,u, f$ and $B\,v$ and $D\,v, g$, and $C\,u$, respectively. The first two terms of each problem are covered by the `prepare` and `solve` methods on the scheme classes. To describe the full problem the coupling terms $B\,v, C\,u$ need to be implemented. To this end, the scheme classes are extended by one additional method:

C++ code

```
template <class OtherDiscreteFunctionType>
void couple(const OtherDiscreteFunctionType &otherSolution);
```

where the template parameter is the solution of the other problem. As already mentioned above, coupling is achieved using the Dune-Grid-Glue module. In this version of the module only a linear coupling between the two schemes is considered. This is to be extended in future releases.

The actual coupling mechanism is itself implemented in a `Scheme` class, taking the bulk and the surface scheme classes and then itself providing a `prepare` and `solve` method.

As described above this module provides three such implementations:
`GaussSeidelCouplingScheme` implementing equation (12), `JacobiCouplingScheme` for (11) and finally `GMResCouplingScheme` for equation (16), which may all be found in the folder

dune-fem-bem-toolbox/dune/fem_bem_toolbox/coupling_schemes

Of course such coupling necessitates a number of different grid and function space types associated with the communication of data, and the setting-up of the glue objects, especially with the moving grids of the third case study described below. For convenience, all these types are grouped together with the original type declarations of the schemes themselves within a `traits.hh` file inside the source code folder `source` for each example application.

5 Parallel Coupling Approach

Underpinning all of the coupling approaches is the use of the **dune-grid-glue** module to create "glue" objects between the bulk and surface meshes allowing communication of solution values held on each.

For parallel runs, the MPI based solvers in Dune exploit a (possibly user defined) partitioning of full volume or surface grids into different pieces, each of which is attributed to a different process, such that the matrix assembly and solution is performed independently on each process.

Now Dune-Grid-Glue will create distinct "glue objects" between each of these individual pieces. With one piece of the volume grid, and one piece of the surface grid on a particular rank, one glue object will be created (on the same rank) that essentially creates and holds a surface grid corresponding to the intersection of the bulk and surface grids it is gluing together – and which thus normally coincides with the surface mesh itself.

The glue object then provides an "iterator" that iterates over the "glue elements" of this glue intersection grid which each have pointers to the bulk and surface elements of the bulk and surface grids it joins.

All that is thus required to exchange information from the bulk problem to the surface one (or vice-versa), is thus to iterate through these glue elements, for each one evaluating (at the Lagrange or quadrature points) the bulk solution that needs to be communicated within the glue element (coincident with a face of its own bulk element), and using the glue element to then assign this information to the surface solution at the corresponding Lagrange point within the corresponding surface element or assemble the desired integral terms. In this version of the module the coupling matrices B, C are not assembled but the coupling is computed on the fly. For the current problems where inverting A, D is the dominant cost the required iterations over the partitioned surface grid is an acceptable overhead.

The Bem++ library, in both serial *and* parallel execution modes, always works with full, unpartitioned (surface) grids. The parallelization being achieved through use of the multi-threaded assembly and solver routines both with or without the use of H-matrices to further speed-up assembly Smigaj et al. [2015]. Clearly the first challenge is thus to allow the communication of the solution variables held on the "full" surface grid (for the `Bem++` operations), to and from the grid parts held on each process performing the Dune calculations. As storage backend for the degrees of freedom ("dofs") the vector classes from the `Eigen` package already used extensively by `Bem++` are used.

These `Eigen` vector structures give the option of being setup using a chunk of specified raw memory so that special shared memory blocks accessed via a special "shared" dof vector class `EigenVector` may be used for reading and writing through the vector interface of `Eigen` and thus enabling a seamless exchange of data between the Dune-Fem and the `Bem++` solvers and between different processes, but to `Bem++` they look just like a regular vector of unknowns over the whole of the surface grid.

This wrapper class is called `Dune::Fem::EigenVector` and implements the vector interface used in Dune-Fem to store the degrees of freedom for discrete functions to be stored both over a full grid and a partitioned grid. This class combines the vector classes of `Eigen` together with a block of raw memory to store the vector elements.

The module Dune-Fem is flexible with respect to the storage structure of the degrees of freedom. So through simple adapter classes, Dune-Fem makes it possible to use a wide range of solver backends for storing dof vectors for discrete functions and it is straight-forward to use the vectors of `Eigen`. Thus since `Eigen` can make use of the shared memory blocks, accessing those through the Dune-Fem interface is seamlessly possible.

Of course, to `Bem++` these dof vectors look like a regular vector of unknowns over the whole of the surface grid it has been given previously, but because they are built on shared memory, each of the Dune MPI ranks may access parts of the vector to fill in/evaluate each on the partitioned mesh.

Furthermore, because the `Eigen` vector classes provide an option for the user to provide raw memory to use as storage space, by allowing the first rank of a parallel execution to arrive at the particular point to create a special piece of shared memory, whose address it then communicates to all the other ranks, all ranks may then pass this address to the constructor of a new eigenvector storage array which they each create, but the addresses of whose elements will then be the same for all ranks.

By letting all ranks read from, and "non-clashing" (see below) ranks write to these shared elements, communication of values is possible.

This is the principle means for collecting and distributing data required for the `Bem++` routines, which are all run from a single process (which creates multiple threads). Typically the Dirichlet data required as the right hand side of the bem problem would be accumulated into a shared vector by the various ranks, the Neumann data returned by the bem solve (initiated on the rank zero process) would similarly be put into a shared vector, from which each rank could then extract the values relevant to it.

The construction of the shared memory block is done in the class `SharedMemory`. Rank zero tries to construct a block with a randomly generated name using `boost::interprocess::mapped_region`. If this is successful, the name is communicated to all other processes and they all use the same block of memory to store the degrees of freedom. The class has the following constructor

C++ code

```
1  EigenVector ( unsigned int size = 0, const bool shared = false );
```

where the last argument can be used to determine if a block of shared memory is to be used. The following code snippet shows how to setup both a distributed and a shared discrete function for a given `DiscreteFunctionSpaceType`, e.g., a Lagrange space:

C++ code

```
1  typedef Dune::Fem::EigenVector<double> DofVectorType;
2  typedef Dune::Fem::VectorDiscreteFunction< DiscreteFunctionSpaceType, DofVectorType
       > > DiscreteFunctionType;
3  DiscreteFunctionSpaceType distributedSpace( distributedGridPart );
4  DiscreteFunctionSpaceType fullSpace( fullGridPart );
5  DiscreteFunctionType rankLocalDiscreteFunction("rank_local", distributedSpace );
6  DiscreteFunctionType sharedDiscreteFunction("shared", true, fullSpace );
```

Of course, appropriate synchronization of the filling/extracting data tasks is needed. While reading from a particular piece of memory does not usually pose any problems with multiple processes trying to access the same memory address, writing to such an address does. The synchronization of these tasks is performed by allowing each rank to run over its own specially created glue object between its particular gridpart, and the full grid (on which the bem operations are performed), putting-down or picking-up values at the positions given by the glue object in accordance with a special "dofrank" array which grants (or not) the permission to write to any particular memory address.

Each rank builds up its own "dofrank" array, defined over the full grid, where the integer values at each dof (corresponding to a particular grid point of the full grid) are just the rank number of the highest rank whose individual gridpart has a coincident grid point there.

Thus in summary to minimize communication and synchronization costs the degrees of freedom of g_Γ are stored in a block of shared memory using `Boost`'s interprocess package. Note that g_Γ is a continuous Lagrange function so that some of the degrees of freedom are shared between processes. To avoid race conditions, a degree of freedom in the shared memory block is only written to by one process (the one with the lowest rank). The solution of the `Bem++` solver is also directly written to a block of shared memory. After the Neumann data (piecewise constant) is computed, the second glue object is used again to move the data from the full surface grid onto the distributed surface grid. Note that each step described above is almost completely parallel. The assembly routines and the solvers from the `Bem++` library are only called from one process which then sets up its own multiple threads to carry out the computations in parallel.

A particular element (corresponding to a particular dof) in a shared memory eigenvector may then only be modified by a process whose rank is equal to that held in the dofrank array at the same point. It is thus just the highest ranking process at any particular gridpoint who is allowed to fill in dof values at the corresponding point.

For the finite-element problems we use a distributed grid parallelization strategy based on MPI. The required communication for both the assembly and the solvers are directly available through the Dune-Fem module and no changes are required to the `FemScheme` class to use it for parallel runs. Although the Dune-Grid-Glue will work for an arbitrary partitioning of both the surface and the bulk grid, in this version we make the following assumption: for each process p, the part Ω_p of the bulk grid on processor p and the part Γ_p of the surface grid satisfy $\Gamma_p = \partial\Omega_p$. As a consequence of this restriction on the partitioning, surface and bulk entities that are glued together always reside on the same process so that no additional communication is required.

