VŠB-TECHNICAL UNIVERSITY OF OSTRAVA
FACULTY OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE

Parallel Boundary Element Methods in Space and Time

PHD THESIS

2016 Michal Merta

Declaration of Authorship

I, Michal Merta, declare that this thesis titled, 'Parallel Boundary Element Methods in Space and Time' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where I have quoted from the work of others, the source is always given.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Abstract

Efficient parallel implementation of the boundary element method is crucial for its applicability to the solution of real world engineering problems. Several approaches to the parallelization of BEM are presented in this thesis, including in-core vectorization using SIMD instruction set extensions, a novel distributed fast boundary element method based on the cyclic graph decompositions, and parallel space-time Galerkin boundary element method for the time dependent wave equation utilizing smooth temporal basis functions. In the thesis we provide a brief introduction to the boundary element method in both space and time, describe our in-house boundary element environment, in which most of the methods are implemented, and provide a commentary to author's published papers in the field.

Key Words: boundary element method, parallel fast BEM, time-domain BEM, HPC, parallel computing.

Abstrakt

Aby bylo možné metodu hraničních prkvů využít pro řešení reálných inženýrských úloh, je potřebná její efektivní paralelní implementace. V předložené disertační práci je představeno několik přístupů k paralelizaci této metody — vektorizace pomocí SIMD instrukčních sad v rámci jednoho jádra, nová distribuovaná rychlá metoda hraničních prvků založená na cyklických dekompozicích grafů a paralelní prostoro-časová Galerkinova metoda hraničních prvků pro řešení časově závislé vlnové rovnice s využitím hladkých časových bázových funkcí. Práce obsahuje stručný úvod do metody hraničních prvků v prostoru i čase, popisuje námi vyvíjený hraničně prvkový software, ve kterém je většina z výše jmenovaných metod implementována, a poskytuje komentář k autorovým publikacím v tomto oboru.

Klíčová slova: metoda hraničních prvků, paralelní rychlé metody hračních prvků, BEM pro časově závislé problémy, HPC, paralelní počítání.

Acknowledgements

First of all I would like to thank my supervisor doc. Ing. Dalibor Lukáš, Ph.D. for introducing me to the world of the boundary element methods. I appreciate his help and support during my studies, as well as his pursuit of international cooperation thanks to which a significant part of this work has been created.

I would also like to offer my thank to prof. Ing. Tomáš Kozubek, Ph.D., Head of the Research Programme 3 at IT4Innovations National Supercomputing Center, for maintaining an inspiring work environment that allowed me to finish this work, and to doc. RNDr. Jiří Bouchala, Ph.D., Head of the Department of Applied Mathematics, for keeping the athmosphere at the department enjoyable during all the years of my studies.

Since this work is written in the form of a collection of papers, I want to thank all of the co-authors for a joint work and a fruitful cooperation.

My thanks of course belong to my colleagues and fellow students from IT4Innovations or the Department of Applied Mathematics – especially to Ing. Jan Zapletal for a long-term cooperation and friendship.

Special thanks go to my family for supporting me during both my undergraduate and graduate studies and to my friends for all the good times that we have had during those years.

Contents

List of Figures

List of Tables

Introduction

This work is presented in the form of collection of three research papers focusing on the parallelization of the boundary element method. All of them have been published or accepted in journals with impact factor. We provide them together with the commentary in Part II of the thesis. Due to the complexity of the presented topic this main part is preceded by Part I which provides a brief introduction to the boundary element method and its efficient implementation.

In the first paper an approach to the acceleration of the boundary element method (BEM) using the SIMD (single instruction multiple data) capabilities of modern processors is presented. These vector instruction set extensions enable a concurrent computation on multiple operands using a single computational core, thus provide an energy efficient way to speed-up the computation. While the original SSE instruction set extension introduced by Intel in 1999 contained 128-bit wide vector registers supporting a parallel computation on two double and four single precision operands, the current instruction set architecture available in the Intel Xeon Phi coprocessors supports a concurrent computation on up to 8 double and 16 single precision operands. The AVX-512 instruction set extension provides similar capabilities to the Intel Xeon server processor based on the Skylake architecture. Therefore to exploit the full potential of current processors one has to modify and optimize his code to enable the utilization of the SIMD instruction sets. In the presented work we propose an approach to vectorize BEM computation based on the existing high level C++ library [17]. This relatively easily enables us to vectorize the original code while keeping its object oriented structure.

The second paper deals with the parallelization of the fast boundary element method. The boundary element method for the stationary (elliptic) problems is now well established. There are techniques for ovecoming two main setbacks - the integration of singular kernels and the dense matter of the system matrices. The former one can be treated using, e.g., semi-analytic integration [24] or a fully numerical integration based on Duffy's substitution [26, 9, 6]. The latter problem is usually treated using a sparsification technique based on the low rank approximations of the system matrices, such as the adaptive cross approximation (ACA) [2, 4, 3, 5], the panel clustering method [16], or the fast multipole method (FMM) [25, 11]. Yet, to enable the solution of large scale problems, a distribution of the system

matrices and a parallelization of the solution process is necessary. Parallel implementation of the fast boundary element method in distributed memory is still an open topic, so far it has been addressed, e.g., in [4]. In this work we present a new approach for distribution of the system matrices among nodes of a computational cluster (MPI processes) based on cyclic graph decompositions.

The last paper aims at efficient parallel implementation of BEM-based solver for time-dependent wave scattering modelled by the hyperbolic partial differential equation. Although the initial ideas about the solution of time dependent problems using the boundary element method date back to 1960s [15] the research in this field is far less established than in the case of elliptic problems. The application of BEM on the governing hyperbolic time dependent wave equation is not as straightforward as in the case of elliptic equations. Besides the collocation methods, which are hard to analyse mathematically, there are two major approaches in the field of the time dependent boundary integral equations [8, 12]: the convolution quadrature method introduced by Lubich [19] and the Galerkin method presented by Bamberger and Ha Duong [1]. In this work we deal with the latter one. The main drawback of the Galerkin formulation is a need for a special quadrature when computing the elements of system matrices, since integration domains are intersections of boundary elements with a discrete light cone. Therefore we implement the new approach introduced by Sauter and Veit [27] which uses compactly supported, infinitely smooth basis functions to overcome this problem. Its efficient parallel implementation, as far as we know, has not yet been presented and tested on large scale problems.

The outline of this work is as follows: in Part I we provide a brief introduction to the boundary integral equations for both static and time-dependent problems, focusing on the Laplace and wave equation. We present the boundary integral equations suitable for the solution of Dirichlet and Neumann problems together with their variational formulations. In the next chapter we describe the discretization of the problems using the boundary element method. The spatial discretization for the elliptic problems as well as the space-time Galerkin method for the solution of the wave equation is described. In the next chapter the parallel boundary element environment BEM4I, where most of the presented techniques are implemented, is presented. In Part II we provides three of the author's published or accepted papers together with the commentary on results and the contribution of the author. The first paper focuses on the acceleration of BEM using the SIMD instruction set extensions, the next one presents the new method for a distribution of stationary BEM system matrices among computational nodes, and the last one describes a new approach for parallel solution of time-dependent wave equation. In the last chapter we provide conclusion and an outlook of the future work.

Part I

Introduction to BEM and its efficient implementation

Chapter 1

Boundary Integral Equations

Since this thesis deals with the parallelization of the boundary element method for both stationary and time-dependent problems we focus on the boundary integral equation (BIE) formulation of the Laplace and time-dependent wave equations. Formulations and theory concerning the Helmholtz, Lamé, or Stokes equations may be found in [32].

1.1 Laplace equation

In this section we provide a boundary integral formulations suitable for the solution of the Laplace equation. An interested reader should consult, e.g., [24, 32] for more details. Let $\Omega \subset \mathbb{R}^3$ be a bounded domain with Lipschitz boundary $\Gamma := \partial\Omega$. The solution of the Laplace equation

$$-\triangle u(\boldsymbol{x}) = 0 \quad \text{for } \boldsymbol{x} \in \Omega \tag{1.1}$$

is given by the representation formula

$$u(\boldsymbol{x}) = \int_\Gamma G(\boldsymbol{x},\boldsymbol{y})\gamma_1^{\text{int}} u(\boldsymbol{y})\mathrm{d}s_{\boldsymbol{y}} - \int_\Gamma \gamma_{1,y}^{\text{int}} G(\boldsymbol{x},\boldsymbol{y})u(\boldsymbol{y})\mathrm{d}s_{\boldsymbol{y}}, \tag{1.2}$$

where

$$G(x,y) := \frac{1}{4\pi\|\boldsymbol{x}-\boldsymbol{y}\|}$$

is the fundamental solution of the Laplace equation, $\gamma_1^{\text{int}}\colon H^1_\triangle(\Omega) \to H^{-1/2}(\Gamma)$ is the interior conormal derivative operator defined for functions from

$$H^1_\triangle(\Omega) := \left\{v \in H^1(\Omega) : -\triangle v \in L^2(\Omega) \text{ in the sense of distributions}\right\}$$

and satisfying

$$\gamma_1^{\text{int}} u(\boldsymbol{x}) = \frac{\partial u(\boldsymbol{x})}{\partial \boldsymbol{n}} \quad \text{for all} \quad u \in C^\infty(\overline{\Omega}), \boldsymbol{x} \in \Gamma.$$

The space $H^{-1/2}(\Gamma)$ is the dual of $H^{1/2}(\Gamma)$ with respect to the pivot space $L^2(\Gamma)$. Finally $\boldsymbol{n}$ denotes the unit outward normal vector to Ω.

The mappings

$$(\tilde{V}w)(\boldsymbol{x}) := \int_\Gamma G(\boldsymbol{x},\boldsymbol{y})w(\boldsymbol{y})\mathrm{d}s_{\boldsymbol{y}}, \qquad \tilde{V}\colon H^{-1/2}(\Gamma) \to H^1(\Omega) \tag{1.3}$$

and

$$(Wv)(\boldsymbol{x}) := \int_\Gamma \gamma_{1,y}^{\mathrm{int}} G(\boldsymbol{x},\boldsymbol{y})v(\boldsymbol{y})\mathrm{d}s_{\boldsymbol{y}}, \qquad W\colon H^{1/2}(\Gamma) \to H^1(\Omega) \tag{1.4}$$

are called the single layer potential and the double layer potential, respectively.

Applying the interior trace operator $\gamma_0^{\mathrm{int}}\colon H^1(\Omega) \to H^{1/2}(\Gamma)$ to (1.3) we define the single layer potential operator

$$V := \gamma_0^{\mathrm{int}}\tilde{V}\colon H^{-1/2}(\Gamma) \to H^{1/2}(\Gamma).$$

The operator is bounded, $H^{-1/2}(\Gamma)$-elliptic and for $w \in L^\infty(\Gamma)$ and $\boldsymbol{x} \in \Gamma$ there holds the representation [24]

$$(Vw)(\boldsymbol{x}) = \int_\Gamma G(\boldsymbol{x},\boldsymbol{y})w(\boldsymbol{y})\mathrm{d}s_{\boldsymbol{y}}. \tag{1.5}$$

Similarly, applying γ_0^{int} to (1.4) we obtain the bounded boundary integral operator

$$\gamma_0^{\mathrm{int}}W\colon H^{1/2}(\Gamma) \to H^{1/2}(\Gamma)$$

with the representation

$$\gamma_0^{\mathrm{int}}(Wv)(\boldsymbol{x}) = (-1+\sigma(\boldsymbol{x}))v(\boldsymbol{x}) + (Kv)(\boldsymbol{x}) \tag{1.6}$$

for $\boldsymbol{x} \in \Gamma$ and $v \in H^{1/2}(\Gamma)$. For $v \in L^\infty(\Gamma)$ the double layer potential operator K takes the form

$$(Kv)(\boldsymbol{x}) = \int_\Gamma \frac{\partial G}{\partial \boldsymbol{n}_{\boldsymbol{y}}}(\boldsymbol{x},\boldsymbol{y})v(\boldsymbol{y})\mathrm{d}s_{\boldsymbol{y}}. \tag{1.7}$$

Moreover, $\sigma(\boldsymbol{x}) = 1/2$ for $\boldsymbol{x} \in \Gamma$ on a smooth part of the boundary.

Applying the interior trace operator to the representation formula (1.2) and using the relations (1.5), (1.7), and (1.6) we obtain the boundary integral equation

$$\gamma_0^{\mathrm{int}}u(\boldsymbol{x}) = (V\gamma_1^{\mathrm{int}}u)(\boldsymbol{x}) + \frac{1}{2}\gamma_0^{\mathrm{int}}u(\boldsymbol{x}) - (K\gamma_0^{\mathrm{int}}u)(\boldsymbol{x}) \tag{1.8}$$

for almost all $\boldsymbol{x} \in \Gamma$.

Since for $w \in H^{-1/2}(\Gamma)$ we have $\triangle(\tilde{V}w) = 0$ in the sense of distribution, we can apply the interior conormal derivative operator γ_1^{int} to (1.3) to define the bounded boundary integral

operator

$$\gamma_1^{\text{int}}\tilde{V}\colon H^{-1/2}(\Gamma) \to H^{-1/2}(\Gamma).$$

For $w \in H^{-1/2}(\Gamma)$ and $\boldsymbol{x} \in \Gamma$ in the smooth part of the boundary there holds the representation

$$\gamma_1^{\text{int}}(\tilde{V}w)(\boldsymbol{x}) = \frac{1}{2}w(\boldsymbol{x}) + (K'w)(\boldsymbol{x}),$$

where K' is the adjoint double layer potential operator. For $w \in L^\infty(\Gamma)$ we obtain [26]

$$(K'w)(\boldsymbol{x}) = \int_\Gamma \frac{\partial G}{\partial \boldsymbol{n}_{\boldsymbol{x}}}(\boldsymbol{x},\boldsymbol{y})w(\boldsymbol{y})\mathrm{d}s_{\boldsymbol{y}}. \tag{1.9}$$

Similarly as in the case of (1.8) we derive the second boundary integral equation by applying the interior conormal derivative operator to the formula (1.2), obtaining

$$\gamma_1^{\text{int}}u(\boldsymbol{x}) = \frac{1}{2}\gamma_1^{\text{int}}u(\boldsymbol{x}) + (K'\gamma_1^{\text{int}}u)(\boldsymbol{x}) + (D\gamma_0^{\text{int}}u)(\boldsymbol{x}), \tag{1.10}$$

where D is the hypersingular operator. Since $\frac{1}{2}\gamma_1^{\text{int}}u(\boldsymbol{x})+(K'\gamma_1^{\text{int}}u)(\boldsymbol{x})$ are linear and bounded from $H^{1/2}(\Gamma)$ to $H^{-1/2}(\Gamma)$, so is D. Moreover, D is $H^{1/2}(\Gamma)$-semi-elliptic.

Combining (1.8) and (1.10) the Calderon projector is defined as

$$\begin{pmatrix}\gamma_0^{\text{int}}u\\ \gamma_1^{\text{int}}u\end{pmatrix} = \begin{pmatrix}\frac{1}{2}I - K & V\\ D & \frac{1}{2}I + K'\end{pmatrix}\begin{pmatrix}\gamma_0^{\text{int}}u\\ \gamma_1^{\text{int}}u\end{pmatrix}. \tag{1.11}$$

Let us now apply the presented relations to solution of the boundary value problems for the Laplace equations using both direct and indirect approaches.

1.1.1 Dirichlet boundary value problem

Let us start with the direct formulation of the problem. The solution of the Dirichlet boundary value problem for the Laplace equation

$$\begin{cases} -\triangle u(\boldsymbol{x}) = 0 \quad \text{for } \boldsymbol{x} \in \Omega,\\ \gamma_0^{\text{int}}u(\boldsymbol{x}) = g(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \Gamma \end{cases} \tag{1.12}$$

is given by the representation formula (1.2). To find the missing Cauchy data we employ the first equation in (1.11). This leads to the problem

$$\begin{cases} \text{Find } t \in H^{-1/2}(\Gamma), \text{ such that}\\ (Vt)(\boldsymbol{x}) = \frac{1}{2}g(\boldsymbol{x}) + (Kg)(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \Gamma. \end{cases} \tag{1.13}$$

Since the operator V is bounded and $H^{-1/2}(\Gamma)$-elliptic the problem (1.13) has a unique solution (see [32], Theorem 3.4). The variational formulation of the problem reads as

$$\begin{cases} \text{Find } t \in H^{-1/2}(\Gamma) \text{ such that} \\ \langle Vt, \tau \rangle_\Gamma = \left\langle (\frac{1}{2}I + K)g, \tau \right\rangle_\Gamma \quad \text{for all } \tau \in H^{-1/2}(\Gamma). \end{cases} \tag{1.14}$$

Here $\langle \cdot, \cdot \rangle$ denotes a duality pairing. For $g \in H^{1/2}(\Gamma)$ this problem admits a unique solution (see [24], Theorem 1.5).

Using the indirect approach we start from the representation

$$u(\boldsymbol{x}) = (\tilde{V}w)(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \Omega,$$

where $w \in H^{-1/2}(\Gamma)$ is an unknown density function, and applying the interior trace operator γ_0^{int} we obtain the problem

$$\begin{cases} \text{Find } w \in H^{-1/2}(\Gamma), \text{ such that} \\ (Vw)(\boldsymbol{x}) = g(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \Gamma. \end{cases} \tag{1.15}$$

Again, from the properties of the integral operator V we conclude a unique solvability of the problem (1.15).

Using the second equation in (1.11) as well as the properties of the double layer potential one can analogically derive two integral equations suitable for the solution of the problem (1.12). For the sake of clarity of this text we ommit these formulations and refer the reader to [24, 32] for more details.