This module provides two load balancing handlers for the `partition` method of Dune-ALUGrid to enforce this type of matching partitioning. The first `SimpleLoadBalanceHandle` class uses a simple coordinate bisection approach to partition both the bulk and the surface grid. While this approach leads to a reasonably good partition of the surface the bulk partitioning suffers from bad connectivity at the origin. To improve the bulk partitioning this module also provides `ZoltanLoadBalanceHandle`. The partitioning generated by this class is identical to the simple partitioning described previously but combines this with `Zoltan`'s graph partitioner Boman et al. [2012] around the origin to reduce the number of shared vertices there. In summary, coupled surface and bulk finite-element simulations can be in parallel with little change to the serial code.

As befits the nature of boundary element formulations the `Bem++` library, however, provides only shared (and not distributed) memory parallelization and to assemble the boundary integral operators requires an unpartitioned mesh for parallel assembly. For the matrix inversion, `Bem++` uses the OpenMP multi-threaded solvers of the `Eigen` package and with the additional option of using internal H-matrix based solvers. Again the dof vectors for the right hand side and the solution of the boundary element problem need to be accessible from all threads. Combining the distributed grid approach used in the Dune packages with the solvers in `Bem++` requiring the full unpartitioned grid is challenging. To achieve this an additional scheme (`SharedBemScheme`) is available in this module. On each process this scheme is constructed using both the full surface grid Γ and a distributed grid $\Gamma = \bigcup_p \Gamma_p$. Discrete functions g_Γ, λ_Γ defined on the full grid are setup to be used with the solvers in the `Bem++` library. To setup the right hand side for the boundary element problem on the full grid the term $\Pi_\Gamma u$ in the `couple` method needs to be computed. Note that u will be distributed and so a process p will only "see" the part of u computed on the partition Ω_p.

First we use the glue objects setup between the partitioned bulk grid and the partitioned surface grid (used also in the fem-fem coupling example below) to transfer the bulk data onto Γ_p.

A second glue object defined between the part of the surface Γ_p onto the full surface Γ is now used to transfer the Dirichlet data on Γ_p to the discrete function g_Γ defined on Γ.

This two-stage process allows any surface problems requiring only finite elements to have the performance benefits of distributed grid parallelization, while allowing any BEM (surface) problems to work on undistributed grids.

6 Case Studies

Presented here are three example applications to illustrate the use of the new fem-bem toolbox, the source code for which may be found in the folders within

dune-fem-bem-toolbox/case_studies

as will be indicated. All experiments were carried out on an Intel Xeon(R) E5-2650v2, 2.6GHz CPU node with 8 physical cores. For completeness, and to further later discussions into parallel performance of the coupling methods, which will make reference to "efficiencies" as measured by the percentage “efficiency”

$$E = 100 \times \frac{\text{time to solve on one core}}{\text{core count} \times \text{solve time}} \tag{22}$$

two additional folders are included within the case studies as example FEM only and BEM only applications.

The first solves a simple real valued Dirichlet bounded Poisson problem inside a unit cube, whilst the second solves for the Helmholtz equation, (which will be introduced in the second case study) outside a unit sphere.

These two reference problems are briefly discussed here, but only in terms of their parallel performance efficiencies shown in Figure 1, Figure 2 and Figure 5(left), respectively. The source code for them may be found within folders

dune-fem-bem-toolbox/case_studies/fem_test

and

dune-fem-bem-toolbox/case_studies/bem_test

respectively, with the same general structure, build and running instructions as the three studies which will be investigated in detail below.

The BEM only test results were performed on the surface mesh `sphere-h-0.05.msh` of 11459 elements found in the `bem_test` data folder, while the FEM only test results used the `unitcube-3d.dgf` in the `fem_test` data folder with chosen element diameters of 1/52 the (unit) edge length.

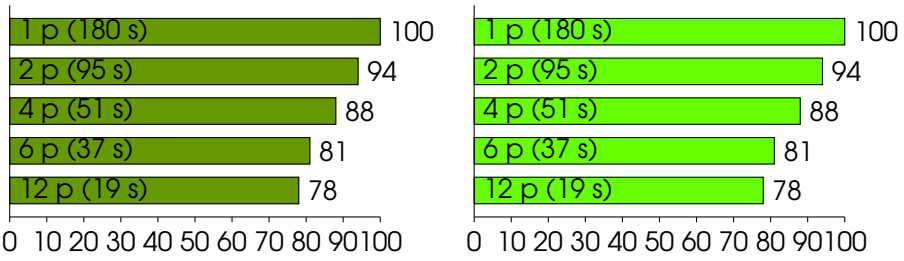

Figure 1: Percentage efficiencies, E, see equation (22) of operator assembly (left) and assembly+solve (right) times in seconds, “s”, for solution of a BEM only Helmholtz problem using just `Bem++` *without* H-matrices for different processor/thread counts, “p”.

First we note, however, that the assembly of the operators within Bem++ may be performed with or without the use of so-called "H-matrices". If they are not used, then an LU decomposition of the system matrix may be performed in the assembly step, leaving a simple back-substitution for the solve to do. This allows assembly and solve times as indicated in Figure 1 to be obtained with the test problem. Thus the solve step would appear to be instantaneous relative to the assemble step.

If, however, H-matrices are turned on, then a direct LU decomposition is not possible, and the solve step will involve a Krylov iterative solver and necessarily more work, and thus we obtain the solve times (and efficiencies) as appearing in Figure 2. Clearly the assembly times are much reduced with the H-matrices, but this is offset slightly by the more expensive solves.

This leads us to the conclusion that if repeated solution using the same basic BEM operators is going to be required, for instance in the repeated evaluations of a solution within a fixed-point iterative coupling scheme (as will be seen in the second case study), then it is perhaps best not to use the H-matrices in Bem++, because one LU decomposition, performed just once within the assembly stage, will allow very short times for many repeated solve steps within the iterative coupling scheme. In future, alternative more efficient parallelizations of the H-matrix assembly and solve steps Kriemann [2013], not yet implemented in Bem++, might change this advice however.

Now, if the BEM operators instead need to be reassembled often, as will be seen in the third case study, with no more solve steps than assembly steps, then H-matrices might be advantageous because of their speed up of the assembly process leading to generally significantly smaller overall combined assemble and solve times, compare Figure 1(right) with Figure 2(right).

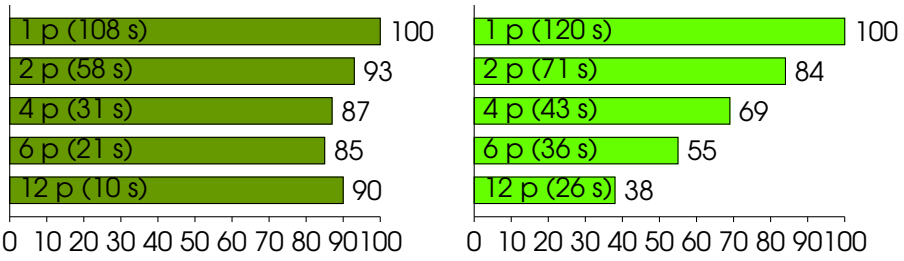

Figure 2: Percentage efficiencies, E, see equation (22) of operator assembly (left) and assembly+solve (right) times in seconds, "s", for solution of a BEM only Helmholtz problem using just Bem++ *with* H-matrices for different processor/thread counts, "p".

It should also be noted that the Bem++ H-matrix assembly of the BEM operators is very efficient with increasing parallelization, and near optimal, as can be seen in Figure 2(left). However, as soon as the solve stage to numerically invert these operators is taken into these timings as well, the parallel performance drops off significantly, see Figure 2(right).

The first case study will show a simple FEM-FEM iterative coupling on a toy problem, using the Gauss-Seidel approach, to illustrate performance with core count of a purely finite element problem for comparison.

The second and third examples will demonstrate real world problems with, respectively, the iterative GMRes scheme for the direct coupling of FEM and BEM in an electromagnetic scattering example, and an indirect FEM-BEM coupling for a fluid example, which may not be so easily cast into the form of equations (12), (11) or (16), but illustrates the future possibilities for many other types of FEM-BEM coupling.

For additional clarity, the key governing equations and boundary conditions in the following have been labelled with "FEM" or "BEM" to indicate the discretization scheme used for the former, and

"Dir", "Neu", "Rob" and "Rad" for the Dirichlet, Neumann, Robin and Radiation type boundary conditions of the latter.

Instructions on running the examples are included after each problem description, but of course, this should not be attempted until after a successfull installation of the software, for which the reader is referred to appendix B.

6.1 Bulk-Surface Fem-Fem Coupling Example

The source code for the first case study may be found within the following folder:

dune-fem-bem-toolbox/case_studies/coupled_sphere

Notes on how to run this example will be presented after the following problem description.

Here a volume (or "bulk") solution, $u \in \Omega$, is coupled to a surface solution on Ω's surface, $v \in \Gamma \equiv \partial\Omega$. Both solutions being represented by piecewise linear and globally continuous finite element basis functions.

The coupling, through the boundary conditions, and governing equations are Ranner and Elliott [2013]:

$$FEM: \quad -\Delta u + u = f \quad in\ \Omega \tag{23a}$$

$$Rob: \quad u - v + \frac{\partial u}{\partial n} = 0 \quad on\ \Gamma \tag{23b}$$

$$FEM: \quad -\Delta_\Gamma v + v + \frac{\partial u}{\partial n} = g \quad on\ \Gamma \tag{23c}$$

The forcings f and g must, of course, be compatible, and may be found from any choice of analytic solutions for u and v. In the following, we chose the exponential solutions:

$$\begin{aligned} u(x,y,z) &= exp\left[-x\,(x-1) - y\,(y-1)\right] \\ v(x,y,z) &= \left[1 + x\,(1-2\,x) + y\,(1-2\,y)\right] \\ &\quad \times exp\left[-x\,(x-1) - y\,(y-1)\right] \end{aligned} \tag{24}$$

which may easily be seen to satisfy the governing equations and boundary conditions.

The weak formulation of (23c) suggests an iterative coupling of the form

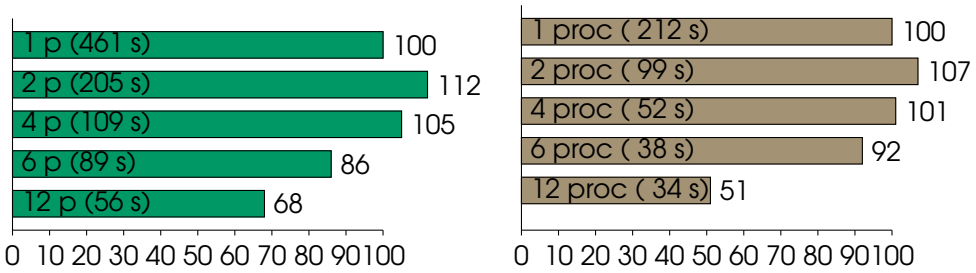

Figure 3: Percentage efficiencies, E, see equation (22) and solve times in seconds, "s", for solution of the coupled sphere example with element diameter ("cellsize") 0.019, giving 1924626 elements, see § 6.1, using just Dune modules (left) and for solution of the charged droplet example (right) on the "maxi" meshes of 18239 elements, see § 6.3 later, using both Dune and Bem++, for different processor/thread counts, "p".

$$\underbrace{\int_\Omega (\nabla u \cdot \nabla \eta + u\eta) + \int_\Gamma u\eta}_{Au} = \underbrace{\int_\Omega f\eta}_{f} - \underbrace{\int_\Gamma -v\eta}_{Bv}$$
$$\underbrace{\int_\Gamma (\nabla_\Gamma v \cdot \nabla_\Gamma \xi + v\,\xi) + \int_\Gamma v\xi}_{Dv} = \underbrace{\int_\Gamma g\xi}_{g} - \underbrace{\int_\Gamma -u\xi}_{Cu}$$

which clearly shows the matrix form of equation (3) where $\eta \in V_\Omega$ and $\zeta \in V_\Gamma$ are first order Lagrange finite element in the bulk and on the surface, respectively. We choose to use the Gauss-Seidel coupling scheme of (12), but the user could equally well try one of the other coupling schemes.

Table 1: Convergence rates for bulk-surface example.

Refinement Level	Iteration Count	Error	Time (secs)	Convergence Order
0	20	4.09689	2	-
1	19	0.702664	2	2.54362
2	18	0.229774	6	1.61262
3	18	0.0729218	32	1.65579
4	18	0.0205689	229	1.82588
5	18	0.00548633	1952	1.90656
6	18	0.00144776	19900	1.92201

Because of the availability of an analytic solution to this problem, shown in equations (24), it is possible to calculate standard $\mathbb{L}^2$ error norms for the solution, which can then be tested for the optimal second order convergence with grid refinement expected of linear solution interpolations.

In Table 1 can be seen the convergence orders of a series of six such grid refinements, together with indications of the iteration counts and solve times it took to achieve them on a single processor.

Clearly the convergence orders are about where one would hope to see them for a single FEM problem using linear elements, indicating that nothing has been "lost" in the suggested coupling scheme adopted for two such problems.

Having now shown that the coupling of two schemes had no adverse impact on the error convergence rates with mesh refinement, we now check that coupling has no serious impact on parallel (multi-core) performance either, as measured by the efficiency defined in equation (22).

In Figure 3 (left) can be seen the percentage efficiencies, E, and solve times in seconds, for the solution on a single Intel Xeon node using 1, 2, 4, 6 or 12 cores of the bulk-surface coupled sphere problem with a chosen finite element diameter, or "cellSize" (see below) of 0.019.

Running the example: Within the example's `source` folder type `make main` at the command prompt. This will build the executable. To run the code in serial type `./main`; for parallel use type `mpiexec -np N main` where "N" is the number of processors/threads you would like to use. The volume mesh used for the example is specified within the `parameter` file within the `data` folder on the line beginning with `fem.io.macroGridFile_3d:`.

The surface mesh is generated automatically from the volume mesh when running `main`, however, due to issues within the grid management software on which the present module depends, this automatic surface generation is not possible in parallel. For parallel operation the user is recommended to run first in serial with surface.use:false on the command line (overiding that in the parameter file), thus `./main surface.use:false`, to generate the surface mesh, and then

ensure `surface.use: true` within the parameter file. This will then use the previously created surface grid with user specified location set by `fem.io.macroGrydFile_2d: surface.dgf`.

For convenience, the software is distributed with relatively small volume and surface meshes pre-prepared. For higher grid resolutions the user should adapt the "cellSize" within the `sphere.gmsh` file in the `data` folder, and then use the `gmsh` software Geuzaine and Remacle [2009] installed into the misc directory to generate a mesh file "sphere.msh", using the command

```
../../../misc/gmsh-2.8.4-Linux/bin/gmsh sphere.gmsh -3 -v 0 -format msh -o sphere.msh
```

typed from within the `data` folder.

To view the grid based "vtu" output that will have been placed inside the `output` folder after program execution, the `paraview` software Ayachit and Utkarsh [2015] is required to be installed.

Finally, the efficiency tests may be run by typing `./eff.bot` at the command prompt. For these tests the user need only adapt the "cellSize" mentioned above for checking the efficiencies on different meshes, the mesh file generation and surface grid creation are done automatically by the script.

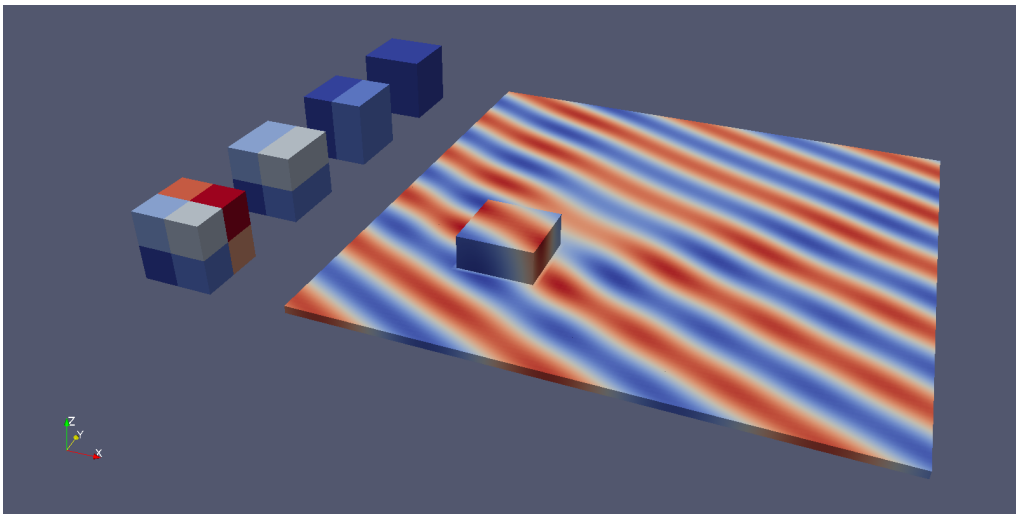

Figure 4: Real component of electromagnetic wave scattering solution to equation (25) with $k = 12$ on the surface of the box corresponding to Ω (see text) with finite element diameters 1/16 edge length, and projected onto a surrounding plane by the exterior representation formulae together with multi-core grid partitioning examples.