1.1.2 Neumann boundary value problem

Let us consider the Neumann boundary value problem for the Laplace equation

$$\begin{cases} -\triangle u(\boldsymbol{x}) = 0 \quad \text{for } \boldsymbol{x} \in \Omega, \\ \gamma_1^{\text{int}} u(\boldsymbol{x}) = g(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \Gamma, \end{cases} \tag{1.16}$$

where g satisfies the solvability condition

$$\int_\Gamma g(\boldsymbol{y}) \mathrm{d}s_{\boldsymbol{y}} = 0. \tag{1.17}$$

In order to find the unknown Dirichlet data we employ the second equation in (1.11), obtaining the boundary integral equation of the first kind

$$\begin{cases} \text{Find } \bar{u} := \gamma_0^{\text{int}} u \in H_{**}^{1/2}(\Gamma), \text{ such that} \\ (D\bar{u})(\boldsymbol{x}) = \frac{1}{2} g(\boldsymbol{x}) - (K'g)(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \Gamma, \end{cases} \tag{1.18}$$

where $H_{**}^{1/2}(\Gamma) := \{v \in H^{1/2}(\Gamma) \colon \langle v, 1\rangle_\Gamma = 0\}$. Alternatively, one can consider the variational problem

$$\begin{cases} \text{Find } \bar{u} \in H_{**}^{1/2}(\Gamma) \text{ such that} \\ \langle D\bar{u}, v\rangle_\Gamma = \left\langle (\frac{1}{2} I - K')g, v \right\rangle_\Gamma \quad \text{for all } v \in H_{**}^{1/2}(\Gamma). \end{cases} \tag{1.19}$$

The unique solution $\bar{u}_\alpha = \bar{u} + c$ is obtained afterwards using the scaling condition $\langle \bar{u}_\alpha, 1\rangle = \int_\Gamma \bar{u}_\alpha(\boldsymbol{y}) \mathrm{d}s_{\boldsymbol{y}} = \alpha$. Another approach is to solve the extended variational problem

$$\begin{cases} \text{Find } \bar{u} \in H^{1/2}(\Gamma) \text{ such that} \\ \langle D\bar{u}, v\rangle_\Gamma + \langle \bar{u}_\alpha, 1\rangle_\alpha \langle v, 1\rangle_\Gamma = \left\langle (\frac{1}{2} I - K')g + \alpha \langle v, 1\rangle, v \right\rangle_\Gamma \quad \text{for all } v \in H^{1/2}(\Gamma). \end{cases} \tag{1.20}$$

For $g \in H^{-1/2}(\Gamma)$ and $\alpha \in \mathbb{R}$ the problem (1.20) admits a unique solution $\bar{u} \in H^{1/2}(\Gamma)$. Moreover, if g satisfies (1.17) then $\bar{u}_\alpha$ solves (1.18) with the appropriate scaling condition (see [24], Theorem 1.9).

For a detailed description of the formulation using the second kind boundary integral equation, as well as the indirect approach, we refer the reader to the above-mentioned literature.

1.2 Wave equation

Let us consider the initial boundary value problem for the homogeneous wave equation in an unbounded domain $\Omega^+ \in \mathbb{R}^3$

$$\begin{cases} \frac{1}{c^2}\frac{\partial^2 u^{\mathrm{e}}}{\partial t^2}(\boldsymbol{x},t) - \triangle u^{\mathrm{e}}(\boldsymbol{x},t) = 0 \quad \text{for } (\boldsymbol{x},t) \in \Omega^+ \times \mathbb{R}^+, \\ u^{\mathrm{e}}(\boldsymbol{x},0) = 0 \quad \text{for } \boldsymbol{x} \in \Omega^+, \\ \frac{\partial u^{\mathrm{e}}}{\partial t}(\boldsymbol{x},0) = 0 \quad \text{for } \boldsymbol{x} \in \Omega^+, \\ \mathcal{B}u^{\mathrm{e}}(\boldsymbol{x},t) = g(\boldsymbol{x},t) \quad \text{for } (\boldsymbol{x},t) \in \Gamma \times \mathbb{R}^+, \end{cases} \tag{1.21}$$

where c is the speed of sound in a given medium (considered to be 1 in what follows) and $\Omega^+ := \mathbb{R}^3 \setminus \overline{\Omega^-}$ is the exterior of an open bounded domain Ω^- with Lipschitz boundary $\Gamma := \partial\Omega^-$. Let $g(\boldsymbol{x},t) := 0$ for $t \leq 0$ and $g(\boldsymbol{x},t) := -\mathcal{B}u^{\mathrm{inc}}$ for $t > 0$. In this case the solution u of (1.21) represents the wave scattered off Ω^- when being hit by an incoming wave u^{inc} (see Figure 1.1). The boundary operator $\mathcal{B}$ corresponds to the different kinds of scatterers: in the case of Dirichlet boundary condition (i.e., $\mathcal{B}u := u$) we call the scatterer acoustically soft, whereas in the case of Neumann boundary condition (i.e., $\mathcal{B}u := \frac{\partial u}{\partial \boldsymbol{n}}$, where $\boldsymbol{n}$ is the unit outer normal vector to Γ) we call the scatterer acoustically hard. The case of the absorbing scatterer (i.e., $\mathcal{B}u := \frac{\partial u}{\partial \boldsymbol{n}} - \frac{\alpha}{c}\frac{\partial u}{\partial t}, \alpha > 0$) is not considered in this work. For results on solvability of the problem and uniqueness of the solution, see [1].

Following the methodology presented in the original work by Bamberger and Ha-Duong [1, 13, 12, 14] we aim to solve the problem (1.21) using the method of retarded potentials. To do this one can either use a direct approach, when the appropriate integral equations stem from the representation formula, or indirect approach, when the solution is expressed using the surface retarded potentials. Although we will mainly focus on the latter one, we start by

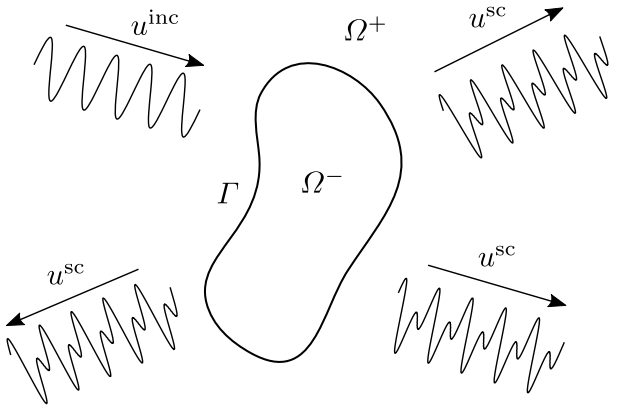


FIGURE 1.1: Wave scattered off a domain Ω^-

recalling the representation formula in order to introduce the surface potential operators for the wave equation.

Let u be the solution of the wave equation in $\mathbb{R}^3 \setminus \Gamma$ satisfying $u = u^{\mathrm{i}}$ in Ω^- and $u = u^{\mathrm{e}}$ in Ω^+. Then for all $(\boldsymbol{x}, t)$ from $(\mathbb{R}^3 \setminus \Gamma) \times \mathbb{R}^+$ the so-called Kirchhoff's formula holds

$$\begin{aligned} u(\boldsymbol{x}, t) = &- \frac{1}{4\pi} \int_\Gamma \frac{\boldsymbol{n}_y(\boldsymbol{x} - \boldsymbol{y})}{\|\boldsymbol{x} - \boldsymbol{y}\|} \left(\frac{\varphi(\boldsymbol{y}, \tau)}{\|\boldsymbol{x} - \boldsymbol{y}\|^2} + \frac{\dot{\varphi}(\boldsymbol{y}, \tau)}{\|\boldsymbol{x} - \boldsymbol{y}\|} \right) \mathrm{d}s_{\boldsymbol{y}} \\ &+ \frac{1}{4\pi} \int_\Gamma \frac{p(\boldsymbol{y}, \tau)}{\|\boldsymbol{x} - \boldsymbol{y}\|} \mathrm{d}s_{\boldsymbol{y}}, \end{aligned} \tag{1.22}$$

where

$$\varphi := u^{\mathrm{i}} - u^{\mathrm{e}}, \qquad p := \frac{\partial u^{\mathrm{i}}}{\partial \boldsymbol{n}} - \frac{\partial u^{\mathrm{e}}}{\partial \boldsymbol{n}}$$

are the jumps of u and $\frac{\partial u}{\partial \boldsymbol{n}}$ defined on $\Gamma \times \mathbb{R}^+$ and

$$\tau := t - \|\boldsymbol{x} - \boldsymbol{y}\|$$

is the retarded time. Finally, we denote $\dot{\varphi} := \frac{\partial \varphi}{\partial t}$ the time derivative of φ.

The integral

$$Lp(\boldsymbol{x}, t) := \frac{1}{4\pi} \int_\Gamma \frac{p(\boldsymbol{y}, \tau)}{\|\boldsymbol{x} - \boldsymbol{y}\|} \mathrm{d}s_{\boldsymbol{y}}, \quad (\boldsymbol{x}, t) \in (\mathbb{R}^3 \setminus \Gamma) \times \mathbb{R}^+ \tag{1.23}$$

occuring in (1.22) is called the single layer retarded potential, whereas the double layer retarded potential is defined for all $(\boldsymbol{x}, t)$ in $(\mathbb{R}^3 \setminus \Gamma) \times \mathbb{R}^+$ by the integral

$$M\varphi(\boldsymbol{x}, t) := \frac{1}{4\pi} \int_\Gamma \frac{\boldsymbol{n}_y(\boldsymbol{x} - \boldsymbol{y})}{\|\boldsymbol{x} - \boldsymbol{y}\|} \left(\frac{\varphi(\boldsymbol{y}, \tau)}{\|\boldsymbol{x} - \boldsymbol{y}\|^2} + \frac{\dot{\varphi}(\boldsymbol{y}, \tau)}{\|\boldsymbol{x} - \boldsymbol{y}\|} \right) \mathrm{d}s_{\boldsymbol{y}}. \tag{1.24}$$

1.2.1 Time domain boundary integral operators

Similarly as in the case of the Laplace equation we define the following well known time domain retarded potential operators for appropriate densities p, φ on $\Gamma \times \mathbb{R}$ in order to take the limits of L, M as $\boldsymbol{x} \to \Gamma$:

- single layer boundary integral operator

$$Sp(\boldsymbol{x}, t) := \frac{1}{4\pi} \int_\Gamma \frac{p(\boldsymbol{y}, \tau)}{\|\boldsymbol{x} - \boldsymbol{y}\|} \mathrm{d}s_{\boldsymbol{y}}$$

- double layer boundary integral operator

$$K_t p(\boldsymbol{x}, t) := \frac{1}{4\pi} \int_\Gamma \frac{\boldsymbol{n}_x(\boldsymbol{x} - \boldsymbol{y})}{\|\boldsymbol{x} - \boldsymbol{y}\|} \left(\frac{p(\boldsymbol{y}, \tau)}{\|\boldsymbol{x} - \boldsymbol{y}\|^2} + \frac{\dot{p}(\boldsymbol{y}, \tau)}{\|\boldsymbol{x} - \boldsymbol{y}\|} \right) \mathrm{d}s_{\boldsymbol{y}}$$

- adjoint double layer boundary integral operator

$$K_t'\varphi(\boldsymbol{x},t) := \frac{1}{4\pi}\int_\Gamma \frac{\boldsymbol{n}_y(\boldsymbol{x}-\boldsymbol{y})}{\|\boldsymbol{x}-\boldsymbol{y}\|}\left(\frac{\varphi(\boldsymbol{y},\tau)}{\|\boldsymbol{x}-\boldsymbol{y}\|^2}+\frac{\dot\varphi(\boldsymbol{y},\tau)}{\|\boldsymbol{x}-\boldsymbol{y}\|}\right)\mathrm{d}s_{\boldsymbol{y}}$$

- hypersingular boundary integral operator

$$D_t\varphi(\boldsymbol{x},t) := \lim_{\Omega\ni\boldsymbol{x}'\to\boldsymbol{x}\in\Gamma}\boldsymbol{n}_x\cdot\nabla_{\boldsymbol{x}'}\left(-\frac{1}{4\pi}\int_\Gamma \boldsymbol{n}_y\cdot\nabla_{\boldsymbol{x}'}\left(\frac{\varphi(\boldsymbol{y},t-\|\boldsymbol{x}'-\boldsymbol{y}\|)}{\|\boldsymbol{x}'-\boldsymbol{y}\|}\right)\mathrm{d}s_{\boldsymbol{y}}\right)$$

The limit in the definition of D_t' is taken in the sense of distributions. For the sake of simplicity we will skip the subscript t when no confusion is caused.

Taking the limits of L, M and their normal derivatives for $\tilde{\boldsymbol{x}}$ approaching Γ we obtain

$$\begin{cases}
\lim\limits_{\Omega^-\ni\tilde{\boldsymbol{x}}\to\boldsymbol{x}\in\Gamma}(Lp)(\tilde{\boldsymbol{x}},t) = \lim\limits_{\Omega^+\ni\tilde{\boldsymbol{x}}\to\boldsymbol{x}\in\Gamma}Lp(\tilde{\boldsymbol{x}},t) = Sp(\boldsymbol{x},t),\\
\lim\limits_{\Omega^-\ni\tilde{\boldsymbol{x}}\to\boldsymbol{x}\in\Gamma}\dfrac{\partial(Lp)(\tilde{\boldsymbol{x}},t)}{\partial\boldsymbol{n}} = (I/2+K)p(\boldsymbol{x},t),\\
\lim\limits_{\Omega^+\ni\tilde{\boldsymbol{x}}\to\boldsymbol{x}\in\Gamma}\dfrac{\partial(Lp)(\tilde{\boldsymbol{x}},t)}{\partial\boldsymbol{n}} = (-I/2+K)p(\boldsymbol{x},t),\\
\lim\limits_{\Omega^-\ni\tilde{\boldsymbol{x}}\to\boldsymbol{x}\in\Gamma}(M\varphi)(\tilde{\boldsymbol{x}},t) = (-I/2+K')\varphi(\boldsymbol{x},t),\\
\lim\limits_{\Omega^+\ni\tilde{\boldsymbol{x}}\to\boldsymbol{x}\in\Gamma}(M\varphi)(\tilde{\boldsymbol{x}},t) = (I/2+K')\varphi(\boldsymbol{x},t),\\
\lim\limits_{\Omega^-\ni\tilde{\boldsymbol{x}}\to\boldsymbol{x}\in\Gamma}\dfrac{\partial(M\varphi)(\tilde{\boldsymbol{x}},t)}{\partial\boldsymbol{n}} = \lim\limits_{\Omega^+\ni\tilde{\boldsymbol{x}}\to\boldsymbol{x}\in\Gamma}\dfrac{\partial(M\varphi)(\tilde{\boldsymbol{x}},t)}{\partial\boldsymbol{n}} = D\varphi(\boldsymbol{x},t).
\end{cases} \tag{1.25}$$

In the following subsections we apply these formulas to derive boundary integral equations suitable for the solution of (1.21) with Dirichlet and Neumann boundary conditions.

1.2.2 Function spaces

Before providing the boundary integral equations let us define suitable Sobolev spaces in which we will search for the solution. Let E be a Hilbert space and $\mathcal{D}_+'(E)$, $\mathcal{S}_+'(E)$ the sets of distributions and tempered distributions, respectively, on $\mathbb{R}$ with the values in E and support in $[0,\infty)$. We define

$$LT(\sigma,E) := \{f\in\mathcal{D}_+'(E) : \mathrm{e}^{-\sigma t}f\in\mathcal{S}_+'(E)\}$$

and the set of Laplace transformable distributions

$$LT(E) := \bigcup_{\sigma\in\mathbb{R}} LT(\sigma,E).$$

A Fourier-Laplace transform of $f \in LT(E)$ is defined as

$$\hat{f}(\omega) := \mathcal{F}(e^{-\sigma t} f)(\eta), \tag{1.26}$$

where $\omega := \eta + \mathrm{i}\sigma$ and $(\mathcal{F}(u(t)))(\eta) := \int_{-\infty}^{\infty} \mathrm{e}^{\mathrm{i}\eta t} u(t)\,\mathrm{d}t$ is the Fourier transform of a given function $u(t)$.

Let us define the following Hilbert space [13]

$$H^s_\sigma(\mathbb{R}^+, E) := \left\{ f \in LT(\sigma, E) : \mathrm{e}^{-\sigma t} \Lambda^s f \in L^2(\mathbb{R}, E) \right\},$$

where $s \in \mathbb{R}$, $\sigma > 0$, and $\widehat{\Lambda^s f} := (-\mathrm{i}\omega)^s \hat{f}(\omega)$.

In what follows we will also make use of the following space

$$H^{1,1}_\sigma(\Omega) := \left\{ u \in LT(\sigma, H^1(\Omega)) : \int_{-\infty+\mathrm{i}\sigma}^{+\infty+\mathrm{i}\sigma} \|\hat{u}\|_{1,\omega,\Omega}\,\mathrm{d}\omega < +\infty \right\},$$

where $\|f\|_{1,\omega,\Omega} := \int_\Omega (|\nabla u(\boldsymbol{x})|^2 + |\omega f(\boldsymbol{x})|^2)\,\mathrm{d}\boldsymbol{x}$. Let us note that using the Parceval theorem one obtains $u \in H^{1,1}_\sigma(\Omega)$ if and only if $\mathrm{e}^{-2\sigma t}\nabla u \in L^2(\mathbb{R}, L^2(\Omega))$ and $\mathrm{e}^{-2\sigma t} u \in L^2(\mathbb{R}, L^2(\Omega))$ (see [12]). By $L(\mathbb{R}, X)$, where X is a Hilbert space, we mean the space of strongly measurable functions f with values in $L^2(\Omega)$ such that (see [18])

$$\|f\|_{L^2(\mathbb{R},X)} := \left(\int_{\mathbb{R}} \|f(t)\|_X\,\mathrm{d}t \right)^{1/2} < +\infty. \tag{1.27}$$

Space $L^2(\mathbb{R}, X)$ with this norm is a Hilbert space. Using this one can define

$$H^m(\mathbb{R}, X) := \left\{ f : f, f^{(1)} := \frac{\partial f}{\partial t}, \ldots, f^{(m)} := \frac{\partial^m f}{\partial t^m} \in L^2(\mathbb{R}, X) \right\},$$

which is a Hilbert space equipped with the scalar product $\sum_{j=0}^m \int_{\mathbb{R}} (f^{(j)}(t), g^{(j)}(t))_X\,\mathrm{d}t$. Spaces $L^2([a,b], X)$ and $H^m([a,b], X)$ can be defined in a similar manner [18].