6.2 Direct Coupling: Scattering Electro-Magnetic Wave Example

The source code for the second case study may be found within the following folder:

dune-fem-bem-toolbox/case_studies/scattering_wave

Notes on how to run this example will be presented after the following problem description.

The classic Fem-Bem coupling example of an incident wave hitting a target, with a Fem scheme defined for the Helmholtz equation of the "total" solution – equal to the "incident" plus "scattered" solutions – with possibly spacially varying coefficients, over the finite "interior" domain of the target – a cube $\Omega = [-0.5, 0.5]^3$ in our case, and corresponding to the central box in Figure 4 – coupled to a Bem scheme on $\partial\Omega$ supporting a scattered solution to the same equation, but with constant coefficients, in the "exterior" where the incident wave originates.

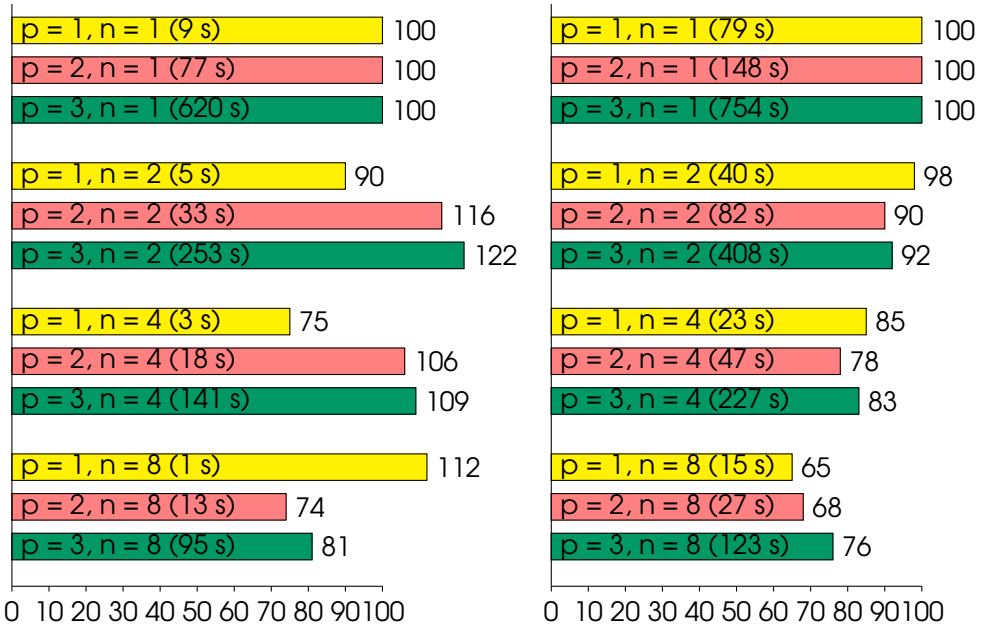

Figure 5: Percentage efficiencies, E, see equation (22), (and solve times in seconds, "s") for the solution of a simple Poisson example system (left) and system (25) in Ω with finite element diameters 1/8 edge length and $k = 6$ (right) for different core counts, "n", and interpolation polynomial orders, "p".

Now while the Bem solution is always represented by piecewise linear globally continuous basis functions (those supported by `Bem++` at present), because of the system of coupling adopted, the interior solution may use linear, quadratic and even cubic basis functions.

The values of this exterior (total) solution on an arbitrary plane around the box (discretized with linear finite elements of diameter 1/16 the side length) are also shown in Figure 4, and were computed via the appropriate exterior representation formula for the scattered field (which will be seen to be the starting point for the BEM formulations next) plus the incident field.

We require that the total solution, and its normal derivative, are continuous across the interior/exterior boundary $\partial\Omega$, and that the total (exterior) solution tends to a plane wave towards infinity at a rate inversely proportional to the distance, giving the governing equations and boundary conditions as

$$
\begin{aligned}
FEM: &\quad \Delta u + n(\mathbf{x})\, k^2 u &&= 0 && in\ \Omega && (25a)\\
BEM: &\quad \Delta v + k^2 v &&= 0 && in\ \mathbb{R}^3\backslash\Omega && (25b)\\
Dir: &\quad u - v &&= 0 && in\ \Gamma && (25c)\\
Neu: &\quad \frac{\partial u}{\partial n} - \frac{\partial v}{\partial n} &&= 0 && in\ \Gamma && (25d)\\
Rad: &\quad \left|\frac{\partial v}{\partial r} - i\,k\,v(x)\right| &&= O\left(|x|^{-1}\right) && as\ \ r \equiv |x| \to \infty && (25e)
\end{aligned}
$$

where u and v are the interior and exterior solutions, k the exterior wave-number, and $n(\mathbf{x})$ provides a spacially varying coefficient for the interior

$$n(\mathbf{x}) = \frac{1 - 0.5\exp\left(-\|\mathbf{x}\|_\infty^2\right)}{1 - 0.5\exp\left(-0.25\right)} \tag{26}$$

Clearly this coefficient is 1 on the boundary and decreases towards the centre of the cube, representing an object which is denser towards its centre.

Let the incident wave be given by

$$u^{\text{inc}} = e^{ik\mathbf{x}\cdot\mathbf{d}} \tag{27}$$

where $\mathbf{d}$ gives the direction of wave travel.

In the exterior, v can be written as $v = v^{\text{inc}} + v^{\text{sca}}$ where $v^{\text{inc}} \equiv u^{\text{inc}}$ is the incident wave and v^{sca} is the scattered wave it generates from the target.

Following the classic boundary element derivations, see Colton and Kress [1983], for the discretization of equation (25b) in the exterior region $\Omega^\infty = \mathbb{R}^3 \backslash \Omega$, we start with Green's Exterior Representation Formula – which is only valid for scattered fields and is used for the plotting on the plane mentioned above – for this exterior region

$$u^{sca}(x) = -\hat{V}\psi + \hat{K}\phi \quad \forall x \in \Omega^\infty \tag{28}$$

in terms of single, $\hat{V}$, and double, $\hat{K}$, layer potential operators defined in the appropriate Hilbert spaces

$$\hat{V} : H^{-1/2}(\Gamma) \to H^1(\Omega^\infty) \qquad \hat{K} : H^{1/2}(\Gamma) \to H^1(\Omega^\infty)$$

and acting on total Dirichlet, ϕ, and Neumann, ψ, data located on the surface of the exterior region Γ.

Now "bringing down" the unknown, $u^{sca} = u^{sca}(x)$, on to the surface, Γ, and then adding the incident field in order to identify it with the total Dirichlet data there, introduces a jump to the double layer operator, and equation (28) becomes

$$\underbrace{V\psi}_{Dv} = \underbrace{u^{\text{inc}}}_{g} - \underbrace{\left(\frac{1}{2}I - K\right)\phi}_{Cu} \quad \forall x \in \Gamma \tag{29}$$

where the operators have taken on the true BEM forms

$$V : H^{-1/2}(\Gamma) \to H^{1/2}(\Gamma) \qquad K : H^{1/2}(\Gamma) \to H^{1/2}(\Gamma)$$

Note that ϕ is the Dirichlet data defined on the boundary grid which due to the coupling conditions is the trace of the solution u from the bulk solution. Since `Bem++` only accepts piecewise linear Dirichlet functions computing Cu involves the projection of the trace of the bulk solution onto the space of piecewise linear functions on the surface grid. Having established the BEM formulation, the FEM derivations start with the weak form of equation (25a)

$$\underbrace{\int_\Omega \nabla u \cdot \nabla \eta - k^2 \int_\Omega n^2 u\,\eta}_{Au} = \underbrace{0}_{f} + \underbrace{\int_{\partial\Omega} \psi\eta}_{Bv} \tag{30}$$

where $\psi = \frac{\partial u}{\partial n}$ is identical to the Neumann data of the BEM formulation.

With the corresponding matrices A, B, C and D of equation (3) for this problem as indicated, where the Dirichlet data ϕ is identical to the solution u of the first problem on the boundary Γ, and the

Neumann data ψ is chosen as the solution v of the second problem, the coupling scheme (16) of section 2 with a GMRes Krylov iterative solver may then be used to find a solution to the complete problem.

For parallel processing in the numerical implementation of the scattering problem, the cube mesh is subdivided into block shaped pieces, as demonstrated to the left of Figure 4, with assembly and solution of the system matrix corresponding to each piece performed by the FEM scheme with one piece per rank using MPI communications to keep the calculations in sync.

The mesh of the surface of the cube, however, is not subdivided, but passed to the bem scheme whole. The use of multiple threads within the BEM solver is performed via the TBB libraries.

Glue objects are created, one per rank, to transport Dirichlet and Neumann data between the whole surface grid used by the BEM scheme, to the portion of that surface coincident with the surface of the volume grid piece located on the particular thread.