To define the trace spaces we consider a finite covering of Γ by open sets $(\mathcal{O}_i)_{i=1}^I$, a smooth partition of unity $(\alpha_i)_{i=1}^I$ subordinate to this covering, and diffeomorphisms $(\varphi_i)_{i=1}^I$ mapping $\mathcal{O}_i$ to $Q := \{(x_1, x_2, x_3) : -1 < x_j < 1\}$, $\mathcal{O}_i \cap \Omega$ into $Q^+ := \{x \in Q : x_3 > 0\}$, and $\mathcal{O}_i \cap \Gamma$ into $\Sigma := \{x \in Q : x_3 = 0\}$. We define

$$|f|_{1/2,\omega,\Gamma} := \left(\sum_{i=1}^I \int_{\mathbb{R}^2} (|\omega|^2 + |\xi|^2)^{1/2} |\widehat{\phi_i f}(\xi)|^2\,\mathrm{d}\xi \right),$$

with $(\phi_i f)(x') := (\alpha_i f) \circ \varphi_i^{-1}(x', 0)$ defined for $x' \in \Sigma$.

This allows us to define the spaces

$$H_\sigma^{1/2,1/2}(\Gamma) := \left\{ u \in LT(\sigma, H^{1/2}(\Gamma)) : \int_{-\infty+\mathrm{i}\sigma}^{+\infty+\mathrm{i}\sigma} |\hat{u}|^2_{1/2,\omega,\Gamma} \,\mathrm{d}\omega < +\infty \right\}$$

and

$$H_\sigma^{-1/2,-1/2}(\Gamma) := \left\{ u \in LT(\sigma, H^{-1/2}(\Gamma)) : \int_{-\infty+\mathrm{i}\sigma}^{+\infty+\mathrm{i}\sigma} |\hat{u}|^2_{-1/2,\omega,\Gamma} \,\mathrm{d}\omega < +\infty \right\},$$

where $|f|_{-1/2,\omega,\Gamma}$ is the dual norm to $|f|_{1/2,\omega,\Gamma}$, i.e.,

$$|f|_{-1/2,\omega,\Gamma} := \sup_{0 \neq u \in H_\sigma^{1/2,1/2}(\Gamma)} \frac{|f(u)|}{|u|_{1/2,\omega,\Gamma}}.$$

Again, from Parceval theorem we get that $u \in H_\sigma^{1/2,1/2}(\Gamma)$ if and only if

$$\mathrm{e}^{-\sigma t} u \in L^2(\mathbb{R}, H^{1/2}(\Gamma)) \cap H^{1/2}(\mathbb{R}, L^2(\Gamma)),$$

where

$$H^{1/2}(\mathbb{R}, L^2(\Gamma)) := \{ f : (1 + |\eta|^2)^{1/2} \hat{f} \in L^2(\mathbb{R}, L^2(\Gamma)) \}.$$

Finally we denote

$$H_\sigma^{k,1/2,1/2}(\Gamma) := \left\{ f : \frac{\partial^k f}{\partial t^k} \in H_\sigma^{1/2,1/2}(\Gamma) \right\}$$

the Hilbert space equipped with the norm

$$\|u\|_{H_\sigma^{k,1/2,1/2}(\Gamma)} := \int_{-\infty+\mathrm{i}\sigma}^{+\infty+\mathrm{i}\sigma} |\omega|^k |\hat{u}|^2_{1/2,\omega,\Gamma} \,\mathrm{d}\omega.$$

1.2.3 Dirichlet boundary value problem and its variational formulation

A complete overview of boundary integral equations suitable for the solution of (1.21) with $\mathcal{B}u := u$ together with the analysis of the solvability is available, e.g., in [7, 12]. In what follows we focus on the first kind retarded potential boundary integral formulation using the representation by the single layer potential

$$u(\boldsymbol{x}, t) = Lp(\boldsymbol{x}, t) = \frac{1}{4\pi} \int_\Gamma \frac{p(\boldsymbol{y}, t - \|\boldsymbol{x} - \boldsymbol{y}\|)}{\|\boldsymbol{x} - \boldsymbol{y}\|} \,\mathrm{d}s_{\boldsymbol{y}}, \quad t > 0, \boldsymbol{x} \notin \Gamma.$$

The unknown density p representing the jump of normal derivatives of u passing Γ is obtained by solving the retarded potential boundary integral equation

$$\begin{cases} \text{Find } p \in H_{\sigma}^{-1/2,-1/2}(\Gamma), \text{ such that} \\ (Sp)(\boldsymbol{x},t) = g(\boldsymbol{x},t) \quad \text{for } (\boldsymbol{x},t) \in \Gamma \times \mathbb{R}^+. \end{cases} \tag{1.28}$$

As it is usual in the case of the retarded potential boundary integral equations, the theoretical results concerning the uniqueness and solvability are obtained by transforming the problem into the frequency domain by the Laplace-Fourier transform. The results are then transferred back to the time-domain using the inverse transform and the Paley-Wiener theorem [12, 7].

Theorem 1.1 (Paley-Wiener)**.** *The following statements are equivalent.*

(i) $h(\omega) = \hat{f}(\omega)$ *for* $f \in LT(E)$

(ii) h *is holomorphic in a semi-plane* $\{\operatorname{Im}\omega > \omega_0^I\}$ *and*

$$\exists \sigma' > \omega_0^I, C > 0, k \in \mathbb{N} \text{ s.t. } \forall \omega \in \{\operatorname{Im}\omega \geq \sigma'\}: \quad \|h(\omega)\|_E \leq C(1+|\omega|)^k$$

Applying the Laplace-Fourier transform to the original Dirichlet problem we obtain the Helmholtz equation in a frequency domain

$$\begin{cases} \text{Find } \hat{u}(\omega,\boldsymbol{x}) \in H^1(\Omega^+), \text{ such that} \\ \triangle \hat{u}(\omega,\boldsymbol{x}) + \omega^2 \hat{u}(\omega,\boldsymbol{x}) = 0 \quad \text{for } \boldsymbol{x} \in \Omega^+, \\ \hat{u}(\omega,\boldsymbol{x}) = \hat{g}(\omega,\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \Gamma. \end{cases}$$

Representing the solution using the single layer potential

$$\hat{u}(\omega,\boldsymbol{x}) = (S_\omega p)(\boldsymbol{x}) := \int_\Gamma \frac{\mathrm{e}^{\mathrm{i}\omega\|\boldsymbol{x}-\boldsymbol{y}\|}}{4\pi\|\boldsymbol{x}-\boldsymbol{y}\|} p(\boldsymbol{y})\,\mathrm{d}s_{\boldsymbol{y}}, \quad \boldsymbol{x} \notin \Gamma,$$

the problem in the frequency domain can be solved using the first kind boundary integral equation

$$(S_\omega p)(\boldsymbol{x}) = \hat{g}(\omega,\boldsymbol{x}), \quad \boldsymbol{x} \in \Gamma, \tag{1.29}$$

which is the Fourier-Laplace transform of the equation in (1.28). Multiplying (1.29) by $-\mathrm{i}\omega q$, $q \in H^{-1/2}(\Gamma)$, one obtains the variational problem

$$\begin{cases} \text{Find } p \in H^{-1/2}(\Gamma), \text{ such that} \\ a_\omega(p,q) := \langle -\mathrm{i}\omega S_\omega p, q\rangle = \langle -\mathrm{i}\omega f, q\rangle \quad \forall q \in H^{-1/2}(\Gamma). \end{cases} \tag{1.30}$$

As it is shown in [12] and [1] the sesquilinear form a_ω is bounded and coercive and the problem (1.30) admits a unique solution. Using this in combination with the Paley-Wiener theorem one can derive the following result in the time domain.

Theorem 1.2. *If* $g \in H_\sigma^{1,1/2,1/2}(\Gamma) := \{f : \dot{f} \in H_\sigma^{1/2,1/2}(\Gamma)\}$ *then the equation* (1.28) *has a unique solution* p *satisfying*

$$\|p\|_{H_\sigma^{-1/2,-1/2}(\Gamma)} \leq C\|\dot{g}\|_{H_\sigma^{1/2,1/2}(\Gamma)}.$$

The variational formulation in time domains stems from the problem (1.30). Note that from the definition (1.26) one can see that $-\mathrm{i}\omega S_\omega \hat{p}$ is the transformation of $S\dot{p}$. The variational formulation reads

$$\begin{cases} \text{Find } \dot{p} \in H^{-1/2,-1/2}(\Gamma), \text{ such that} \\ \int_0^\infty \mathrm{e}^{-2\sigma t} \int_\Gamma q(\boldsymbol{x},t) S\dot{p}(\boldsymbol{x},t)\,\mathrm{d}s_{\boldsymbol{x}}\,\mathrm{d}t = \int_0^\infty \mathrm{e}^{-2\sigma t} \int_\Gamma \dot{g}(\boldsymbol{x},t) q(\boldsymbol{x},t)\,\mathrm{d}s_{\boldsymbol{x}}\mathrm{d}t \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \forall q \in H^{-1/2,-1/2}(\Gamma). \end{cases}$$

1.2.4 Neumann boundary value problem and its variational formulation

Focusing now on the Neumann problem we set $\mathcal{B}u := \frac{\partial u}{\partial \boldsymbol{n}}$ in (1.21) and express the solution using the double layer potential

$$\begin{aligned} u(\boldsymbol{x},t) =& \frac{1}{4\pi} \int_\Gamma \boldsymbol{n}_{\boldsymbol{y}} \cdot \nabla_{\boldsymbol{x}} \left(\frac{\varphi(t - \|\boldsymbol{x}-\boldsymbol{y}\|, \boldsymbol{y})}{\|\boldsymbol{x}-\boldsymbol{y}\|} \right) \mathrm{d}s_{\boldsymbol{y}} \\ =& \frac{1}{4\pi} \int_\Gamma \frac{\boldsymbol{n}_{\boldsymbol{y}} \cdot (\boldsymbol{y}-\boldsymbol{x})}{\|\boldsymbol{x}-\boldsymbol{y}\|^2} \left(\frac{\partial \varphi}{\partial t}(t - \|\boldsymbol{x}-\boldsymbol{y}\|, \boldsymbol{y}) + \frac{\varphi(t - \|\boldsymbol{x}-\boldsymbol{y}\|, \boldsymbol{y})}{\|\boldsymbol{x}-\boldsymbol{y}\|} \right) \mathrm{d}s_{\boldsymbol{y}}. \end{aligned}$$

Recall that φ represents the jump of u across boundary Γ. The unknown φ can be obtained by solving the boundary integral equation

$$D\varphi = f,$$

where D is the hypersingular boundary integral operator defined in Section 1.2.1. It is known [13] that D is linear and continuous operator from $H_\sigma^2(\mathbb{R}^+, H^{1/2}(\Gamma)$ to the space $H_\sigma^0(\mathbb{R}^+, H^{-1/2}(\Gamma))$. The operator satisfies the coercivity property

$$\int_{-\infty}^{+\infty} \mathrm{e}^{-2\sigma t} \langle D\varphi(\cdot,t), \dot{\varphi}(\cdot,t) \rangle \,\mathrm{d}t \geq C|\varphi|^2_{\sigma,0,1/2} \quad \forall \varphi \in H_\sigma^2(\mathbb{R}^+, H^{1/2}(\Gamma)), \sigma > \sigma_0 > 0,$$

where C only depends on Γ and σ_0. This motivates the variational formulation

$$\begin{cases} \text{Find } \varphi \in H^2_\sigma(\mathbb{R}^+, H^{1/2}(\Gamma)), \text{ such that} \\ \int_{-\infty}^{+\infty} \mathrm{e}^{-2\sigma t} \langle D\varphi(\cdot,t), \dot{\psi}(\cdot,t)\rangle \,\mathrm{d}t = \int_{-\infty}^{+\infty} \mathrm{e}^{-2\sigma t} \langle f(\cdot,t), \dot{\psi}(\cdot,t)\rangle \,\mathrm{d}t \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \forall \psi \in H^2_\sigma(\mathbb{R}^+, H^{1/2}(\Gamma)). \end{cases} \tag{1.31}$$

Here

$$\begin{aligned} \langle D\varphi(\cdot,t), \psi(\cdot,t)\rangle :=& \frac{1}{4\pi} \int_\Gamma \int_\Gamma \frac{\boldsymbol{n}_{\boldsymbol{x}} \cdot \boldsymbol{n}_{\boldsymbol{y}}}{\|\boldsymbol{x}-\boldsymbol{y}\|} \ddot{\varphi}(\boldsymbol{y}, t-\|\boldsymbol{x}-\boldsymbol{y}\|)\psi(\boldsymbol{x},t) \\ &+ \frac{\operatorname{curl}_\Gamma \varphi(\boldsymbol{y}, t-\|\boldsymbol{x}-\boldsymbol{y}\|) \operatorname{curl}_\Gamma \psi(\boldsymbol{x},t)}{\|\boldsymbol{x}-\boldsymbol{y}\|} \,\mathrm{d}s_{\boldsymbol{x}} \mathrm{d}s_{\boldsymbol{y}} \end{aligned}$$

comes out from treating the hypersingularity in the operator D using the theory of distributions.

Chapter 2

Boundary Element Method

In the following part we discuss the discretization of the variational formulations presented in the previous chapter using the Galerkin boundary element method. Since the discretization of the stationary problems is widely described in numerous literature (see, e.g., [26, 32, 24]) we do not intend to go into comprehensive details on this subjects. Rather we will focus on the discretization of the space-time variational formulation for the wave equation which is less known. We explain the problems with numerical integration and present the method to overcome them using smooth temporal basis function firstly described by Sauter and Veit [27, 29, 28, 30]. In the Part II the paper focusing on the parallelization of the approach is presented, which is one of the main results of this thesis [34].

2.1 BEM for the Laplace equation

Let us assume that the boundary Γ of a Lipschitz domain Ω is piece-wise polyhedral and consider its triangulation $\tau = \{\tau_i\}_{i=1}^N$

$$\Gamma = \bigcup_{l=1}^{N} \overline{\tau}_l$$

consisting of N triangular elements and M nodes $\{\boldsymbol{x}_i\}_{i=1}^M$. We suppose that the triangulation is admissible, i.e., an intersection of two elements τ_i, τ_j, $i \neq j$ is an empty set, a node, or an edge.

In order to provide finite dimensional counterparts to the function spaces presented in the previous chapter, we define the piece-wise constant shape functions

$$\psi_l(\boldsymbol{x}) := \begin{cases} 1 & \text{for } \boldsymbol{x} \in \tau_l, \\ 0 & \text{elsewhere,} \end{cases}$$

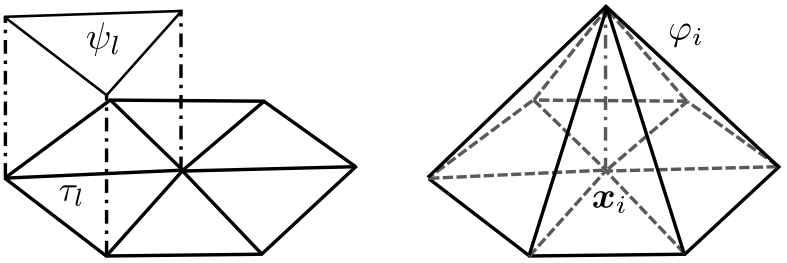


FIGURE 2.1: Piece-wise constant and continuous piece-wise linear shape functions

for $l \in \{1, 2, \ldots, N\}$ and the continuous piece-wise linear shape functions $\{\varphi_i\}_{i=1}^{M}$ such that

$$\varphi_i(\boldsymbol{x}) := \begin{cases} 1 & \text{for } \boldsymbol{x} = \boldsymbol{x}_i, \\ 0 & \text{for } \boldsymbol{x} = \boldsymbol{x}_j \neq \boldsymbol{x}_i \end{cases} \tag{2.1}$$

(see Figure 2.1). These sets of shape functions form the bases of the finite dimensional spaces

$$S_h^N(\Gamma) := \operatorname{span}\{\psi_l\}_{l=1}^{N} \subset H^{-1/2}(\Gamma) \tag{2.2}$$

and

$$S_h^M(\Gamma) := \operatorname{span}\{\varphi_i\}_{i}^{M} \subset H^{1/2}(\Gamma),$$

therefore any $u_h \in S_h^N(\Gamma)$ can be written in the form

$$u_h = \sum_{l=1}^{N} u_i \psi_i, \quad u_i \in \mathbb{R},$$

and, naturally, for any $v_h \in S_h^M$ we can write

$$v_h = \sum_{i=1}^{M} v_i \varphi_i, \quad v_i \in \mathbb{R}.$$

These expressions can be identified with the vectors of coefficients $\boldsymbol{u} := (u_1, u_2, \ldots, u_N)^\top \in \mathbb{R}^N$ and $\boldsymbol{v} := (v_1, v_2, \ldots, v_M)^\top \in \mathbb{R}^M$, respectively.

2.1.1 Galerkin discretization of the boundary integral equations

As a model problem we will at first search for the approximate solution of the interior boundary value problem for the Laplace equation using the variational formulation (1.14). We set

$$t \approx t_h := \sum_{i=1}^{N} t_i \psi_i \in S_h^N(\Gamma),$$

where $t_i \in \mathbb{R}$ for all $i \in \{1, 2, \ldots, N\}$, and plug it into the equation (1.14) to obtain the Galerkin approximation

$$\begin{cases} \text{Find } \boldsymbol{t}_h := (t_1, t_2, \ldots, t_N)^\top \in \mathbb{R}^N \text{ such that} \\ \sum_{i=1}^{N} t_i \langle V\psi_i, \psi_j \rangle_\Gamma = \sum_{i=1}^{M} g_i \langle (I/2 + K)\varphi_i, \psi_j \rangle_\Gamma \quad \text{for all } j \in \{1, 2, \ldots, N\}. \end{cases}$$

The aproximation $g_h = \sum_{i=1}^{M} g_i \varphi_i$ of the Dirichlet data, i.e., the vector of coefficients $\boldsymbol{g_h} := (g_1, g_2, \ldots, g_M) \in \mathbb{R}^M$, can either be obtained by an interpolation or by L^2 projection (see [24, 32]). Defining the matrices

$$V_h[i,j] := \frac{1}{4\pi} \int_{\tau_i} \int_{\tau_j} \frac{1}{\|\boldsymbol{x} - \boldsymbol{y}\|} \, \mathrm{d}s_{\boldsymbol{y}} \, \mathrm{d}s_{\boldsymbol{x}}, \quad i, j \in \{1, 2, \ldots, N\}, \tag{2.3}$$

$$K_h[k,l] := \frac{1}{4\pi} \int_{\tau_k} \int_{\Gamma} \frac{(\boldsymbol{x} - \boldsymbol{y}, \boldsymbol{n_y})}{\|\boldsymbol{x} - \boldsymbol{y}\|^3} \varphi_l(\boldsymbol{y}) \, \mathrm{d}s_{\boldsymbol{y}} \, \mathrm{d}s_{\boldsymbol{x}}, \quad k \in \{1, 2, \ldots, N\}, l \in \{1, 2, \ldots, M\}, \tag{2.4}$$

and the identity matrix

$$M_h[k,l] := \int_{\tau_k} \varphi_l(\boldsymbol{y}) \, \mathrm{d}s_{\boldsymbol{y}}, \quad k \in \{1, 2, \ldots, N\}, l \in \{1, 2, \ldots, M\},$$

we can write the discrete problem as

$$\begin{cases} \text{Find } \boldsymbol{t}_h \in \mathbb{R}^N \text{ such that} \\ V_h \boldsymbol{t}_h = (\frac{1}{2} M_h + K_h) \boldsymbol{g}_h. \end{cases}$$

In this case the system matrix V_h is symmetric and positive definite [24], therefore the conjugate gradient algorithm with appropriate preconditioner can be used to solve the system.

To approximate the solution of the interior Neumann problem for the Laplace equation we start from the regularized formulation (1.20). The unknown function $\bar{u}$ is approximated using the piece-wise linear basis functions

$$\bar{u} \approx \bar{u}_h := \sum_{i=1}^{M} \bar{u}_i \varphi_i \in S_h^M(\Gamma),$$

with $\bar{u}_i \in \mathbb{R}$ for $i \in \{1, 2, \dots, M\}$. Inserting this into the equation (1.20) we obtain

$$\begin{cases} \text{Find } \bar{\boldsymbol{u}}_h := (\bar{u}_1, \bar{u}_2, \dots \bar{u}_N)^\top \in \mathbb{R}^M \text{ such that} \\ \sum_{i=1}^M \bar{u}_i \left(\langle D\varphi_i, \varphi_j \rangle_\Gamma + \langle \varphi_i, 1 \rangle_\Gamma \langle \varphi_j, 1 \rangle_\Gamma \right) = \langle (I/2 - K')g, \varphi_j \rangle_\Gamma + \alpha \langle \varphi_j, 1 \rangle \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{for all } j \in \{1, 2, \dots, M\}. \end{cases}$$

In order to introduce a corresponding matrix formulation we set

$$D_h[i,j] := \frac{1}{4\pi} \int_\Gamma \int_\Gamma \frac{(\operatorname{curl}_\Gamma \varphi_j(\boldsymbol{y}) \operatorname{curl}_\Gamma \varphi_i(\boldsymbol{x}))}{\|\boldsymbol{x} - \boldsymbol{y}\|} \, \mathrm{d}s_{\boldsymbol{x}} \, \mathrm{d}s_{\boldsymbol{y}}, \tag{2.5}$$

where the surface $\operatorname{curl}_\Gamma$ operator is defined as

$$\operatorname{curl}_\Gamma \varphi_i(\boldsymbol{x}) := \boldsymbol{n}_{\boldsymbol{x}} \times \nabla_{\boldsymbol{x}} \tilde{\varphi}_i(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \Gamma.$$

Here $\tilde{\varphi}_i$ is a locally defined extension of φ_i by a constant along $\boldsymbol{n}_{\boldsymbol{x}}$. This allows us to write

$$D_h[i,j] = \sum_{\tau_k \in \operatorname{supp} \varphi_i} \sum_{\tau_l \in \operatorname{supp} \varphi_j} \left(\operatorname{curl}_\Gamma \varphi_{i|\tau_k}, \operatorname{curl}_\Gamma \varphi_{j|\tau_l} \right) \frac{1}{4\pi} \int_{\tau_k} \int_{\tau_l} \frac{1}{\|\boldsymbol{x} - \boldsymbol{y}\|} \, \mathrm{d}s_{\boldsymbol{x}} \, \mathrm{d}s_{\boldsymbol{y}}.$$

Moreover, we present the vectors $\boldsymbol{a} := (a_1, a_2, \dots, a_M)^\top \in \mathbb{R}^M$ and $\boldsymbol{f} := (f_1, f_2, \dots, f_M)^\top \in \mathbb{R}^M$ with entries given by

$$a_i := \int_\Gamma \varphi_i(\boldsymbol{x}) \, \mathrm{d}s_{\boldsymbol{x}},$$

and

$$f_i := \frac{1}{2} \int_\Gamma g(\boldsymbol{x}) \varphi_(\boldsymbol{x}) \, \mathrm{d}s_{\boldsymbol{x}} - \frac{1}{4\pi} \int_\Gamma \varphi_i(\boldsymbol{x}) \int_\Gamma \frac{(\boldsymbol{x} - \boldsymbol{y}, \boldsymbol{n}_{\boldsymbol{x}})}{\|\boldsymbol{x} - \boldsymbol{y}\|^3} g(\boldsymbol{y}) \, \mathrm{d}s_{\boldsymbol{y}} \, \mathrm{d}s_{\boldsymbol{x}},$$

respectively. The matrix formulation is then given by

$$\begin{cases} \text{Find } \bar{\boldsymbol{u}}_h \in \mathbb{R}^M \text{ such that} \\ (D + \boldsymbol{a}\boldsymbol{a}^\top) \bar{\boldsymbol{u}}_h = \boldsymbol{f} + \alpha \boldsymbol{a}. \end{cases}$$

Again, the system matrix $D + \boldsymbol{a}\boldsymbol{a}^\top$ is symmetric and positive definite and the (preconditioned) conjugate gradient method can be used to iteratively solve the problem.

2.1.2 Numerical evaluation

When assembling the system matrices presented in the previous subsection one has to overcome several problems. Firstly, since the kernels inside the integrals are non-local the resulting system matrices are full. Therefore the classical BEM is of the quadratic complexity both in

memory and CPU time with respect to the number of surface elements. Memory requirements as well as computational time limits us to problems with relatively small number of surface elements. However, this can be treated using a fast BEM method in combination with parallelization as will be described in the next chapter. Another problem is caused by the singularity occuring inside the integrals in (2.3), (2.4), and (2.5). To treat this a more sophisticated quadrature method is needed than for, e.g., the finite element method (at least for pairs of elements close to each other).

In our work we use two methods for treating the problems with singularities. The first of them is the so-called semi-analytic approach described by Steinbach and Rjasanow in their monograph [24]. In this case the inner integrals containing the singularities are treated analytically while the outer integrals are evalueated using a numerical quadrature. The semi-analytical schemes for the Laplace and Lamé equations are presented in [24], the Helmholtz equation is treated in [6].

Another approach presented by Sauter and Schwab [26] introduces a suitable transformation of variables that renders the singular integrals analytic. This results in four one-dimensional integrals that can be evaluated numerically using Gaussian quadrature on line. Four different cases have to be taken into account as the transformation differs based on the mutual positions of elements – we distinguish among identical elements, elements sharing an edge, elements sharing a vertex, and disjoint elements.

Moreover, for well separated elements, i.e., those satisfying the condition

$$\|\boldsymbol{t}_i - \boldsymbol{t}_j\| > \eta \max\{\text{diam } \tau_i, \text{diam } \tau_j\}, \tag{2.6}$$

where $\boldsymbol{t}_i$, $\boldsymbol{t}_j$ are centers of gravity of given elements, $\eta \in (0, 1)$, the numerical integration can be carried out using Gaussian quadrature over pairs of triangles. This holds for both semi-analytical and fully numerical approach.

It is clear that the numerical evaluation of the integrals is an expensive process. Therefore, in Part II we present a method for its acceleration using the SIMD instructions of modern processors [21].

2.2 BEM for the wave equation

In what follows we will focus on the spatial and temporal discretization of the Neumann initial boundary value problem for the wave equation (1.21). Discretization of the Dirichlet problem follows the similar procedure. We will look for a solution in the time interval $[0, T]$ with

$0 < T < +\infty$. We start from the variational formulation (1.31)

$$\begin{cases} \text{Find } \varphi \in H^2_\sigma(\mathbb{R}^+, H^{1/2}(\Gamma)), \text{ such that} \\ \int_{\mathbb{R}} \mathrm{e}^{-2\sigma t}\langle D\varphi(\cdot,t), \dot{\psi}(\cdot,t)\rangle \,\mathrm{d}t = \int_{\mathbb{R}} \mathrm{e}^{-2\sigma t}\langle f(\cdot,t), \dot{\psi}(\cdot,t)\rangle \,\mathrm{d}t \\ \qquad\qquad \forall \psi \in H^2_\sigma(\mathbb{R}^+, H^{1/2}(\Gamma)), \end{cases} \tag{2.7}$$

however, later on we will simplify this by setting $\sigma = 0$ and considering only a bounded time interval as it is usual in practical computation. Before proceeding let us recall

$$\begin{aligned} \langle D\varphi(\cdot,t), \psi(\cdot,t)\rangle :=& \frac{1}{4\pi}\int_\Gamma\int_\Gamma \frac{\boldsymbol{n_x}\cdot\boldsymbol{n_y}}{\|\boldsymbol{x}-\boldsymbol{y}\|}\ddot{\varphi}(\boldsymbol{y}, t-\|\boldsymbol{x}-\boldsymbol{y}\|)\psi(\boldsymbol{x},t) \\ &+ \frac{\operatorname{curl}_\Gamma \varphi(\boldsymbol{y}, t-\|\boldsymbol{x}-\boldsymbol{y}\|)\operatorname{curl}_\Gamma \psi(\boldsymbol{x},t)}{\|\boldsymbol{x}-\boldsymbol{y}\|}\,\mathrm{d}s_{\boldsymbol{x}}\mathrm{d}s_{\boldsymbol{y}}. \end{aligned}$$

2.2.1 Spatial discretization

For simplicity we assume that the boundary Γ is polyhedral. Similarly as in the case of the stationary problem we discretize the boundary into an admissible triangulation composed of N triangular elements $\{\tau_i\}_{i=1}^N$ and M nodes $\{\boldsymbol{x}_i\}_{i=1}^M$. To approximate the space $H^{1/2}(\Gamma)$ we will use the space $S_h^M(\Gamma)$ of continuous piece-wise linear functions with basis formed by functions $\{\varphi_i\}_{i=1}^M$ satisfying (2.1) (see Figure 2.1 right). The unknown function φ is therefore approximated by

$$\varphi(\boldsymbol{x},t) \approx \varphi_h(\boldsymbol{x},t) := \sum_{i=1}^M \alpha_i(t)\varphi_i(\boldsymbol{x}),$$

where $\alpha_i \in H^2_\sigma(\mathbb{R}^+, \mathbb{R})$, $i \in \{1, 2, \ldots, M\}$. Inserting this into the variational formulation (2.7) leads to the semi-discrete problem

$$\begin{cases} \text{Find } \alpha_i \in H^2_\sigma(\mathbb{R}^+, \mathbb{R}), \text{ such that} \\ \sum_{i=1}^M \int_{\mathbb{R}} \mathrm{e}^{-2\sigma t}\langle D(\alpha_i(t)\varphi_i(\boldsymbol{x})), \dot{\psi}_h(\cdot,t)\rangle \,\mathrm{d}t = \int_{\mathbb{R}} \mathrm{e}^{-2\sigma t}\langle f(\cdot,t), \dot{\psi}_h(\cdot,t)\rangle \,\mathrm{d}t \\ \qquad\qquad \forall \psi_h \in H^2_\sigma(\mathbb{R}^+, S_h^M(\Gamma)). \end{cases} \tag{2.8}$$

In the next section we present a full discretization of the problem by approximating α_i using a suitable finite element space.

2.2.2 Time discretization

The approach often seen in the literature on the subject (see [7, 14, 12]) is to provide a discrete variational formulation employing temporal basis function of order $m \geq 2$ and appropriate error estimates, then simplifying the numerical realisation by only using the temporal basis

functions of order $m = 1$. Similarly here, we will firstly recall results concerning the approximation by piece-wise quadratic elements, then discuss difficulties arising during the numerical integration and their treatment using smooth temporal basis functions presented in [28].

Since $H^2_\sigma(\mathbb{R}^+, \mathbb{R})$ is the space of functions $\alpha \in LT(\sigma, \mathbb{R})$ such that $\mathrm{e}^{-\sigma t}\partial^2\alpha/\partial t^2 \in L^2(\mathbb{R}^+, \mathbb{R})$, the approximation of the space should consist of piece-wise polynomial functions of order $m \geq 2$ on each subinterval $I_n := [n\Delta t, (n+1)\Delta t]$.

Let us at first consider the case when $m = 2$. Then

$$\varphi_h(\boldsymbol{x}, t) = \sum_{i=1}^{M} \alpha_{i,\Delta t}\varphi(\boldsymbol{x}),$$

where $\alpha_{i,\Delta t|I_i} \in P_2$ for all $i \in \{1, 2, \ldots, M\}$. Since the function is approximated by piece-wise quadratic polynomial, its second derivative is a constant a_i^n on each subinterval I_n. From this and the fact that $\varphi_h(\cdot, 0) = \frac{\partial\varphi_h}{\partial t}(\cdot, 0) = 0$ one can evaluate $\alpha_{i,\Delta t}$ as

$$\alpha_{i,\Delta t} = \frac{1}{2}(t - t_n)^2 a_i^n + \Delta t \sum_{j=0}^{n-1} \left(t - t_j + \frac{1}{2}\Delta t\right) a_i^n.$$

We insert this into the variational formulation

$$\begin{cases} \text{Find } \alpha_{l,\Delta t} \in H^2_\sigma(\Delta t, \mathbb{R}), l = 1, 2, \ldots, M, \text{ such that} \\ \sum_{i=1}^{M} \int_{\mathbb{R}} \mathrm{e}^{-2\sigma t}\beta'_{j,\Delta t}(t) \int_\Gamma \int_\Gamma \frac{\boldsymbol{n_x}\cdot\boldsymbol{n_y}}{4\pi\|\boldsymbol{x}-\boldsymbol{y}\|}\varphi_l(\boldsymbol{y})\varphi_j(\boldsymbol{x})\alpha''_{l,\Delta t}(t - \|\boldsymbol{x} - \boldsymbol{y}\|) \\ + \frac{\mathrm{curl}_\Gamma\,\varphi_l(\boldsymbol{y})\,\mathrm{curl}_\Gamma\,\varphi_j(\boldsymbol{x})}{4\pi\|\boldsymbol{x}-\boldsymbol{y}\|}\alpha_{l,\Delta t}(t - \|\boldsymbol{x} - \boldsymbol{y}\|)\,\mathrm{d}s_{\boldsymbol{x}}\,\mathrm{d}s_{\boldsymbol{y}}\,\mathrm{d}t \\ = \int_{\mathbb{R}} \mathrm{e}^{-2\sigma t}\beta'_{j,\Delta t}(t) \int_\Gamma f_{h,\Delta t}(\boldsymbol{x}, t)\varphi_j(\boldsymbol{x})\,\mathrm{d}s_{\boldsymbol{x}}\,\mathrm{d}t \\ \qquad\qquad \forall\beta_{j,\Delta t} \in H^2_\sigma(\Delta t, \mathbb{R}), \end{cases} \tag{2.9}$$

where $H^2_\sigma(\Delta t, \mathbb{R})$ is the subspace of $H^2_\sigma(\mathbb{R}^+, \mathbb{R})$ and the testing functions are chosen in such a way that

$$\beta'_{i,\Delta t}(t) = (\beta^n_{i,\Delta t})(t) = \begin{cases} t - t_{n-1}, & t_{n-1} < t < t_n, \\ t_{n+1} - t, & t_n < t < t_{n+1}, \\ 0, & \text{otherwise.} \end{cases}$$

As it is shown in [13, 7] the resulting system matrices are symmetric and positive definite. The stability results as well as the error estimates are presented therein.

Remark 2.1. As was already mentioned the computation is often simplified by using the temporal basis and test functions of lower order despite the fact that the stability and error estimates are no longer valid under this setting. In this case the integrals contain Dirac distributions in endpoints of time subintervals and must be treated with special care. The

evaluation of these integrals is in details described, e.g., in [14]. Another freaquently seen simplification lies in setting $\sigma = 0$. This again renders the estimates invalid. However, the parameter σ serves mainly to control the solution in infinity and it is not of such importance when dealing with finite time intervals [7].

One of the main problems with this approach lies in the fact that due to the retarded time argument in the variational formulation the scheme leads to an evaluation of integrals of the type

$$\int_{\tau_i} \int_{\{\boldsymbol{x}\in\tau_j : m\Delta t \le \|\boldsymbol{x}-\boldsymbol{y}\| \le (m+1)\Delta t\}} \dots \, \mathrm{d}s_{\boldsymbol{x}} \, \mathrm{d}s_{\boldsymbol{y}}, \tag{2.10}$$

i.e., an integration over an element and the intersection of an element and a light cone must be performed (see Figure 2.2). For this reason a Gaussian quadrature over pairs of elements does not converge optimally and a complicated techniques must be employed to overcome this problem (see, e.g., [33, 10]). We treat this issue by employing C^∞-smooth temporal basis functions with compact supports recently introduced by Sauter and Veit [27, 28, 31]. This recovers the approximation properties of the classical Gaussian quadrature as was shown in the abovementioned references. However, the evaluation of the smooth temporal basis functions, the assembly of the system matrices, and the solution of the system are significantly more computationaly demanding. Therefore a parallelization of the overall process is necessary for enabling the solution of large scale problems. This is in more details described in paper provided in Part II of this work.

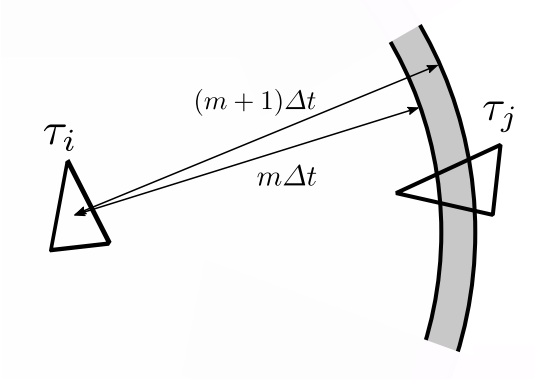


FIGURE 2.2: Integration over an intersection of a light cone and a triangle.