As might be expected from the nature of the spacially varying coefficient for the interior seen in equation (26), the image seen in Figure 4 shows the diffraction of the incident wave as it passes through the cubic block of material with the spacially varying density, calculated after 74 iterations of the coupling scheme. This generates the classic "spokes" of increased and decreased wave amplitudes radiating away from the diffracting object itself. More discussion of such phenomena may be found in Heald and Marion [2012].

As with the previous example, an efficiency test is also provided for the present scattering wave example, the results of which are shown in Figure 5 (right) for the calculation on 1, 2, 4 or 8 cores of the full system (25) but now with finite element interpolation polynomials of orders 1 (yellow), 2 (pink) and 3 (green) for the interior.

To keep computation times practical for easy reproduction by the end-user when trying the cubic interpolations, the finite element diameters used in the efficiency tests here are double those used for Figure 4, with the wavenumber k halved to keep the solutions meaningful on this coarser discretization – it is generally recommended to have at least 5-6 (linear) elements per wavelength in numerical computations.

As indicated above, because for this example a fixed-point iterative coupling scheme is used, the H-matrix option of `Bem++` was turned off, because the solve steps need to be repeated many times during the convergence process, but they only require one assembly performed right at the beginning. However, it can be seen that the parallel performance is just slightly worse than with the previous example, where only FEM calculations were involved, or similarly with the FEM only test (simple real valued Poisson example on a unit cube), whose results, with element diameters of 1/52 edge length, are placed in Figure 5 (left) for direct comparison.

It is also worth noting the slight but significant improvement in efficiencies with the higher order interpolation polynomials. This is attributed to increases in the overall problem size, and thus lower communication costs between the cores, relative to their overall workloads.

Running the example: Within the example's `source` folder type `make main` at the command prompt. This will build the executable. Adapt the `parameter` file in the `data` folder to the settings required – the user may even turn the H-matrices on if curious as to their effect by setting `bempp.hmatrix:true` – the parameters prefixed with "helmholtz" will control the incident field to be used (via "helmholtz.omega" for the wave number and "helmholtz.amplitude" for the amplitude) and the characteristics of the box density via the "helmholtz.boxwave" parameter. If the latter is set to zero, then the result shown in Figure 4 for the density of equation (26) will be obtained, while alternative values n greater than zero will give the box a constant relative density of n relative to the surrounding medium. To run the code in serial type `./main`; for parallel execution use `mpiexec -np N main` where "N" is the number of threads you would like to use. Note that the `Bem++` part usually just uses however many processors it can find, which is fine for most situations, however, to specify the parallelization completely, the `Bem++` thread count environment variable should be set with `export BEMPP_NUM_THREADS=N`, as is done in the

`eff.bot` script for efficiency testing. The volume mesh used for the example is specified within the `parameter` file within the `data` folder on the line beginning with `fem.io.macroGridFile_3d:`. The surface mesh is generated automatically from the volume mesh, however, due to issues within the grid management software on which the present module depends, this automatic surface generation is not possible in parallel. For parallel operation the user is recommended to run first in serial to generate the surface mesh, with `surface.use: false` within the parameter file, or on the command line as `./main surface.use:false`, and then reset `surface.use: true` afterwards for all subsequent operations which will then use the previously created surface grid with user specified location set by `fem.io.macroGrydFile_2d: surface.dgf`.

To alter the refinement of the mesh, the grid file `unitcube-3d.dgf` in the `data` folder may be adapted, and a new surface mesh will then be required by following the above mentioned steps. For convenience the code is distributed with the surface mesh matching the 16x16x16 subdivision in `unitcube-3d.dgf`, which is that used for Figure 4. The grid refinement chosen should bear in mind the wavenumber, k, of solution to be calculated – high wavenumbers computed on overly coarse grids can lead to unexpected results, or even non-convergence of the solver.

The type of grid based "vtu" output may be chosen by the user through the `external_grid.level` parameter; -10 will give no output, -1 will return the vtu output just for the solution within the box, while 0 will give both the solution within the box domain and the exterior projection onto the surface plane as seen in Figure 4. Note that vtu output will have been placed inside the `output` folder, and the `paraview` software is required to be installed in order to view it.

Finally, the efficiency tests may be run by typing `./eff.bot` at the command prompt.

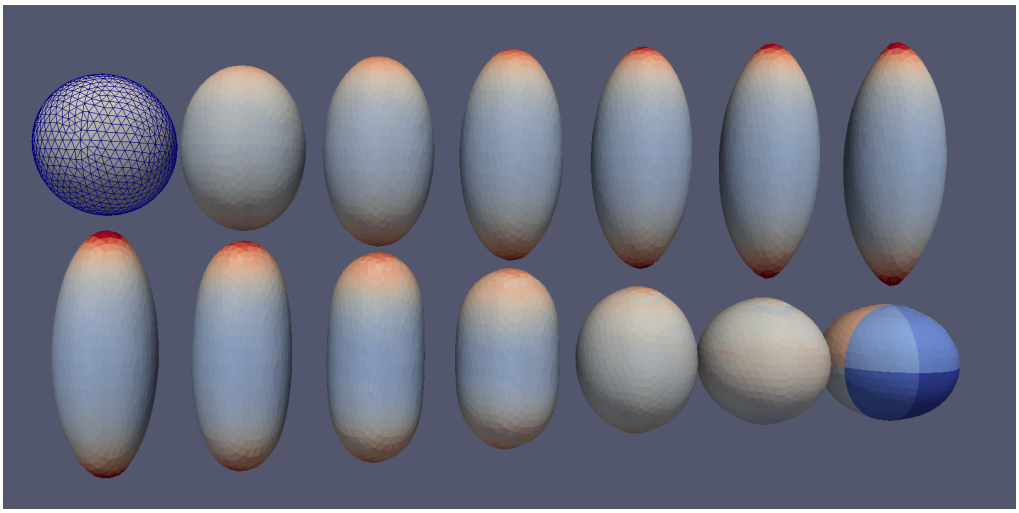

Figure 6: Non-dimensional surface charge concentration on the deformed droplet at various times of extension (top row) and relaxation (bottom row). The mesh and grid partitioning, into 12, used for the actual calculations are as indicated in the first and last images respectively. Note the point of maximum deformation – and the formation of a conical tip Radcliffe [2013] – in the top right image.

6.3 Indirect Coupling: Charged Droplet Example

The source code for the third, and final, case study may be found within the following folder:

dune-fem-bem-toolbox/case_studies/charged_droplet

Notes on how to run this example will be presented after the following problem description.

This is a more advanced example application and shows use of the toolbox outside of the direct FEM-BEM coupling methods of (12), (11) or (16), with the FEM and BEM schemes now computing very different quantites and the coupling between the two being less direct or immediate than in the previous examples.

As with the previous case, however, a BEM solution at one iteration provides Neumann data directly for use as a boundary condition to the next iteration of the FEM scheme. However, the return influence of the FEM on the BEM, while still using Dirichlet data as previously, is slightly more sophisticated.

Instead of the Dirichlet data on the boundary from a FEM solve simply being given directly to the BEM scheme, in this example it is used instead to move the actual boundary Γ on which the BEM integral kernels themselves are calculated. The BEM calculation itself on this new boundary then just uses a constant value in place of the Dirichlet data it might otherwise have received directly from the FEM.

Because the boundary Γ is constantly moving, this iterative strategy does not converge to a fixed point as previously, but provides a simple yet effective means to solve a real world moving boundary problem of great interest.

Furthermore, this motion of the boundary on which the BEM integrals are calculated necessitates a reassembly of the BEM operators every timestep too. For this reason the H-matrix option of `Bem++` mentioned above is used to significantly reduce the operator assembly times.

The exact details of the problem may be found in Radcliffe [2013], but essentially a bulk, incompressible, Navier-Stokes volume FEM scheme is used to determine the vector fluid velocities, $\mathbf{u}$, of an initially spherical (with radius r) droplet interior, $\Omega \subset \mathbb{R}^3$, driven by the normal, $\mathbf{n}$, directed surface tractions on its boundary Γ arising from both a scalar electric surface charge distribution (proportional to the normal derivative of a Laplacian solution v, calculated by a surface BEM scheme) and from classical surface tension.

The strength of this surface tension at any time and position is found by means of a second FEM scheme, but this time just for the surface Γ, solving the classical "mean curvature flow" problem which gives the vector solution, $\mathbf{w}$, corresponding to the new positions each point of Γ would take if displaced in the local normal direction by a distance proportional to (twice) the local mean curvature.

Given that the required surface tension strength is directly proportional to local mean curvature, and acts in the normal direction, the mean curvature flow solution $\mathbf{w}$ can thus be used immediately in the calculations.