Chapter 3

Parallel boundary element environment BEM4I

Most of the techniques presented in this work have been implemented in the library of parallel boundary element solvers BEM4I developed at IT4Innovations National Supercomputing Center [22]. The library is written in C++ in an object oriented manner. The computation is parallelized using OpenMP and MPI in shared and distributed memory, respectively. Moreover, some parts of the library feture acceleration using the Intel Xeon Phi coprocessors. The library currently contains modules enabling solution of the Laplace, Lamé, Helmholtz, and time-dependent wave equation.

3.1 Structure of the library

The back-end of the library consists if three main sets of classes and several auxiliary classes (see Figure 3.1). The `BESpace` class and its descendants are used for approximation of the boundary element spaces. Among other things, they keep the degrees of the test and ansatz functions or properties associated with the approximation of the system matrices by means of a fast boundary element method.

The second level of classes is composed of bilinear form approximations stemming from the `BEBilinearForm` class. The main task of these classes is the assembly of system matrices. The parallelization in the shared memory by OpenMP and distributed memory by MPI is performed at this level. Moreover, some of the subclasses support acceleration by offloading the computation to the Intel Xeon Phi coprocessors.

At last, the numerical quadrature over pairs of elements described in Section 2.1.2 is performed by the class `BEIntegrator` and its descendants. Both semi-analytic and fully numerical approach is supported for the Laplace and Helmholtz equations. The fully numerical approach can be used to treat the Lamé and time-dependent wave equation. To leverage the

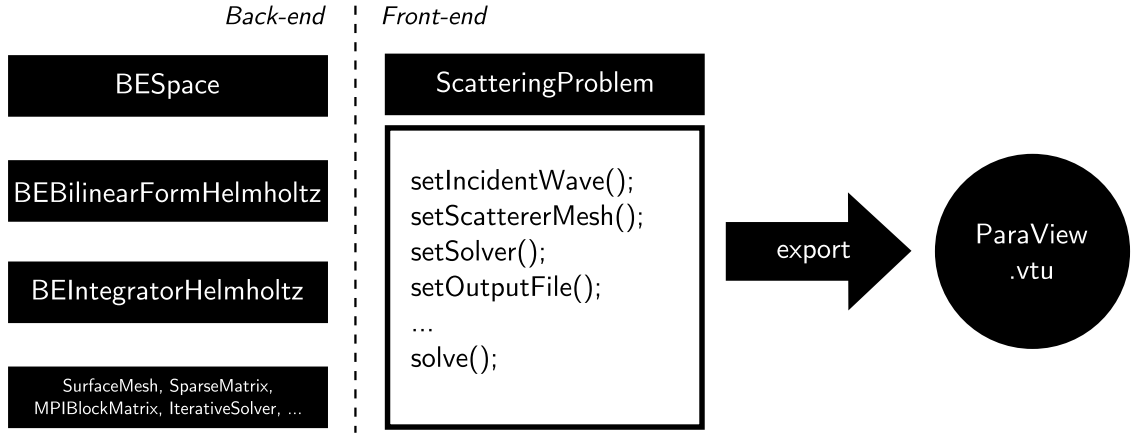


FIGURE 3.1: Structure of the solver for the time-harmonic wave scattering in the BEM4I library

SIMD capabilities of the modern processors the vectorization using the Vc library [17] and pragmas of the Intel compiler is employed at this level during the numerical quadrature.

The library also includes the classes enabling the manipulation with surface meshes, the classes representing the full and sparse matrices, iterative solvers, wrappers to the BLAS and LAPACK routines, methods for exporting the solution etc.

3.2 Parallelization of the code

3.2.1 Acceleration of the computation using SIMD instruction set extensions

Utilization of the SIMD (single instruction multiple data) instruction set extensions is necessary to achieve the full potential of modern processors. Using the vector instructions one can perform simultaneous operations on multiple operands during one clock cycle. The number of vector operands varies depending on the length of vector registers. While the 128-bit wide registers of the SSE - SSE4.2 (Streaming SIMD Extensions) instruction sets enabled concurrent computation on four 32-bit single precision and two 64-bit double precision operands, the AVX (Advanced Vector Extensions) instruction set available in Intel's processors since 2011 doubles this capacity. Moreover, currently available Knights Landing Intel Xeon Phi coprocessors enable utilizatiton of 512-bit wide vector registers via their IMCI (Initial Many Core Instruction) instruction set. With the upcoming AVX-512 similar capabilities will be available for all Intel's Xeon server processors. Thus it is clear that the employment of the vector instructions will be even more important in the future.

```
1
2 for ( int i = 0; i < nQuadsOuter) {
3    for ( int j = 0; j < nQuadsInner; j++ ) {
4       entry += singleLayerKernel(outerX[i], innerX[j]) * weightOut[i]
       * weightIn[j];
5    }
6 }
```

LISTING 3.1: Original loop over pair of elements during a local system matrix computation

```
1 int n = nQuadPointsInner * nQuadPointsOuter;
2 #pragma omp simd linear ( i : 1 ) reduction( + : total )
3 for ( i = 0; i < n; i++ ) {
4    entry += singleLayerKernel(outerX1[i], outerX2[i], outerX3[i],
5    innerX1[i], innerX2[i], innerX3[i] ) * weights[i];
6 }
```

LISTING 3.2: Loop vectorized using `#pragma simd`

```
1 int counter = 0;
2 for ( int i = 0; i < nQuadsOuter) {
3    for ( int j = 0; j < nQuadsInner; j++ ) {
4       outerX1[counter] = outerX[i][0];
5       outerX2[counter] = outerX[i][1];
6       ...
7       weights[counter] = weightOut[i] * weightIn[j];
8       counter++;
9    }
10 }
```

LISTING 3.3: Transformation of variables

The BEM4I library features two ways of vectorization. In [21] (see Part II) we have presented an approach based on the high level C++ library Vc [17]. This proved to be very efficient in combination with the GNU compiler. To enable the vectorization in combination with the Intel compiler we use the pragma based approach. For example, the original numerical quadrature over two well-separated elements during the assembly of a local system matrix depicted in Listing 3.1 is replaced with the manually colapsed loop provided in Listing 3.2. The original variables have to be transformed from the so-called array of structures form to the structure of arrays form (see Listing 3.3) and the arrays have to be properly aligned during their allocation using, e.g., `_mm_malloc` function. The proper vectorization of the computationally demanding loops is even more important when accelerating the code using the Intel Xeon Phi coprocessors, as will be described in Section 3.3.

3.2.2 Shared and distributed memory parallelization

While the vectorization is performed inside the `BEIntegrator` class during the local system matrix computation, the shared memory parallelization using OpenMP is carried out by the `BEBilinearForm` class which combines these local contributions into a global system matrix. The loops over pairs of elements are parallelized using the `#pragma omp parallel` statement. The scalability tests of the OpenMP parallelization in combination with the Vc vectorization are provided in [21] (see Part II). The BEM4I library provides shared memory parallelization of both dense and sparsified matrix assembly.

Moreover, to enable treatment of larger problems we provide the class `MPIACAMatrix` which distributes the matrix approximated by the adaptive cross approximation method among computational nodes (MPI processes). The ACA method splits a mesh into clusters. A pair of clusters represents a block in the system matrix. Well separated clusters are approximated using low rank approximation while close clusters are assembled as full matrices (for details see [2, 4]). The individual submatrices of the global ACA matrix are assigned in a round-robin fashion to the MPI processes. Since this is not an optimally load balanced method we provide an alternative approach based on the cyclic graph decomposition in [20] (see Part II).

A distribution of the system matrix arising in the time-domain solution of the wave equation is performed at this level as well. It is described in details in [34] available in Part II.

3.3 Acceleration by the Intel Xeon Phi coprocessors

In addition to the vectorization and OpenMP and MPI parallelization the BEM4I library accelerates the computation using the Intel Xeon Phi coprocessors. The first commercially available chip using Intel's Many Integrated Core (MIC) architecture released under the brand name Knights Corner (KNC) has been introduced in 2011. It features up to 61 cores with four hardware threads per core running at up to 1.2 GHz. The available memory bandwidth is 350 GB/s. The IMCI (Initial Many Core Instructions) instruction set provides support for a concurrent SIMD operations on eight double-precision or 16 single-precision operands. In overall, the coprocessor card provides over a TFLOPs of compute power. The biggest bottleneck of the technology is currently the data transfer via PCI Express bus. This will be eradicated in the next generations (Knights Landing and Knights Hill) which will be available in the form of stand-alone processors.

According to the November edition of the Top 500 list 29 out of 500 most powerful supercomputers are accelerated using the Intel Xeon Phi coprocessors, including the number one, Tianhe-2 (MilkyWay-2). This number is expected to grow with the introduction of

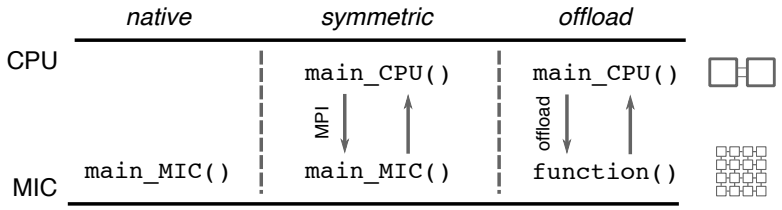


FIGURE 3.2: Intel Xeon Phi computation modes

the KNL and KNH generations. Announced systems utilizing these architectures include the NERCS's next supercomputing system Cori with the theoretical peak performance of 30 PFLOPs or Argonne's future Aurora cluster with estimated peak performance of up to 450 PFLOPS. With its high level of parallelization the Xeon Phi technology therefore represents one of the possible ways towards exascale computing.

There are three main compute modes when utilizing the Xeon Phi coprocessors (see Figure 3.2). In the native mode the whole application runs directly on the coprocessor. In the symmetric mode the MPI processes runs on both CPU and the coprocessor. A communication is performed using MPI messages. Finally, in the offload mode the application runs on CPU and only some parts of the code are offloaded to the coprocessor (see Figure 3.3).

The acceleration in the BEM4I library is based on the offload mode. The approach is in details described in [23]. Let us briefly explain how the original parallel loop assembling the system matrix on CPU (Listing 3.4) translates into the MIC-accelerated code. The original code loops through pairs of elements and based on a distance of elements decides whether to use the classical Gaussian quadrature (for far-enough elements) or one of the methods presented in Section 2.1.2 (for close or identical elements). After the evaluation of the local contribution it is added to the appropriate positions of the global system matrix (line 9 of the

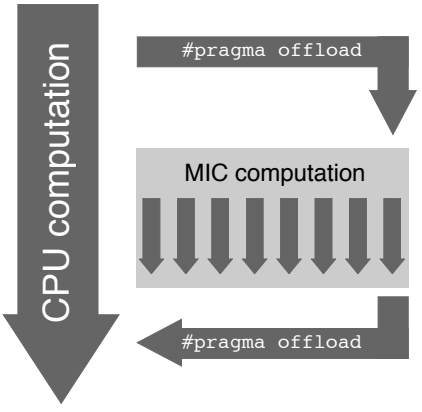


FIGURE 3.3: The offload compute mode

```
#pragma omp parallel for
for (int i = 0; i < nElements; i++) {
   for (int j = 0; j < nElements; j++) {
      if (areElementsDistant(i, j)) {
         getLocalMatrixFarfield(i, j, localMatrix);
      } else {
         getLocalMatrixNearfield(i, j, localMatrix);
      }
      globalMatrix.add(i, j, localMatrix);
   }
}
```

LISTING 3.4: Simplified CPU computation of the system matrix

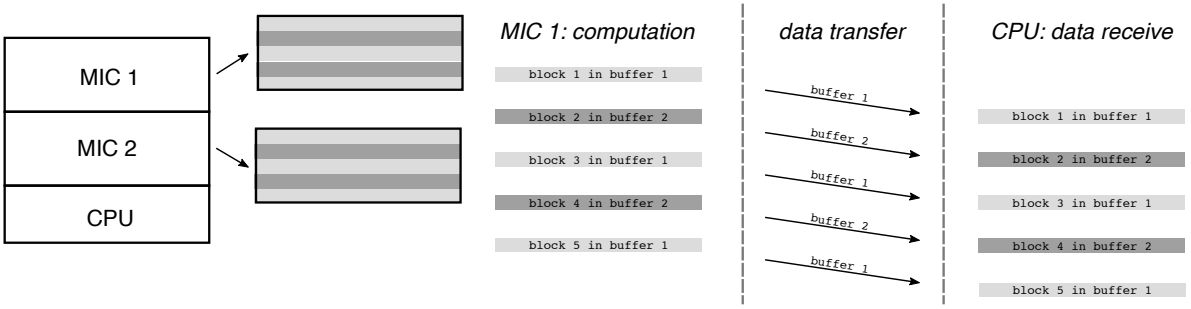


FIGURE 3.4: Matrix decomposition and double buffering

listing). The computation is done in parallel using OpenMP therefore the addition must be carried out using atomic operation.

During the accelerated computation the workload is distributed among the available coprocessors and CPU. The matrix is split into $N_{\text{MIC}}+1$ horizontal blocks while the dimension of blocks should follow the ratio between the theoretical peak performance of Xeon Phi and host CPU. The evaluation of the integrals over far-enough elements is split among coprocessors and CPU, however the integration over adjacent or identical elements is carried out by CPU in order to keep the computation on coprocessors as simple as possible. This does not result in a significant load balancing issue since the number of close elements only grows linearly, while the number of distant elements grows quadratically.

Since the amount of memory on the coprocessors is limited the submatrices to be assembled on Xeon Phis are further decomposed into smaller blocks. This enables a processing of the submatrix by parts. Moreover, the method of double buffering can be employed to hide the communication between the coprocessor and host by computation. In this case a pair of data buffers is created on a coprocessor and a host. When one of them is used for computation, data from the other one, computed during the previous iteration, is being sent to the host (see Figure 3.4).

In Listing 3.5 we provide a simplified source code for the offloaded computation. Two

```
1  // allocate computation buffers
2  double ** matrixBuffer = new double*[2];
3  matrixBuffer[0] = new double[ bufferSize ];
4  matrixBuffer[1] = new double[ bufferSize ];
5  // initiate parallel region on CPU
6  #pragma omp parallel
7  {
8     // thread num. 0 communicates with Xeon Phi
9     if ( omp_get_thread_num() == 0 ) {
10       for ( int i = 0; i < nSubmatrices; ++i ) {
11          #pragma offload target( mic ) signal( i ) \\
12          in ( elements ) in( nodes ) in( matrixBuffer[ i % 2] ) ...
13          {
14             // offloaded region, 244 threads available on MIC
15             #pragma omp parallel num_threads(244)
16             {
17                ...
18                for ( int j = myStart; j < myEnd; ++j ) {
19                   for ( int k = 0; k < nElems; ++k ) {
20                      getElementMatrixOnMIC( i, j, elemMatrix );
21                   }
22                }
23                addToGlobalMatrix( elemMatrix, matrixBuffer[i] );
24             }
25          }
26          // send result from the previous iteration to CPU
27          #pragma offload_transfer target( mic ) \\
28          wait( i - 1 ) out( matrixBuffer[ (i - 1) % 2 ] )
29       }
30    } else {
31       // the remaining CPU threads works on their portion of a matrix
32       ...
33    }
34 }
```

Listing 3.5: Simplified offloaded computation of the system matrix

data buffers for double buffering are allocated on lines 3 and 4. Next, the parallel region is initiated on CPU and one thread is assigned to communicate with the coprocessor. The thread iterates through all submatrices and for each submatrix it initiates the offload region on the line 12. The code inside of the offload region is evaluated on the coprocessor therefore we can use its 244 threads when iterating through the pairs of elements. On the line 24 the local matrices are added to the part of the global matrix stored in the appropriate buffer. Since the offload region was initiated with the clause `signal` the asynchronous computation was invoked. After its invocation the CPU thread does not wait until the offload region is finished. For this reason the transfer of data from the previous iteration on the lines 28 and 29 can be performed simultaneously with the computation. For the sake of simplicity we provided the code for offloading to a single Xeon Phi. Similar approach can be used when multiple coprocessor cards are available.

	Host	Host + 1 MIC				Host + 2 MICs			
# threads	24	60	120	180	240	60	120	180	240
V_h	128	135	90	77	70	92	62	54	53
K_h	137	173	108	91	88	120	76	60	55

TABLE 3.1: Assembly times [s] of the system matrices for the Laplace equation using CPU and the Intel Xeon Phi coprocessors

3.3.1 Numerical experiments

The following set of numerical experiments was carried out using the Salomon supercomputer at Supercomputing Center in Ostrava, Czech Republic. The cluster consists of 1008 compute nodes equipped with two 12-core Intel Xeon E5-2680v3 Haswell processors running at 2.5 GHz. 432 of the nodes are accelerated by two Intel Xeon Phi 7120P coprocessor cards. The cards features 61 cores with four hardware threads per core and 16 GB of RAM. The theoretical peak performance of the coprocessor is 1.2 TFLOP/s.

The times of the assembly of the system matrices V_h and K_h for the Laplace equations using the mesh consisting of 81920 elements are depicted in Table 3.1. In the column 'Host' the assembly times using 24 threads of the host CPU are depicted. The following four columns depict the assembly time for a concurrent computation on 23 threads of host CPU and an appropriate number of threads on one accelerator (one host's thread is assigned to communication with the coprocessor). Finally, the last four columns show the assembly times when 22 cores of CPU and both Xeon Phi accelerators are employed concurrently (two threads on CPU are dedicated to communication with the accelerators).

One of the 61 physical cores is left for the operating system. From the table one can see that it is crutial to employ all the hardware threads available in the cores in order to achieve maximum performance. Since the cores do not support out-of-order execution and one thread cannot issue instructions in two consecutive cycles, it is necessary to use at least two threads per core to feed the arithmetic logic unit. The maximum speed-ups with respect to the non-accelerated code was achieved using two accelerator cards are 2.41 for the matrix V_h and 2.49 for the matrix K_h.

Part II

Summary of author's contribution

Chapter 4

Author's publications on parallel boundary element method

4.1 Acceleration of boundary element method by explicit vectorization

The first presented paper describes the acceleration of BEM using the SIMD (Single Instruction Multiple Data) features of modern processors. While the parallelization of computation in shared and distributed memory has become standard in scientific community, the vectorization capabilities of modern processors are often neglected. Since the state of the art SIMD instruction sets (AVX-512, IMCI) enable concurrent operations on up to eight double precision or sixteen single precision operands, one can easily see why ignoring these capabilities may lead to inefficient code not capable of reaching theoretical performance of modern CPUs. The paper describes vectorization of both main approaches to evaluate singular integrals occuring during assembly of the BEM system matrices (semi-analytical and fully numerical). Details on the implementation in the BEM4I library are given. The numerical experiments demonstrates the speedup of assembly of the system matrices for the Helmholtz equation with respect to the used instruction set and floating point arithmetic. Moreover, the scalability of the vectorized code in shared memory is presented. The paper was published in **Advances in Engineering Software, Volume 86** in **2015**.

Author's main contribution to the paper include:

- **review of the possible vectorization techniques**

  There are several approaches to the vectorization of the code. One can utilize the low level assembly or intrinsic functions, however this leads to the portability issues and a need for rewriting the code for each newly introduced instruction set. Another approach is to exploit the auto-vectorization capabilities of compilers, possibly combined with

compilers' pragmas used to assist with the vectorization (e.g., `#pragma simd`, `#pragma omp simd`, `#pragma ivdep`). While these capabilities of compilers have been improving lately, at the time of writing of the paper, there were still needs for more detailed control of the vectorization to deal with the BEM computation. For this reason the external C++ library Vc was employed. It provides a high level wrapper on various SIMD instruction set extensions and enables branching of the vectorized code using masked instructions.

- **transformation of the quadrature kernels to be used with the vectorization library**

  The original methods used for numerical quadrature have to be accordingly rewritten in order to be used with the Vc library. Among the most important modifications we include the change of data layout from array of structures to structure of arrays. This significantly improves the memory performance since the coordinates of quadrature points are loaded from memory to vector registers with contiguous access. Furthermore, the original loops (especially for the semi-analytic quadrature) contain numerous conditional evaluations. These kind of loops are hard for compiler to auto-vectorize. With the Vc library they are treated using the masked instructions, as described in the paper.

- **implementation in the BEM4I environment**

- **numerical experiments on the Anselm cluster**

The numerical experiments in the paper have shown the relevance of the vectorization of BEM on modern CPU architectures. The tests were performed mainly using the SSE instruction set. In some cases the speed up of the vectorized code achieved the theoretical value of the used instruction set. In other cases lower speed up can be explained by frequent application of the masked vector instructions. Concerning the AVX and newer instruction sets, one can also see that their longer vector registers enable more accurate evaluation of numerical integrals without increasing the computational time. The obtained know-how becomes even more important when porting the code to modern many-core architectures (e.g., the Intel Xeon Phi coprocessors).

4.2 A parallel fast boundary element method using cyclic graph decompositions

The parallel implementation of the fast boundary element method for the elliptic problems is still an open topic. Most of the papers deal with the distributed versions of the various tree-based algorithms (e.g., fast multipole method, Barnes-Hut method) and omit the topic of

efficient parallel assembly and action of the system matrix. The second paper provided here presents a novel approach to distribution of the BEM system matrices sparsified using the adaptive cross approximation method (ACA) or the fast multipole method (FMM) among N computational nodes (MPI processes). The underlying mesh is decomposed into N submeshes, subsequently the system matrix is decomposed into $N \times N$ submatrices. Each of the processes is assigned N blocks, in such a way that minimizes the number of mesh part owned by the process. Moreover, since the diagonal blocks are the most time and memory consuming within the fast BEM, exactly one of them is assigned to each process. This way, the parallel scalability of $O((n/\sqrt{N})\log(n/N))$ is obtained for memory consumption per process, time for setup, matrix action and assembly.

The main contributions of the author are:

- **adapting the method for usage with FMM**

  The fast multipole method has lower memory requirements in comparison to the ACA method, thus enables a solution of larger problems. In the paper the method was used to approximate the individual blocks of the global system matrix.

- **numerical experiments on the Anselm cluster**

The paper was published in **Numerical Algorithms, Volume 70 (4)** in **2015**. The numerical experiments performed up to 2744832 surface elements and 273 MPI processes show that the computational time scales with $O(1/N)$ while the memory demands scale with $O(1/\sqrt{N})$. Although our method was applied to the problems modelled by the Laplace equation, it is applicable to other types of problems (such as linear elasticity or acoustic scattering) as well.

4.3 Efficient solution of time-domain boundary integral equations arising in sound-hard scattering

The last paper aims at efficient parallel solution of the time-dependent wave equation. As was mentioned in Part I, the efficient implementation and parallelization of the boundary element solver utilizing the smooth temporal basis functions is necessary to enable us to solve large engineering problems. Although the method simplifies the numerical quadrature and thus the assembly of system matrices, the evaluation of the smooth temporal basis functions is costly. Moreover, due to the structure of the system matrix the application of the methods like marching-on-time (MOT) is not possible. The system matrix is global in space and time. In order to preserve its special sparsity properties an iterative solver has to be applied to find a solution. Therefore, a development of a suitable preconditioner is necessary to reduce a high number of iterations of the conjugate gradient method. All of these tasks were aimed

in the following paper which was accepted for publication in **International Journal for Numerical Methods in Engineering** in **2015**.

Let us summarize the author's contribution to the paper:

- **development of the scheme for parallel distribution of block of system matrix**

  The system matrix for the space-time Galerkin method with smooth temporal basis function possesses a special structure. Since most of the blocks are duplicated, one has to take this into consideration when distributing the matrix blocks among computational nodes. At the same time the workload during the matrix vector multiplication has to be addressed. A possible approach to treat this by distribution of the block diagonals of matrix among computation nodes has been presented in the paper.

- **parallel assembly of the blocks of the system matrix**

  Despite several approaches to simplification of the temporal basis funcions evaluation and reduction of the number of their evaluation described in the paper, there is still the need for parallel assembly the individual matrix blocks. In the paper a hybrid parallelization by OpenMP and MPI has been presented. For each matrix block a non-zero pattern is precomputed and the pairs of elements contributing to the block are distributed among MPI processes. Assembly of the process' portion of data is parallelized in shared memory using OpenMP. This kind of work distribution enables us to parallelize the computation in both space and time.

- **development of a suitable iterative solver and preconditioner**

  Because of the implicitness of the scheme and the special block structure of the system matrix an iterative solver has to be employed to solve the arising linear system. In the paper several variants of the restarted GMRES algorithm have been used and compared – GMRES without preconditioning, DGMRES preconditioned by deflations, and FGMRES preconditioned by an experimental algebraic preconditioner. The original recursive algebraic preconditioner proposed by A. Veit was modified in order to be used with the FGMRES which enables nonexact solution of inner systems, which significantly reduces its application time.

- **parallel version of the recursive algebraic preconditioner**

- **numerical experiments on the Anselm cluster**

Without these contributions the method would only be able to solve problems of hundreds of spatial unknowns in reasonable time. The numerical experiments presented in the paper show that the code for the parallel assembly of the system matrix is able to scale up to more

than thousand of cores, reducing the computational time from hours to minutes. Moreover, the solution time is significantly reduced using the DGMRES iterative solver or the FGMRES iterative solver in combination with the proposed recursive algebraic preconditioner, as is shown on several examples.

Chapter 5

Full text of the papers

Advances in Engineering Software 86 (2015) 70–79

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Acceleration of boundary element method by explicit vectorization

Michal Merta *, Jan Zapletal

IT4Innovations National Supercomputing Center, VŠB-TU Ostrava, 17. listopadu 15/2172, 708 33 Ostrava, Czech Republic
Dept. of Applied Mathematics, VŠB-TU Ostrava, 17. listopadu 15/2172, 708 33 Ostrava, Czech Republic

ARTICLE INFO

Article history:
Received 23 February 2015
Received in revised form 7 April 2015
Accepted 11 April 2015

Keywords:
Boundary element method
Sound scattering
Helmholtz equation
Vectorization
SIMD
OpenMP parallelization

ABSTRACT

Although parallelization of computationally intensive algorithms has become a standard with the scientific community, the possibility of in-core vectorization is often overlooked. With the development of modern HPC architectures, however, neglecting such programming techniques may lead to inefficient code hardly utilizing the theoretical performance of nowadays CPUs. The presented paper reports on explicit vectorization for quadratures stemming from the Galerkin formulation of boundary integral equations in 3D. To deal with the singular integral kernels, two common approaches including the semi-analytic and fully numerical schemes are used. We exploit modern SIMD (Single Instruction Multiple Data) instruction sets to speed up the assembly of system matrices based on both of these regularization techniques. The efficiency of the code is further increased by standard shared-memory parallelization techniques and is demonstrated on a set of numerical experiments.

1. Introduction

The boundary element method (BEM) is a counterpart to the finite element method (FEM) suitable for the solution of partial differential equations which can be formulated in the form of boundary integral equations. Since BEM reduces the given problem to the boundary of a computational domain, it is especially suitable for problems stated in unbounded domains, such as acoustic or electromagnetic wave scattering, or shape optimization.

System matrices arising from the classical BEM are dense and the method has quadratic computational and memory complexity with respect to the number of surface elements. Moreover, special quadrature methods are needed due to the singularities in the kernels of the boundary integrals [1,2], which further contribute to computational demands of the method. Several fast BEM approaches can be employed to reduce computational and memory requirements to almost linear. The common methods are based on the decomposition of the surface mesh into clusters and subsequent low rank approximation of matrix blocks corresponding to admissible pairs of clusters. Nonadmissible blocks are assembled in the standard way as full rank matrices. The fast multipole method (FMM) is based on the approximation of the system matrices by the multipole series expansion [3–5], whereas the adaptive cross approximation (ACA) assembles the low rank approximation from an algebraic point of view [6,2].

Regardless the above mentioned approximation techniques, there is still a need for an efficient assembly of nonadmissible matrix blocks or a certain number of rows and columns of admissible blocks in the case of ACA. Since these blocks are usually too small to be distributed among computational nodes by MPI, an OpenMP parallelization of the assembly is an obvious choice. In this paper, we discuss further acceleration of the process by means of vectorization of the quadrature over pairs of surface elements.

With new SIMD instruction sets available in modern processors the usage of vectorization becomes more important in scientific computation. Neglecting it may lead to inefficient code not capable of reaching the theoretical performance of current CPUs. The SSE instruction set introduced by Intel in 1999 provided eight 128-bit registers and enabled concurrent operations on four 32-bit single-precision floating point numbers. Its successors, SSE2–SSE4, extended this capability to support SIMD operations on two 64-bit double-precision floating point operands while incrementally adding more instructions. The AVX instruction set supported by Intel processors since 2011 extends the registers length from 128 bits to 256 bits and introduces a three-operand SIMD instruction format. Its capabilities are further extended by AVX2. The AVX-512 should provide registers with 512-bit length allowing for concurrent operation on eight 64-bit double precision numbers. Its support is announced for Intel's Knights Landing processor available in 2015 and for Intel's Skylake microprocessor architecture [7].

* Corresponding author at: IT4Innovations National Supercomputing Center, VŠB-TU Ostrava, 17. listopadu 15/2172, 708 33 Ostrava, Czech Republic.
E-mail addresses: michal.merta@vsb.cz (M. Merta), jan.zapletal@vsb.cz (J. Zapletal).

http://dx.doi.org/10.1016/j.advengsoft.2015.04.008
0965-9978/

To use the vector instructions the existing scalar code usually has to be modified. While the automatic loop vectorization provided by the compiler is not capable of vectorizing more complex loops often occurring in scientific codes, exploiting the supported intrinsic functions may lead to a confusing and hardly maintainable code. One of the possibilities avoiding these issues is to use a higher level library, such as VML from Intel's Math Kernel Library [8], VDT [9], or the Vc library [10], which is the main focus of this paper. The library provides a high level wrapper on SIMD intrinsics and enables explicit vectorization of C++ code. It is portable among various compilers and SIMD instruction sets and enables easy vectorization without the need for a major redesign of the existing object oriented C++ code.

The topic of the vectorization of the BEM computation has been presented in several publications. In [11] an example of automatic loop vectorization of Fortran boundary element computation is provided. The original routines are manually altered using techniques such as loop unrolling and loop reordering in order to enable the compiler to employ SIMD instructions. Although a reasonable speedup with respect to the non-vectorized version is obtained, modifications lead to a significantly more complex code. The interested reader may also consult [12] for a comprehensive presentation of BEM quadrature vectorization. The author provides a general overview of the SIMD parallelism, compares two approaches to handling data during the computation (inter- and intra-register operations), and presents results of numerical experiments with a code vectorized using intrinsic functions. However, the work does not discuss the treatment of singularities in the related surface integrals, which is one of the crucial tasks of BEM computations.

The structure of the paper is as follows. In the next section we provide a model problem on which we demonstrate the boundary element workflow. In Section 3, a short overview of our BEM library is provided, Section 4 discusses the vectorization of the computationally most demanding parts of the code. Finally, we provide results of numerical experiments and conclude.

2. Boundary element method for sound-hard scattering

In this section we present the model problem under consideration, derive the corresponding boundary integral equations and their Galerkin discretization.

2.1. Helmholtz boundary integral equation

We consider a time-harmonic scattering problem with a sound-soft obstacle modelled by a bounded Lipschitz domain $\Omega \subset \mathbb{R}^3$. The incident wave is of the form $u_\mathrm{i} := e^{\mathrm{i}\kappa\langle \boldsymbol{x},\boldsymbol{d}\rangle}$ with the imaginary unit i, the wave number $\kappa \in \mathbb{R}_+$, and a unit direction vector $\boldsymbol{d}$. The wave u_s scattered from Ω satisfies the exterior Dirichlet boundary value problem for the Helmholtz equation [13,2]

$$\begin{cases} \Delta u_\mathrm{s} + \kappa^2 u_\mathrm{s} = 0 & \text{in } \Omega^{\mathrm{ext}} := \mathbb{R}^3 \setminus \overline{\Omega}, \\ u_\mathrm{s} = g & \text{on } \partial\Omega, \\ \left| \left\langle \nabla u_\mathrm{s}(\boldsymbol{x}), \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|} \right\rangle - \mathrm{i}\kappa u_\mathrm{s}(\boldsymbol{x}) \right| = \mathcal{O}\left(\frac{1}{\|\boldsymbol{x}\|^2}\right) & \text{for } \|\boldsymbol{x}\| \to \infty. \end{cases} \tag{1}$$

The total wave field around the obstacle is given by $u_\mathrm{t} := u_\mathrm{i} + u_\mathrm{s}$. The Dirichlet condition in (1) is given by the negative of the incident wave $g := -u_\mathrm{i}$. Thus, the total wave field u_t vanishes on $\partial\Omega$. The Sommerfeld radiation condition ensures uniqueness of the solution $u_\mathrm{s} \in H^1_{\Delta,\mathrm{loc}}(\Omega^{\mathrm{ext}})$, which is equivalent to

$$u_\mathrm{s}|_{\widetilde{\Omega}} \in H^1_\Delta(\widetilde{\Omega}) := \left\{ u \in L^2(\widetilde{\Omega}) : \frac{\partial u}{\partial x_i} \in L^2(\widetilde{\Omega}) \wedge \Delta u \in L^2(\widetilde{\Omega}) \right\},$$

for all bounded domains $\widetilde{\Omega} \subset \Omega^{\mathrm{ext}}$ with the derivatives understood in the distributional sense.

To solve the boundary value problem (1) we use the direct approach via the representation formula [13,14,2,1,15]

$$u_\mathrm{s} = W_\kappa \gamma^{0,\mathrm{ext}} u_\mathrm{s} - \widetilde{V}_\kappa \gamma^{1,\mathrm{ext}} u_\mathrm{s} \quad \text{in } \Omega^{\mathrm{ext}}, \tag{2}$$

with the trace operators

$$\gamma^{0,\mathrm{ext}} : H^1_{\Delta,\mathrm{loc}}(\Omega^{\mathrm{ext}}) \to H^{1/2}(\partial\Omega), \quad \gamma^{0,\mathrm{ext}} u = u|_{\partial\Omega} \text{ for } u \in C^\infty(\overline{\Omega^{\mathrm{ext}}}),$$
$$\gamma^{1,\mathrm{ext}} : H^1_{\Delta,\mathrm{loc}}(\Omega^{\mathrm{ext}}) \to H^{-1/2}(\partial\Omega), \quad \gamma^{1,\mathrm{ext}} u = \frac{\partial u}{\partial \boldsymbol{n}} \text{ for } u \in C^\infty(\overline{\Omega^{\mathrm{ext}}}),$$

$\boldsymbol{n}$ denoting the unit exterior normal vector to Ω, the single-layer and double-layer potentials

$$\widetilde{V}_\kappa : H^{-1/2}(\partial\Omega) \to H^1_{\Delta,\mathrm{loc}}(\Omega^{\mathrm{ext}}),$$
$$(\widetilde{V}_\kappa q)(\boldsymbol{x}) := \int_{\partial\Omega} v_\kappa(\boldsymbol{x},\boldsymbol{y}) q(\boldsymbol{y}) \, d\boldsymbol{s}_\boldsymbol{y}, \tag{3}$$

$$W_\kappa : H^{1/2}(\partial\Omega) \to H^1_{\Delta,\mathrm{loc}}(\Omega^{\mathrm{ext}}),$$
$$(W_\kappa t)(\boldsymbol{x}) := \int_{\partial\Omega} \frac{\partial v_\kappa}{\partial \boldsymbol{n}_\boldsymbol{y}}(\boldsymbol{x},\boldsymbol{y}) t(\boldsymbol{y}) \, d\boldsymbol{s}_\boldsymbol{y}, \tag{4}$$

and the fundamental solution of the Helmholtz equation in 3D

$$v_\kappa(\boldsymbol{x},\boldsymbol{y}) := \frac{1}{4\pi} \frac{e^{\mathrm{i}\kappa\|\boldsymbol{x}-\boldsymbol{y}\|}}{\|\boldsymbol{x}-\boldsymbol{y}\|}.$$

To evaluate the solution u_s in $\boldsymbol{x} \in \Omega^{\mathrm{ext}}$ using the formula (2) it is necessary to complete the Cauchy data $\gamma^{0,\mathrm{ext}} u_\mathrm{s}$, $\gamma^{1,\mathrm{ext}} u_\mathrm{s}$. Since the Dirichlet trace $\gamma^{0,\mathrm{ext}} u_\mathrm{s}$ is given by $g = -\gamma^{0,\mathrm{ext}} u_\mathrm{i}$, only the missing Neumann trace $\gamma^{1,\mathrm{ext}} u_\mathrm{s}$ needs to be computed. Applying the Dirichlet trace operator $\gamma^{0,\mathrm{ext}}$ to the representation formula (2) and using the well-known properties of the potential operators (3) and (4) (see, e.g., [14,2,15,1]) we obtain the boundary integral equation

$$(V_\kappa \gamma^{1,\mathrm{ext}} u_\mathrm{s})(\boldsymbol{x}) = -\frac{1}{2} g(\boldsymbol{x}) + (K_\kappa g)(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \partial\Omega, \tag{5}$$

with the single-layer and double-layer boundary integral operators

$$V_\kappa : H^{-1/2}(\partial\Omega) \to H^{1/2}(\partial\Omega), \quad (V_\kappa q)(\boldsymbol{x}) := \int_{\partial\Omega} v_\kappa(\boldsymbol{x},\boldsymbol{y}) q(\boldsymbol{y}) \, d\boldsymbol{s}_\boldsymbol{y}, \tag{6}$$

$$K_\kappa : H^{1/2}(\partial\Omega) \to H^{1/2}(\partial\Omega), \quad (K_\kappa t)(\boldsymbol{x}) := \int_{\partial\Omega} \frac{\partial v_\kappa}{\partial \boldsymbol{n}_\boldsymbol{y}}(\boldsymbol{x},\boldsymbol{y}) t(\boldsymbol{y}) \, d\boldsymbol{s}_\boldsymbol{y}. \tag{7}$$

Note that contrary to the potentials (3) and (4), the functions $V_\kappa q$, $K_\kappa t$ are only defined on $\partial\Omega$.