Indeed, it is remarked here that the orientation of the displacements in the FEM calculated vector $\mathbf{w}$ after each time increment is actually what is used to establish the normal direction $\mathbf{n}$ in a robust manner in the computations themselves. With a simple first order semi implicit time discretization of the Navier-Stokes equations – and in contrast to Radcliffe [2013], using linear piecewise continuous basis functions with a simple pressure stabilization – the problem may be cast into the form of the previous examples by a multiplication through by the timestep τ.

With initial conditions of $\mathbf{u} \equiv \mathbf{0}$ and $\mathbf{w} = \mathbf{x}(\Gamma)$, the initial (discretized) position of Γ, the governing equations for the scalar v, and the vector $\mathbf{u}$ & $\mathbf{w}$, valued unknowns (with a minus "-" superscript

indicating values at a previous timestep) are then

$$
\begin{aligned}
FEM: &\quad -\kappa\,\Delta\,\mathbf{u} + \rho\,\mathbf{u} &&= L\left(\mathbf{u}^-\right) && in\ \Omega && (31a)\\
FEM: &\quad \nabla\cdot\mathbf{u} &&= 0 && in\ \Omega && (31b)\\
FEM: &\quad -\tau\,\Delta_\Gamma\,\mathbf{w} + \mathbf{w} &&= \mathbf{w}^- && on\ \Gamma && (31c)\\
Rob: &\quad \frac{\partial\mathbf{u}}{\partial n} - a\,\mathbf{w} + \mathbf{b}\left|\frac{\partial v}{\partial n}\right|^2 &&= -a\,\mathbf{w}^- && on\ \Gamma && (31d)\\
BEM: &\quad \Delta\,v &&= 0 && in\ \mathbb{R}^3\backslash\Omega && (31e)\\
BEM: &\quad v &&= Const. = 1 && on\ \Gamma && (31f)
\end{aligned}
$$

where we have scalar, κ & a, and vector, $\mathbf{b}$, valued coefficients involving the density, ρ, surface tension, γ, dynamic viscosity, μ, total surface charge Q, and the electrical permitivity of free space, ϵ_0, respectively

$$\kappa = \tau\,\mu\,\theta \qquad a = \frac{\gamma}{\tau} \qquad \mathbf{b} = \frac{Q^2}{r^3\epsilon_0 q^2}\,\mathbf{n}$$

Here $q = \int_\Gamma \partial_n v\,ds$ just allows us to remove the influence of the otherwise arbitrary constant = 1 in (31f) and $\theta \in [0,1]$ determines how implicitly the Laplace term is to be treated – the associated source code has $\theta = 0.5$ at present – and the linear operator L is given by

$$L\left(\mathbf{u}\right) = \rho\,\mathbf{u} + \left(\tau\,\mu - \kappa\right)\Delta\,\mathbf{u} - \tau\,\rho\left(\mathbf{u}\cdot\nabla\right)\mathbf{u} - \tau\,\nabla\,p \tag{32}$$

with a pressure, p, arising as the Lagrange multiplier for the incompressibility condition, equation (31b), in the usual manner.

Non-dimensionalization on a unit radius drop, $r \equiv 1$, via the viscocity and charge by setting $\mu \equiv 1$, indicates that there are just two *independent* parameter groupings, $Oh = \mu/\sqrt{\rho\,r\,\gamma}$ and $\chi = Q^2/\left(64\pi^2\epsilon_0\gamma\,r^3\right)$ allowing $\rho \equiv \gamma \equiv Oh^{-1}$, see Radcliffe [2013], and thus

$$\kappa = \tau\,\theta \qquad a = \frac{1}{\tau\,Oh} \qquad \mathbf{b} = \frac{64\pi^2\chi}{Oh\,q^2}\,\mathbf{n}$$

It is interesting to note that here we only have one formal boundary condition, equation (31d), that involving the normal derivative of the (interior) solution u that allows the BEM to influence the FEM.

As indicated above, the return influence of the FEM on the BEM is still via the undifferentiated solution, however it is just that this solution (now the interior fluid velocity) instead advects the very boundary on which the BEM integrals are located.

Following the BEM discretization, see equation (29), of the Helmholtz equation (25b) for the previous example, the BEM form of the much simpler Laplace equation (31e), which has a solution in the form of a potential, is similar, but lacks the double layer operator or incident field and uses constant Dirichlet data $\phi = Const.$

$$\underbrace{V\psi}_{D\,v} = \underbrace{0}_{g} - \underbrace{(-I)\,\phi}_{C\,u} \quad \forall x \in \Gamma \tag{33}$$

Thus we simply invert the single layer potential operator, V, onto constant Dirichlet data, $\phi \equiv v$, to obtain the Neumann data, ψ, that is needed to determine the surface charge – all the variation

in this Neumann data comes from the shape of the boundary Γ on which the V was calculated, rather than from the Dirichlet data it acts upon.

For the non-dimensionalization above to work, the constant *Const.* should ideally be such that the integral over Γ of v is equal to one. However, owing to the linear nature of equation (33), in practice it is preferable to simply use unit constant Dirichlet data, $\phi \equiv Const. = 1$, as in (31f), and just divide the resulting $v \equiv \psi$ distribution by whatever the integral over Γ of v turns out to be, or the q mentioned above.

Given the great similarities between equations (31a) and (31c) with equation (25a), it follows that their FEM forms are almost identical

$$\underbrace{\int_\Omega \kappa \nabla u \cdot \nabla \eta - \int_\Omega \rho\, u\, \eta}_{A\,u} = \underbrace{\int_\Omega L\,(\mathbf{u}^-)\,\eta}_{f} \underbrace{- \int_\Gamma \mathbf{a}\,(w - w^-)\,\eta + \int_\Gamma \mathbf{b}\, v^2 \eta}_{B\,v} \tag{34}$$

which is accompanied by the FEM form of the incompresibility condition (31b)

$$\int_\Omega \nabla \cdot \mathbf{u}\, p = 0 \tag{35}$$

in the volume, and on the surface there is

$$\int_\Gamma k \nabla w \cdot \nabla \eta - \int_\Gamma w\, \eta = w^- \tag{36}$$

for the FEM discretization of equation (31c).

The possible matrices A, B, C and D of equation (3) have been indicated for completeness but, as described above, the iterations in this example do not lead to a fixed point solution, but rather a sequence of solutions that may be seen in Figure 6, computed using a relatively small mesh of 1400 elements (see "Running the example" below).

Note that to follow the changes in geometry, the computational mesh used for the calculations was required to move between iterations. Now while the calculations for v and $\mathbf{w}$ are dependent only on the instantaneous shape of Γ at any particular iteration, the solution $\mathbf{u}$ involves a "history" bound-up in the operator L on the solution at the previous iteration $\mathbf{u}^-$. To accomodate the fact that this $\mathbf{u}^-$ is defined on a (slightly) different mesh, an ALE technique was adopted with a Laplacian mesh smoothing algorithm to set the interior mesh points once the Γ position had been updated, see Radcliffe [2016] for the details. Within the code this necessitated a further simple FEM scheme for the Laplacian operator to calculate these interior mesh node positions.

In Figure 6 can be seen a series of snapshots showing the evolution of the Γ boundary over time. In the top row of images electric charge, v, accumulates at the top and bottom of the droplet, causing it to deform from the sphere towards the rightmost image, where the parameter χ is reduced from 1 to 0.7 to correspond to an expulsion of charge in a liquid jet – images from experiments may be seen in Giglio et al. [2008] – which incurrs a little fluid loss Radcliffe [2016]. This reduction in overall charge loading allows surface tension, $\propto w$, to regain dominance and restore the droplet shape back to a sphere again in the lower set of images. More discussion on this very interesting behaviour may be found in Radcliffe [2013, 2016].
As with the previous examples, efficiency testing has also been performed here, with the results shown in Figure 3 (right), where they compare very favourably with those of the first case study which didn't involve `Bem++` at all, Figure 3 (left). Note that, to give the parallelization "more to work on" a more refined "maxi" mesh of 18239 elements was used, but, to keep the overall running time of the efficiency tests reasonable, a very short end time (0.01) was adopted.

Thus we may conclude that the parallelization of both the assembly (using H-matrix methods) of the BEM operators, and their solution via Krylov methods drawn from the `Eigen` library has gone quite well, with no significant impact on parallel performance.

Running the example: Within the example's `source` folder type `make main` at the command prompt. This will build the executable. To run the code in serial type `./main`; for parallel use `mpiexec -np N main` where "N" is the number of threads you would like to use. Note that the `Bem++` part usually just uses however many processors it can find, which is fine for most situations, however, to specify the parallelization completely, the `Bem++` thread count environment variable should be set with `export BEMPP_NUM_THREADS=N`, as is done in the `eff.bot` script for efficiency testing. Along with various parameters controlling the dynamics of the droplet and explained in comments above each one, the volume mesh used for the example is specified within the `parameter` file within the `data` folder on the line beginning with `fem.io.macroGridFile_3d:`. The user may create their own volume meshes with varying element diameter ("cellSize"), say, by adapting the `sphere.gmsh` mesh command file and then using the mesh generation software `gmsh` Geuzaine and Remacle [2009] installed into the misc directory to execute these commands and generate a mesh file "sphere.msh", using the command

```
../../../misc/gmsh-2.8.4-Linux/bin/gmsh sphere.gmsh -3 -v 0 -format msh -o sphere.msh
```

typed from within the `data` folder.