The Galerkin formulation equivalent to (5) reads

$$\langle V_\kappa \gamma^{1,\mathrm{ext}} u_\mathrm{s}, s \rangle_{\partial\Omega} = \left\langle \left(-\frac{1}{2} I + K_\kappa \right) g, s \right\rangle_{\partial\Omega} \quad \text{for all } s \in H^{-1/2}(\partial\Omega). \tag{8}$$

2.2. Boundary element method

To discretize the Galerkin formulation (8) we triangulate the surface $\partial\Omega$ into E flat shape-regular open triangles τ_i, i.e.,

$$\partial\Omega \approx \bigcup_{n=1}^{E} \overline{\tau_i}.$$

To approximate the Cauchy data we use piecewise linear ansatz for the Dirichlet data $\gamma^{0,\mathrm{ext}} u_\mathrm{s}$ and piecewise constant ansatz for the Neumann data $\gamma^{1,\mathrm{ext}} u_\mathrm{s}$

$$\gamma^{0,\mathrm{ext}} u_\mathrm{s} = g \approx t := \sum_{i=1}^{N} t_j \varphi_j, \quad \gamma^{1,\mathrm{ext}} u_\mathrm{s} \approx s := \sum_{i=1}^{E} s_i \psi_i,$$

where N denotes the total number of mesh nodes. Using piecewise constant testing functions ψ_ℓ this results in the discrete system of linear equations

$$\mathsf{V}_{\kappa,h}\boldsymbol{s} = \left(-\frac{1}{2}\mathsf{M}_h + \mathsf{K}_{\kappa,h}\right)\boldsymbol{t},$$

with the matrices

$$\mathsf{V}_{\kappa,h}[\ell,j] := \frac{1}{4\pi}\int_{\tau_\ell}\int_{\tau_j}\frac{e^{\mathrm{i}\kappa\|\boldsymbol{x}-\boldsymbol{y}\|}}{\|\boldsymbol{x}-\boldsymbol{y}\|}\,d\boldsymbol{s}_{\boldsymbol{y}}\,d\boldsymbol{s}_{\boldsymbol{x}},$$

$$\mathsf{M}_h[\ell,i] := \int_{\tau_\ell}\varphi_i(\boldsymbol{x})\,d\boldsymbol{s}_{\boldsymbol{x}},$$

$$\mathsf{K}_{\kappa,h}[\ell,i] := \frac{1}{4\pi}\int_{\tau_\ell}\int_{\partial\Omega}\varphi_i(\boldsymbol{y})\frac{e^{\mathrm{i}\kappa\|\boldsymbol{x}-\boldsymbol{y}\|}}{\|\boldsymbol{x}-\boldsymbol{y}\|^3}(1-\mathrm{i}\kappa\|\boldsymbol{x}-\boldsymbol{y}\|)\langle\boldsymbol{x}-\boldsymbol{y},\boldsymbol{n}(\boldsymbol{y})\rangle\,d\boldsymbol{s}_{\boldsymbol{y}}\,d\boldsymbol{s}_{\boldsymbol{x}}.$$

To set up the matrices one has to deal with integration of singular kernels. There are two commonly used methods, one involving analytic integration of the inner integral and numerical quadrature of the outer one [2], or a fully numerical scheme described in [1] and references therein. Since the treatment of all operators is very similar, in the two following sections we only describe the set up of the single-layer operator matrix $\mathsf{V}_{\kappa,h}$ using both approaches. In Section 5, however, we provide results of our experiments for the set up of both boundary element matrices $\mathsf{V}_{\kappa,h}$, $\mathsf{K}_{\kappa,h}$. In addition, we also provide assembly times of the hypersingular operator matrix

$$\begin{aligned}\mathsf{D}_{\kappa,h}[i,j] :&= \frac{1}{4\pi}\int_{\partial\Omega}\int_{\partial\Omega}\frac{e^{\mathrm{i}\kappa\|\boldsymbol{x}-\boldsymbol{y}\|}}{\|\boldsymbol{x}-\boldsymbol{y}\|}\langle\mathrm{curl}_{\partial\Omega}\varphi_j(\boldsymbol{y}),\mathrm{curl}_{\partial\Omega}\varphi_i(\boldsymbol{x})\rangle\,d\boldsymbol{s}_{\boldsymbol{y}}\,d\boldsymbol{s}_{\boldsymbol{x}}\\ &\quad-\frac{\kappa^2}{4\pi}\int_{\partial\Omega}\int_{\partial\Omega}\frac{e^{\mathrm{i}\kappa\|\boldsymbol{x}-\boldsymbol{y}\|}}{\|\boldsymbol{x}-\boldsymbol{y}\|}\varphi_j(\boldsymbol{y})\varphi_i(\boldsymbol{x})\langle\boldsymbol{n}(\boldsymbol{x}),\boldsymbol{n}(\boldsymbol{y})\rangle\,d\boldsymbol{s}_{\boldsymbol{y}}\,d\boldsymbol{s}_{\boldsymbol{x}},\end{aligned}$$

which would be involved, e.g., in a sound-hard scattering problem with the Neumann boundary condition $\frac{\partial u_s}{\partial\boldsymbol{n}} = -\frac{\partial u_i}{\partial\boldsymbol{n}}$ [2,1]. The semi-analytic approach for this matrix is described in [16].

2.3. Semi-analytic assembly

Let us first address the semi-analytic approach. To find the analytic formula the singular inner integral is split up as

$$\int_{\tau_j}\frac{1}{4\pi}\frac{e^{\mathrm{i}\kappa\|\boldsymbol{x}-\boldsymbol{y}\|}}{\|\boldsymbol{x}-\boldsymbol{y}\|}\,d\boldsymbol{s}_{\boldsymbol{y}} = \int_{\tau_j}\frac{1}{4\pi}\frac{e^{\mathrm{i}\kappa\|\boldsymbol{x}-\boldsymbol{y}\|}-1}{\|\boldsymbol{x}-\boldsymbol{y}\|}\,d\boldsymbol{s}_{\boldsymbol{y}} + \int_{\tau_j}\frac{1}{4\pi}\frac{1}{\|\boldsymbol{x}-\boldsymbol{y}\|}\,d\boldsymbol{s}_{\boldsymbol{y}}.$$

Let us denote

$$f(\boldsymbol{x}) := \int_{\tau_j}\frac{1}{4\pi}\frac{e^{\mathrm{i}\kappa\|\boldsymbol{x}-\boldsymbol{y}\|}-1}{\|\boldsymbol{x}-\boldsymbol{y}\|}\,d\boldsymbol{s}_{\boldsymbol{y}},\quad g(\boldsymbol{x}) := \int_{\tau_j}\frac{1}{4\pi}\frac{1}{\|\boldsymbol{x}-\boldsymbol{y}\|}\,d\boldsymbol{s}_{\boldsymbol{y}}. \tag{9}$$

For the first integrand we have

$$\lim_{\boldsymbol{x}\to\boldsymbol{y}}\frac{e^{\mathrm{i}\kappa\|\boldsymbol{x}-\boldsymbol{y}\|}-1}{\|\boldsymbol{x}-\boldsymbol{y}\|} = \lim_{a\to0}\frac{e^{\mathrm{i}\kappa a}-1}{a}\overset{l'\mathrm{H}}{=}\lim_{a\to0}\mathrm{i}\kappa e^{\mathrm{i}\kappa a} = \mathrm{i}\kappa,$$

and thus the integral $f(\boldsymbol{x})$ can be computed numerically without further regularization as

$$f(\boldsymbol{x}) \approx \sum_{n=1}^{k}\omega_n\frac{1}{4\pi}\frac{e^{\mathrm{i}\kappa\|\boldsymbol{x}-\boldsymbol{y}_n\|}-1}{\|\boldsymbol{x}-\boldsymbol{y}_n\|},$$

with suitable quadrature weights ω_n and points $\boldsymbol{y}_n \in \tau_j$. For the quadrature over τ_j we use the 7-point Gauss scheme described in [2]. The explicit analytic formula for g in local coordinates related to τ_j can be found in Chapter C.2 in [2]. For the outer integration of $f+g$ over τ_ℓ we use the Gauss integration

$$\mathsf{V}_{\kappa,h}[\ell,j] \approx \sum_{m=1}^{k}\omega_m(f(\boldsymbol{x}_m)+g(\boldsymbol{x}_m)). \tag{10}$$

2.4. Numerical assembly

Since the fully numerical approach is quite involved, we only summarize the basic concept and refer the interested reader to Chapter 5 of [1] and the references therein.

The main aim is to regularize the integrand by means of integral substitutions. This leads to a formal formulation given by four one-dimensional integrals

$$\int_0^1\int_0^1\int_0^1\int_0^1 \nu_\kappa(\boldsymbol{F}(z_1,z_2,z_3,z_4))\boldsymbol{S}(z_1,z_2,z_3,z_4)\,dz_1\,dz_2\,dz_3\,dz_4,$$

with the substitution $\boldsymbol{F}_n\circ\cdots\circ\boldsymbol{F}_1 =: \boldsymbol{F} : [0,1]^4 \to \tau_i\times\tau_j$,

$$\boldsymbol{F}(z_1,z_2,z_3,z_4) = (\boldsymbol{x},\boldsymbol{y}),\quad d\boldsymbol{s}_{\boldsymbol{y}}\,d\boldsymbol{s}_{\boldsymbol{x}} = \boldsymbol{S}(z_1,z_2,z_3,z_4)\,dz_1\,dz_2\,dz_3\,dz_4,$$

which is different for τ_i, τ_j being identical, sharing exactly one edge, sharing exactly one vertex, or having a positive distance. The function $\boldsymbol{S}$ stems from the parametrization via a reference triangle and a series of transformations given by $\boldsymbol{F}$. Since the integrand

$$h(z_1,z_2,z_3,z_4) := \nu_\kappa(\boldsymbol{F}(z_1,z_2,z_3,z_4))\boldsymbol{S}(z_1,z_2,z_3,z_4),$$

is analytic [1], a tensor-product Gaussian quadrature scheme

$$\sum_{m=1}^{k}\sum_{n=1}^{k}\sum_{o=1}^{k}\sum_{p=1}^{k}\omega_m\omega_n\omega_o\omega_p\,h(z_m,z_n,z_o,z_p), \tag{11}$$

with the weights $\omega_\bullet$ and sampling points $z_\bullet$ can be used.

3. The BEM4I library

The solver for the wave scattering problem based on BEM is implemented in the BEM4I library [17]. The library is written using C++ in an object-oriented way. It utilizes templates to support various indexing and scalar types. OpenMP is used for the parallelization in shared memory and some parts of the code are parallelized in distributed memory by MPI.

The structure of the solver is depicted in Figs. 1 and 2. Three main types of classes are responsible for the assembly of the system matrices.

1. `BESpace` The class and its descendants keep the information needed for the boundary element approximation of the continuous function spaces, such as the order of spatial basis and testing functions or data necessary for the FMM or ACA acceleration. The object of the computational mesh is stored in this class.
2. `BEBilinearForm` The main purpose of the descendants of this class is to assemble appropriate system matrices (either full or sparsified by ACA or FMM). The assembly is performed elementwise using `BEIntegrator` classes. Local contributions from individual pairs of elements are combined to form a global system matrix. The assembly is parallelized using OpenMP at this level.
3. `BEIntegrator` The group of `BEIntegrator` classes is responsible for the assembly of element system matrices and evaluation of the representation formula (2). The most computationally demanding part of the code is the Gaussian quadrature over the pairs of elements. Its vectorization by the Vc library will be described in the next section.

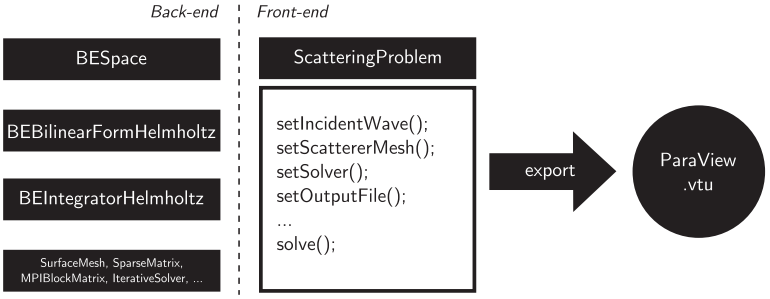


Fig. 1. Structure of the wave scattering solver in the BEM4I library.

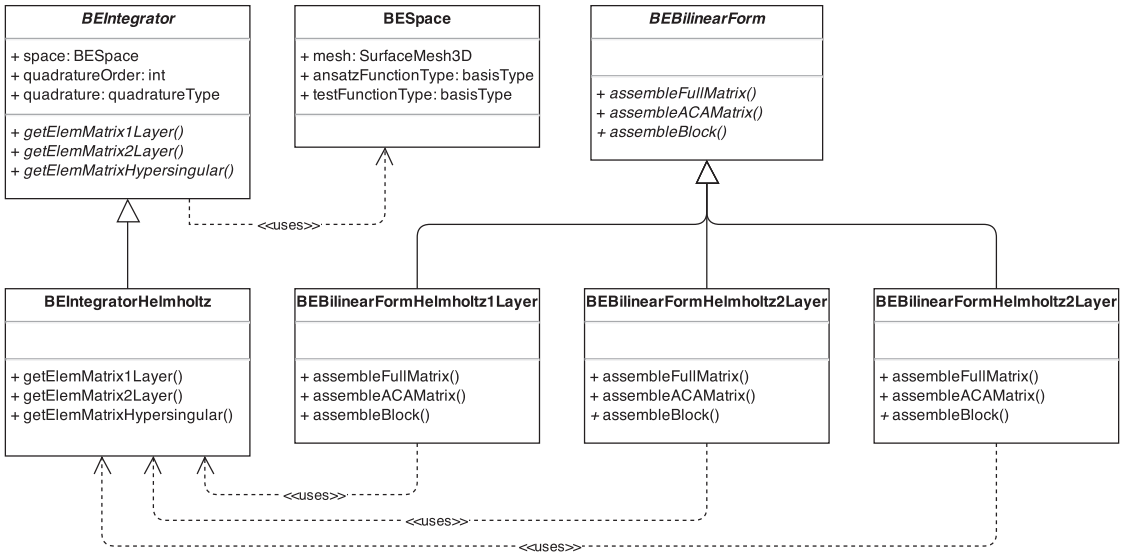


Fig. 2. Simplified class diagram of the wave scattering solver in the BEM4I library.

Besides these main classes the library also implements various supportive classes representing surface meshes, full and sparse matrices, matrices approximated by FMM and ACA, direct and iterative solvers, etc. To solve the problem, the user can either directly manipulate with these classes or use the interface provided in the `ScatteringProblem` class.

4. Vectorization of the numerical quadrature

In the following section we demonstrate the applicability of the Vc library to the vectorization of the quadrature occurring in the boundary element matrices computation. The BEM4I library implements both the semi-analytic [2,16] and numerical [1] methods, therefore the vectorized code is provided for both approaches.

The Vc library contains counterparts of most mathematical methods of the C++ Standard Library which makes it relatively easy to convert the quadrature code from a scalar to a vectorized version. The development of the vectorized code is faster since the structures of both non-vectorized and vectorized versions are similar, the object-oriented design is preserved and the maintenance of the code is easier. In some cases a more complex redesign of the code may be necessary to further increase its efficiency.

A common header file interface using standard C++ data structures is provided for both vectorized and non-vectorized versions. Therefore, the user does not interact directly with the Vc library and the external code remains the same for both scalar and vector computations. The vectorization is activated using the `#define INT_VC` preprocessor directive in the settings file.

For the sake of clarity the template parameters are omitted from the following source code samples. The `int` and `double` types are used for indexing and the default float type, respectively.