The surface mesh is generated automatically from the volume mesh when running the `main` program, however, due to issues within the grid management software on which the present module depends, this automatic surface generation is not possible in parallel. For parallel operation the user is recommended to run first in serial with `surface.use: false` within the parameter file, or on the command line as `./main surface.use:false`, to generate the surface mesh, and then return `surface.use: true` for the parallel runs. This will then use the previously created surface grid with user specified location set by `fem.io.macroGrydFile_2d: surface.dgf`.

To view the grid based "vtu" output that will have been placed inside the `output` folder, the `paraview` software is required to be installed. To adapt to different speed/memory/result requirements, different amounts of output are possible, specified by `vtu.output: n`, where "n" indicates the level required. n=0 gives no output; n=1 just the surface charge and n>10 all output.

For convenience the code is distributed with two sets of volume and surface meshes: a "mini" set involving 1400 elements for reproducing the results shown in Figure 6 in a moderate amount of time with a `droplet.timestep: 0.01`, and a more refined "maxi" set using 18239 elements that may be used for the efficiency tests which may be run by typing `./eff.bot` at the command prompt; with a recommended `droplet.timestep: 0.0023`. The latter command will run the simple bash script within the file `eff.bot`. For the efficiency testing it is suggested the user adapt the `droplet.endtime:` within the `parameter` file to something of their preference less than 3 or so – a value of just 0.01 was used for Figure 3 (right). At present values of 15 or above will allow reproduction of the full deformation pathway with the "mini" meshes as shown in Figure 6 – provided the appropriate vtu output is chosen – but might be a bit time consuming for the lower processor counts in efficiency testing, where it is additionally noted that such vtu output should be turned off for optimal performance.

7 Conclusions

A program structure has been presented that allows the combined use, within the same computation, of two computational discretization schemes, FEM and BEM, with very different computational architecture requirements for optimal performance.

Based on local interactions between the values of the discretized variables at the mesh points, the finite element method is well suited to distributed memory machines, while the global interactions between such discretized values of the boundary element method are best accomodated on shared memory architectures.

By the asyncronous evaluation of one scheme, and then the other, with communication steps between, it is possible to have optimal environments for each scheme on a machine with just shared memory.

A computational toolbox for the development of such programs has been presented, with three-dimensional examples of the electromagnetic scattering off a box, and the deformation of a charged droplet used to give an indication of the performance improvements with processor thread count.

In this first version of the module some restrictions still apply most notably on the partitioning of the bulk and surface grid. This restriction will be removed in a further release of the module. The use of the Dune-Grid-Glue module also allows us to use non matching surface and bulk grids. We have not made use of this feature in the test cases presented here but we have first results for the wave scattering problem with finer bulk grids which look promising. Finally fully non-linear coupling is not yet possible and will be addressed in the next release of this module. This will also allow us to perform on larger scale distributed memory machines for the bulk problem. Since `Bem++` requires an undistributed grid, the surface would still be located on a single node and distributed between threads based on the approach described above. The bulk domain could be distributed differently and make use of multiple cores. This will require some additional investigation with respect to optimal load balancing of the coupled problem.

Finally this module still uses the 2.3 release of the Dune modules. This is due to the fact that `Bem++` is still based on this release. As soon as `Bem++` has been moved to use the newer 2.4 Dune release, then a 2.4 release of this module should follow shortly afterwards.

A Structure of this module

- `dune/fem_bem_toolbox`:
  - `fem_objects`: schemes and model classes for fem problems,
  - `bem_objects`: schemes and model classes for bem problems.
  - `coupling_schemes`: coupling schemes.
  - `data_communication`: main class for gluing of surface/bulk meshes and communication of data between the grids.
  - `load_balancers`: special load balancers for matching surface/bulk grid partitioning.
  - `shared_memory_vectors`: wrapper class for dof vectors using (shared) raw memory blocks.
  - `fluid_models`: schemes and model classes for current and future fluid problems.

  In addition the `dune` folder contains some fixes to other core modules which will not be required once this module have been made compatible with the 2.4 release.
- `misc`: internal Dune configuration management files
- `src`: internal Dune configuration management files
- `doc`: documentation for the module
- `misc`: collection of useful bash scripts for installation of the module and downloading of dependent software/libraries – see installation notes below).
  - `mega.build`: bash script using autotools for building entire module and downloading+configuring all dependent software
  - `mega.make`: bash script using CMake for building entire module and downloading+configuring all dependent software (for 2.4 release)
- `case_studies`: source code for all five case studies.

- `fem_test`: pure fem problem.
- `bem_test`: pure bem problem.
- `coupled_sphere`: coupled bulk fem-surface fem test case.
- `scattering_wave`: coupled bulk fem-surface bem wave scattering problem.
- `charged_droplet`: coupled bulk fem / surface fem + surface bem charged droplet example.

Each case study folder contains three subfolders:

- `source`: the `main.cc` file required to run the simulation.
- `data`: grid files and a file `parameter` containing run time parameters for this example.
- `output`: folder where the simulation output is stored.

B Installation notes

This module is licenced under GPL Version 2. Warning: All the software described in this work has been downloaded, built, run and tested on Linux machines; use on other systems has not been tried, however, if built without `Bem++`– see below – the Dune-Fem-Bem-Toolbox should have the same platform requirements as all other Dune modules.

To obtain the Dune-Fem-Bem-Toolbox module, the user should clone the `releases/2.3` branch from the Dune user website by typing the following at the command prompt in their chosen installation directory:

```
git clone -b releases/2.3 \
https://gitlab.dune-project.org/andreas.dedner/dune-fem-bem-toolbox.git
```

Once this is done, to aid the user in the installation of all the remaining software and the compilation and general set-up of all the source code and examples presented, a bash shell script called `mega.build` is included in the "misc" miscellaneous folder. This script also allows setting-up of the module in a reduced state without the `Bem++` library, and thus just using Dune modules. The scattered wave, *charged* droplet and, of course, bem comparison examples requiring calculation of bem integrals will then be unavailable, however, the FEM comparison and coupled sphere examples will still work, and the droplet example may still be run, although in an *uncharged* state, to give simple oscillations for example.

Before running the script, the user should adapt the `CC` and `CXX` statements at the head of the file to the latest C and C++ compilers available on their system respectively. Additionally, for build efficiency, should a `PYTHON` or `CYTHON` distribution already be available, then typing `export PYTHONPATH=/path/to/my/cython` will avoid the `Bem++` installation process from downloading its own version.

Once the C/C++ compilers have been set, to load all the dependencies and build everything for the toolbox module simply type:

```
./mega.build bem
```

at the comand prompt from within the `misc` directory.

This build script will install and configure some of the ("non-core") dependant Dune modules at the same folder hierachy level as the Dune-Fem-Bem-Toolbox module itself, so if the user already has Dune modules around, it is suggested to checkout the Dune-Fem-Bem-Toolbox into an empty installation folder when using this build script.

The `Bem++` software, and any dependencies it builds itself, will be installed within the `mod` folder at the top level within the Dune-Fem-Bem-Toolbox. The user may also notice that `gmsh` software Geuzaine and Remacle [2009] useful for mesh generation (see the running the examples notes above), is also installed by this script, but within the `misc` folder at the same top level.

Note that Bem++ installs most of its dependencies if required (including the DUNE "core" modules requiring the release version 2.3). The dependency on `Cython` can be problematic and might require the user to install that package (version greater or equal to 0.21). separately which on most systems is possible using `pip install --user cython`. Furthermore, Bem++ is incompatible with certain versions of `Eigen`. Bem++'s build system will download a compatible version if `Eigen` is not found on the system. A pre installed `Eigen` can thus lead to problems also with the include paths used while building this module.

There may also be problems installing the `TBB` package on some systems, and if this occurs, it is recommended to preinstall this software too, probably from a tarball downloaded from the TBB website `https://www.threadingbuildingblocks.org`.

Bem++ will install `Boost` version 1.57 if no version is found on the system during the build process.

To load all the dependencies and build everything *without* *Bem++* simply type

```
./mega.build nobem
```

at the comand prompt within the `misc` directory. Note: `nobem` is also the default setting should no explicit choice be made.

To reload and build just the DUNE dependencies (for dune updates, say) without rebuilding the Bem++ library type

```
./mega.build bembutnobembuild
```

at the comand prompt within the `misc` directory.

C Known Issues

Inevitably, as with any new software release, there will be a few issues affecting the structure and performance of the software that remain to be sorted out. These are principally related to issues within the supporting software upon which the present module is built, and over which the authors have little, or no, influence. Thus certain accomodations have been made in the present 2.3 release to adapt to these issues.

Issue : No projection of `ALUGrid` surfaces in refinements in parallel

Effect : Convergence testing of coupled sphere example may only be done in serial.

Issue : Memory leaks (either local or within Bem++) with destruction of BEM objects for charged drop example when using Bem++. Note this does *not* effect the program execution, but merely the internal clean-up on completion.

Effect : Significant internal system output on program completion.

Acknowledgments

The authors gratefully acknowledge the financial assistence of EPSRC under grant number EP/K038060/1 for the support of the second author during his stay at Warwick university, and the suggestions of the reviewers for improving the manuscript.

References

M. Alkaemper, A. Dedner, R. Kloefkorn, and M. Nolte. The dune-alugrid module. *Archive of Numerical Software*, 4(1), 2016.

Ayachit and Utkarsh. *The ParaView Guide: A Parallel Visualization Application*. Kitware, 2015. ISBN 978-1930934306. URL http://www.paraview.org.

P. Bastian, G. Buse, and O. Sander. Infrastructure for the coupling of dune grids. In *Proceedings of ENUMATH 2009*, pages 107–114, 2010.

E. G. Boman, U. V. Catalyurek, C. Chevalier, and K. D. Devine. The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring. *Scientific Programming*, 20(2), 2012. http://www.cs.sandia.gov/zoltan/.

U. Brink and E.P. Stephan. Adaptive coupling of boundary elements and mixed finite elements for incompressible elasticity. *Numer. Methods Partial Differential Equations*, 17:79–92, 2001.

D. Colton and R. Kress. *Integral Equation Methods in Scattering Theory*. John Wiley & Sons, 1983.

M. Costabel. Symmetric methods for the coupling of finite elements and boundary elements. *Boundary Elements*, 9:411–420, 1987.

Dune. Distributed and unified numerics environment. http://www.dune-project.org, 2012.

dune-fem howto. Howto module for the dune-fem package. http://users.dune-project.org/projects/dune-fem-howto/repository, 2015.

M.. Feischl, T.. Führer, D.. Praetorius, and E. P.. Stephan. Optimal preconditioning for the symmetric and nonsymmetric coupling of adaptive finite elements and boundary elements. *Numerical Methods for Partial Differential Equations*, 2015. ISSN 1098-2426. doi: 10.1002/num.22025. URL http://dx.doi.org/10.1002/num.22025.

D. R. Gaston, C. J. Permann, J. W. Peterson, A. E. Slaughter, D. AndrÅa, Y. Wang, M. P. Short, D. M. Perez, M. R. Tonks, J. Ortensi, L. Zou, and R. C. Martineau. Physics-based multiscale coupling for full core nuclear reactor simulation. *Annals of Nuclear Energy*, 84:45 – 54, 2015. ISSN 0306-4549. doi: http://dx.doi.org/10.1016/j.anucene.2014.09.060. URL http://www.sciencedirect.com/science/article/pii/S030645491400543X. Multi-Physics Modelling of {LWR} Static and Transient Behaviour.

G.N. Gatica and N. Heuer. A dual-dual formulation for the coupling of mixed-fem and bem in hyperelasticity. *SIAM J. Num. Anal.*, 38:380–400, 2000.

C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *Int. J. Num. Meth. Eng.*, 79:1309–1331, 2009.

E. Giglio, B. Gervais, J. Rangama, B. Manil, and B.A. Huber. Shape deformations of surface-charged micro-droplets. *Phys. Rev. E*, 77:0363191–7, 2008.

G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

M.A. Heald and J.B. Marion. *Classical Electromagnetic Radiation*. Dover, 3 edition, 2012.

M. A. Heroux, R. B. Brightwell, and M. M. Wolf. Bi-modal mpi and mpi+threads computing on scalable multicore systems. In *IJHPCA (Submitted)*, 2011.

R. Hiptmair. Coupling of finite elements and boundary elements in electromagnetic scattering. *SIAM J. Num. Anal.*, 41:919–944, 2003.

D. Ibanez, I. Dunn, and M. S. Shephard. Hybrid mpi-thread parallelization of adaptive mesh operations. *Parallel Comput.*, 52(C):133–143, Feb. 2016. ISSN 0167-8191. doi: 10.1016/j.parco.2016.01.003. URL http://dx.doi.org/10.1016/j.parco.2016.01.003.

C. Johnson and J.C. Nedelec. On the coupling of boundary integral and finite element methods. *Math. Comp.*, 35(152):1063–1079, 1980.

R. Klöfkorn. Efficient Matrix-Free Implementation of Discontinuous Galerkin Methods for Compressible Flow Problems. In A. H. et al., editor, *Proceedings of the ALGORITMY 2012*, pages 11–21, 2012.

R. Kriemann. H-lu factorization on many-core systems. *Computing and Visualization in Science*, 16: 105–117, June 2013. doi: 10.1007/s00791-014-0226-7.

C.C. Lin, E.C. Lawton, J.A. Caliendo, and L.R. Anderson. An iterative finite element-boundary element algorithm. *Comp. Struct.*, 59:899–909, 1996.

S. Meddahi and F.-J. Sayas. A fully discrete bem-fem for the exterior stokes problem in the plane. *SIAM J. Num. Anal.*, 37:2082–2102, 2000.

S. Meddahi and V. Selgas. A mixed-fem and bem coupling for a three-dimensional eddy current problem. *Math. Mod. Num. Anal.*, 37:291–318, 2003.

P. Mund and E.P. Stephan. Adaptive coupling and fast solution of fem-bem equations for parabolic-elliptic interface problems. *Math. Meth. Appl. Sci.*, 20:403–423, 1997.

R. Olson, J. Bentz, R. Kendall, M. Schmidt, and M. Gordon. A novel approach to parallel coupled cluster calculations:ÂĆäâ combining distributed and shared memory techniques for modern cluster based systems. *Journal of Chemical Theory and Computation*, 3(4):1312–1328, 2007. ISSN 1549-9618. doi: 10.1021/ct600366k.

A.J. Radcliffe. A comparison between a symmetric and a non-symmetric galerkin finite element - boundary integral equation coupling for the two-dimensional exterior stokes problem. *Eng. Anal. Bound. Elem.*, 35:959–969, August 2011. DOI: 10.1016/j.enganabound.2011.03.003.

A.J. Radcliffe. Fem-bem coupling for the exterior stokes problem with non-conforming finite elements and an application to small droplet deformation dynamics. *Int. J. Num. Meth. Fluids*, 68:522–536, February 2012. DOI: 10.1002/fld.2518.

A.J. Radcliffe. Non-conforming finite elements for axisymmetric charged droplet deformation dynamics and coulomb explosions. *Int. J. Num. Meth. Fluids*, 71:249–268, January 2013. DOI: 10.1002/fld.3667.

A.J. Radcliffe. Arbitrary lagrangian eulerian simulations of highly electrically charged micro-droplet coulomb explosion deformation pathways. *Int. J. Modeling, Simulation, and Scientific Computing*, 7, February 2016. DOI: 10.1142/S1793962316500161.

T. Ranner and C.M. Elliott. Finite element analysis for a coupled bulk-surface partial differential equation. *IMA Journal of Numerical Analysis*, 33(2):377 – 402, 2013. URL http://eprints.whiterose.ac.uk/77779/.

F.J. Sayas. The validity of johnson–nédélec's bem–fem coupling on polygonal interfaces. *SIAM Journal on Numerical Analysis*, 47(5):3451–3463, 2009. doi: 10.1137/08072334X.

F.J. Sayas. The validity of johnson–nédélec's bem–fem coupling on polygonal interfaces. *SIAM Review*, 55(1):131–146, 2013. doi: 10.1137/120892283.

W. Smigaj, S. Arridge, T. Betcke, J. Phillips, and M. Schweiger. Solving boundary integral problems with bem++. *ACM Trans. Math. Software*, 41:6:1–6:40, 2015.

E. P. Stephan. *Coupling of Boundary Element Methods and Finite Element Methods*. Encyclopedia of Computational Mechanics. John Wiley and Sons, Ltd., 2004.

TBB. Threading building blocks (intel tbb) 4.4 update 2. https://www.threadingbuildingblocks.org, 2014.

A. Toselli and O.B. Widlund. *Domain Decomposition Methods - Algortihms and Theory*, volume 34 of *Series in Computational Mathematics*. Springer, 2005. ISBN 978-3-540-20696-5.

L. C. Wrobel. *Applications in Thermo-Fluids and Acoustics*, volume 1 of *The Boundary Element Method*. John Wiley and Sons, Ltd., 2002.