4.1. Semi-analytic assembly vectorization

The main idea of the vectorization of the semi-analytic matrix assembly is to parallelly evaluate the outer quadrature (10), which leads to concurrent evaluation of the functions f and g from (9) in multiple quadrature points. A common interface for both vectorized and scalar versions of the single-layer operator local matrix assembly is provided by the method `computeElemMatrix1LayerPOPO`.

```
void computeElemMatrix1LayerPOPO (
  int outerElem,
  int innerElem,
  FullMatrix& matrix
  );
```

The first part of the method consists mainly of data allocation, establishing the local element coordinate system, and computation of Gaussian quadrature nodes. Contrary to the scalar version,

where this is done using the classical C++ data structures, the vectorized version stores the coordinates of quadrature nodes as well as quadrature weights in instances of the `Vc::Memory < Vc::Vector >` class. The class ensures correct memory alignment depending on the underlying data type and allows efficient scalar and vector access to data. The `Vc::Vector` class abstracts the SIMD registers and their instructions into types familiar to C++ developers. The capacity of the vector depends on the instruction set used. With AVX it can store and concurrently process four double precision and eight single precision operands.

In the next part of the method the Gaussian quadrature over the outer element is performed. The original scalar loop over quadrature points.

```
double entry = 0.0;
for (int i = 0; i < number_of_quadrature_points;
  i++) {
  double value = collocation
  lLayerP0(quadrature_point[i]);
  entry += quadrature_weights[i] * value;
}
```

is replaced by the vectorized loop. The simplified version of the vectorized loop follows.

```
Vc::Memory < Vc::Vector < double > >
  quadrature_weights;
Vc::Memory < Vc::Vector < double > > x0; // first
  coordinates
Vc::Memory < Vc::Vector < double > > x1; // second
  coordinates
Vc::Memory < Vc::Vector < double > > x2; // third
  coordinates
// ...(fill the instances of Vc::Memory class with data)
Vc::Vector < double > entry = Vc::Zero;
for (int i = 0; i < quadrature_
  weights.vectorsCount( ); i++) {
  Vc::Vector < double > value = collocationl
  LayerP0(
    x0.vector(i), x1.vector(i), x2.vector(i));
  entry += quadrature_weights.vector(i) * value;
}
double totalEntry = entry.sum();
```

The computationally most demanding part of the code is the evaluation of the function `collocationlLayerP0` which returns the values of f and g from (10). The method involves evaluation of several transcendental functions, such as logarithms, trigonometric, or inverse trigonometric functions, as well as multiple conditional expressions. Its automatic vectorization is beyond the abilities of current compilers and its proper vectorization using the Vc library is crucial for the efficiency of the code. A snippet of the original method is provided in the following listing.

```
double tmp1 = s - x[0];
double tmp2 = alpha * s - x[1];
if (std::abs(tmp1) > eps) {
  if (tmp2 < 0.0)
    f += tmp1 * std::log((x[2] * x[2] +
      tmp1 * tmp1)/ (a - tmp2));
  else
    f += tmp1 * std::log(tmp2 + a);
}
```

Here, `x` is an array storing three spatial coordinates of a collocation point. Note the nested conditional expression, which is necessary to deal with possible singularities in the integral and complicates the evaluation of the variable `f`.

The vectorized version processes multiple integration points concurrently. Their coordinates are separated into three `Vc::Vector` objects `x0`, `x1`, `x2` containing first, second, and third components, respectively (see Fig. 3). Other variables are substituted with their vector counterparts as well. The `if` statement is replaced with conditional write based on a comparison of two or more vectors as demonstrated in Fig. 4 and the listing below. The results of the right-hand side operations are only stored on those positions of the `logArg` vector, for which the given condition is satisfied.

```
Vc::Vector < double > tmp1 = s - x0;
Vc::Vector < double > tmp2 = alpha * s - x1;
Vc::Vector < double > logArg(Vc::Zero);
logArg(Vc::abs(tmp1) > eps && tmp2 < Vc::Zero) =
  (x2 * x2 + tmp1 * tmp1) / (a - tmp2);
logArg(Vc::abs(tmp1) > eps && tmp2 >= Vc::Zero) =
  tmp2 + a;
f(Vc::abs(tmp1) > eps) += tmp1 * Vc::log(logArg);
```

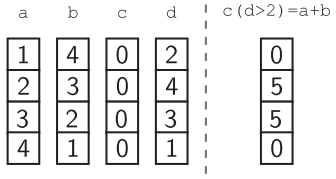


Fig. 4. Conditional write to the `Vc::Vector` object.

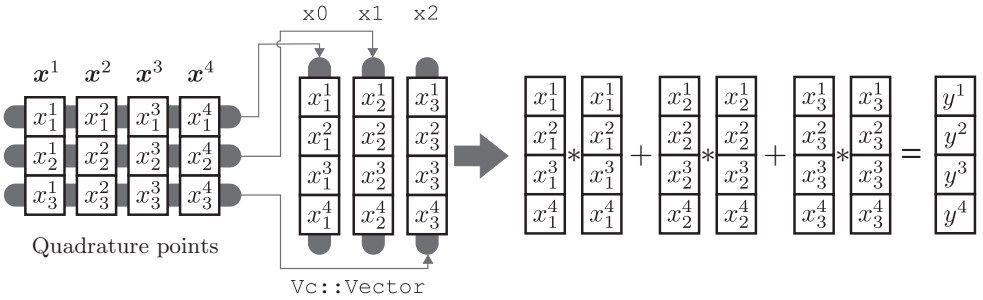


Fig. 3. Vector computation of $y^i := \|\boldsymbol{x}^i\|^2 = (x_1^i)^2 + (x_2^i)^2 + (x_3^i)^2$.

The rest of the method `collocationlLayerP0` is vectorized in the same manner. The method returns the `Vc::Vector` object which is further processed by the Gaussian quadrature mentioned above. The resulting code is structurally very similar to the original one, while the performance gain is significant.

4.2. Numerical assembly vectorization

The scalar version of the fully numerical scheme consists of four `for` loops corresponding to the sums in (11). During the integration a proper technique has to be chosen based on the mutual position of elements. For the sake of simplicity we only provide a listing for the case of identical elements.

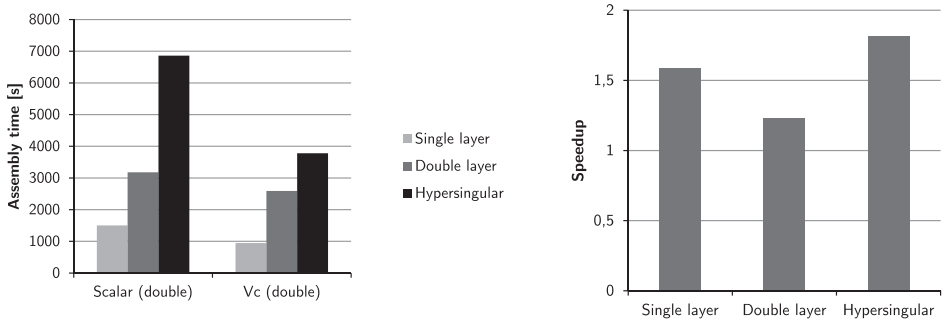
```
for (int m = 0; m < qSize1; m++)
  for (int n = 0; n < qSize2; n++)
    for (int o = 0; o < qSize3; o++)
      for (int p = 0; p < qSize4; p++)
        switch (type) {
          case identicalElements:
            for (int simplex = 0; simplex < 6;
            simplex++) {
              getQuadratureNodes(m, n, o, p, simplex,
              type, x, y, jacobian);
              getQuadratureWeights(m, n, o, p, w);
              entry+=evalSingleLayerKernel(x,
              y) * w[0] * w[1] *
                w[2] * w[3] * jacobian;
            }
            break;
          ...
        }
```

In the vectorized version the loops are unrolled and the appropriate function evaluating h in several combinations of quadrature points is provided. Moreover, the function `getQuadratureNodes` now returns coordinates of multiple quadrature points $\boldsymbol{x}, \boldsymbol{y}$ split into three `Vc::Vector` objects.

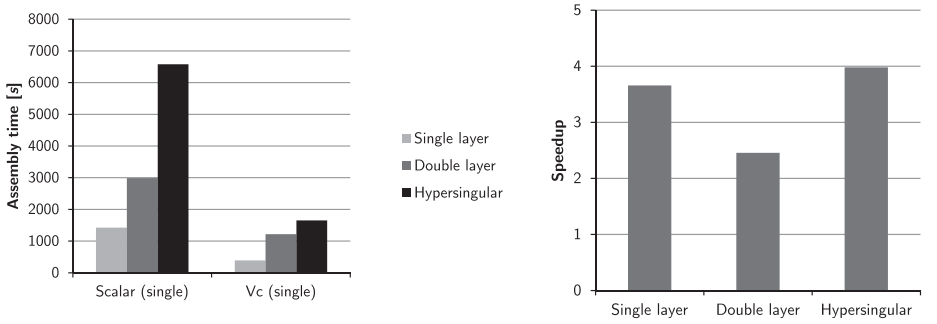
```
int qSize = qSize1 * qSize2 * qSize3 * qSize4;
for (int i = 0; i < qSize; i++)
  switch (type) {
    case identicalPanels:
      for (int simplex = 0; simplex < 6; simplex++) {
        getQuadratureNodes(i, simplex, type, x0, x1,
  x2, y0, y1, y2, jacobian);
        getQuadratureWeights(i, w0, w1, w2, w3);
        entry+=evalSingleLayerKernel(x0, x1, x2,
  y0, y1, y2) * w0 * w1 * w2 * w3 * jacobian;
      }
      break;
    ...
  }
```

4.3. Representation formula vectorization

After the computation of the Cauchy data the representation formula (2) can be used to evaluate the scattered wave u_s from (1) in an arbitrary point $\boldsymbol{x} \in \Omega^{\text{ext}}$. Since the formula is usually evaluated in a large number of nodes and each of these computations can be done independently, it is well suited for parallelization in both shared and distributed memory. In addition, the vectorized version of a function evaluating (2) in multiple nodes concurrently



(a) Double precision.



(b) Single precision.

Fig. 5. Semi-analytic assembly times for $V_{\kappa,h}$, $K_{\kappa,h}$, $D_{\kappa,h}$, scalar and vectorized versions (left) with corresponding speedup (right).

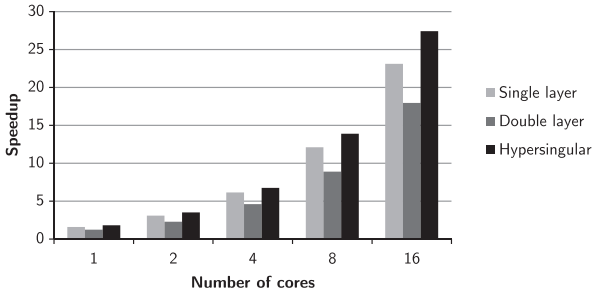


Fig. 6. Combined speedup of OpenMP and vectorization for $V_{\kappa,h}$, $K_{\kappa,h}$, $D_{\kappa,h}$ relative to the scalar sequential version (semi-analytic quadrature, double precision arithmetic).

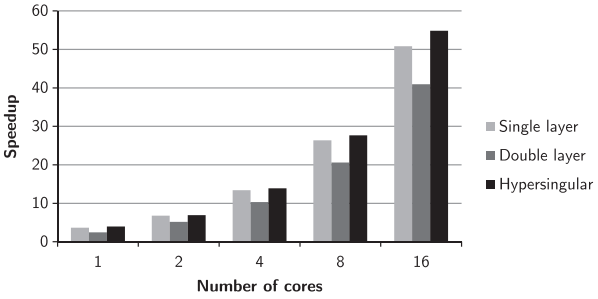


Fig. 7. Combined speedup of OpenMP and vectorization for $V_{\kappa,h}$, $K_{\kappa,h}$, $D_{\kappa,h}$ relative to the scalar sequential version (semi-analytic quadrature, single precision arithmetic).

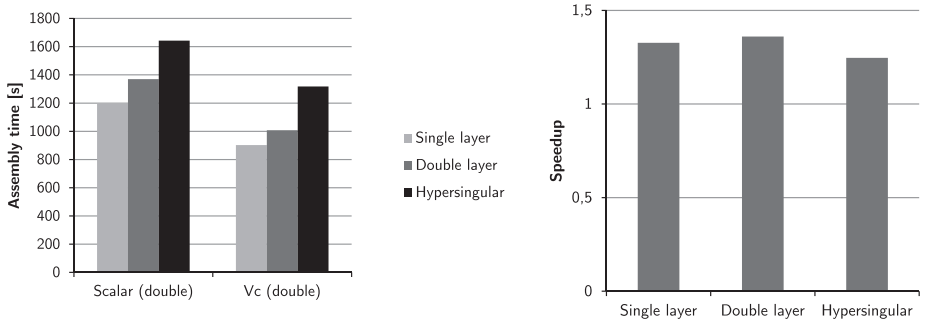
is provided. The method takes arrays of 3D coordinates as arguments and stores them in the `Vc::Memory` objects. Individual `Vc::Vector` objects are loaded from `Vc::Memory` and a numerical quadrature for (3) and (4) is performed for multiple values of $\boldsymbol{x}$. The potentials are evaluated using the analytic formulae similarly as in Section 4.1.

5. Numerical experiments

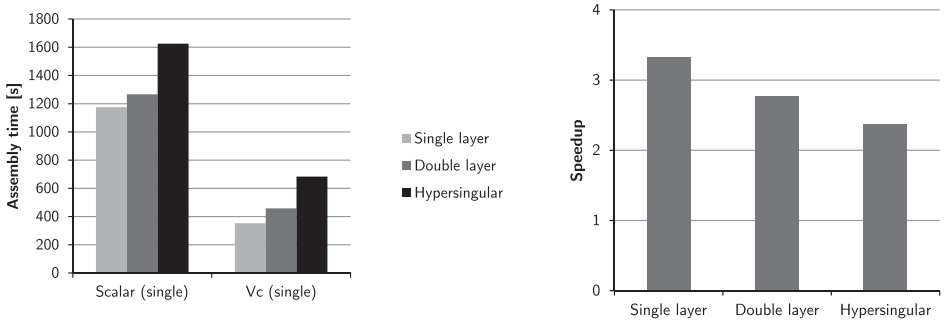
The following numerical experiments were carried out using one node of the Anselm cluster located at the IT4Innovations National Supercomputing Centre, Ostrava, Czech Republic. The node is equipped with two 8-core Intel Xeon E-2665 2.4 GHz processors and 64 GB of RAM. The processor supports the SSE4.2 and AVX instruction set extensions. The tests were performed using the GCC 4.9.0 compiler. Since the 256-bit registers are only supported by a subset of the first generation AVX instructions [18], the performance gain of AVX compared to SSE is not as significant as desired in most cases. For this reason we concentrate on the results obtained with the SSE extension.

In the first set of numerical experiments we compare the assembly times of full system matrices $V_{\kappa,h}$, $K_{\kappa,h}$, and $D_{\kappa,h}$ using the scalar and vectorized version of the library. Moreover, we provide results of the scalability test of the OpenMP code in combination with the Vc vectorization. The used computational domain Ω represents a unit ball with boundary discretized into 11,520 surface elements.

Let us begin with the semi-analytic approach. The comparison of the computational times of scalar and vectorized versions of the code is given in Fig. 5. In double precision arithmetic the speedup reaches 1.81 in the case of the hypersingular operator matrix and 1.58 and 1.23 in the cases of the single- and double-layer operator matrices, respectively. For single precision arithmetic we obtain the speedups of 3.98, 3.65, and 2.45 for the assembly of matrices $D_{\kappa,h}$, $V_{\kappa,h}$, $K_{\kappa,h}$, respectively.



(a) Double precision.



(b) Single precision.

Fig. 8. Numerical (81 quadrature points, SSE) assembly times for $V_{\kappa,h}$, $K_{\kappa,h}$, $D_{\kappa,h}$, scalar and vectorized versions (left) with corresponding speedup (right).

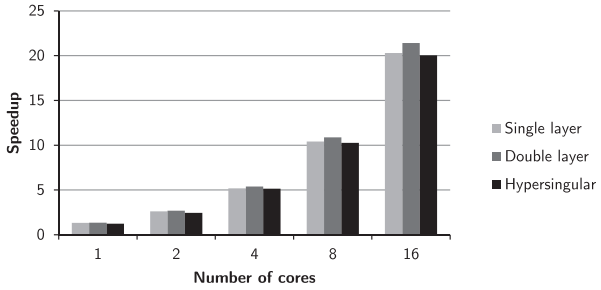


Fig. 9. Combined speedup of OpenMP and vectorization for $V_{\kappa,h}$, $K_{\kappa,h}$, $D_{\kappa,h}$ relative to the scalar sequential version (numerical quadrature, double precision arithmetic).

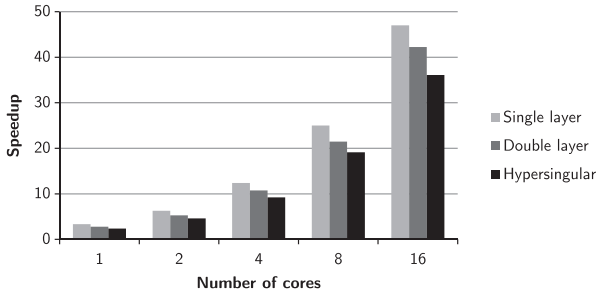


Fig. 10. Combined speedup of OpenMP and vectorization for $V_{\kappa,h}$, $K_{\kappa,h}$, $D_{\kappa,h}$ relative to the scalar sequential version (numerical quadrature, single precision arithmetic).

To further accelerate the assembly of the system matrices, the BEM4I library provides its shared memory parallelization by OpenMP. The combined speedups of the OpenMP parallelization and vectorization relative to the non-vectorized sequential version are depicted in Figs. 6 and 7. In the case of double precision computations we reach the speedup of 27.44 on 16 cores, thus reducing the computational time for the assembly of $D_{\kappa,h}$ from 6580 s on a single core without vectorization to 250 s on 16 cores with vectorization. The time is further reduced to 120 s when using single precision arithmetic. Similar results are obtained for the matrices $V_{\kappa,h}$, $K_{\kappa,h}$.

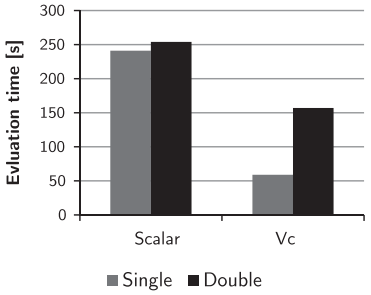
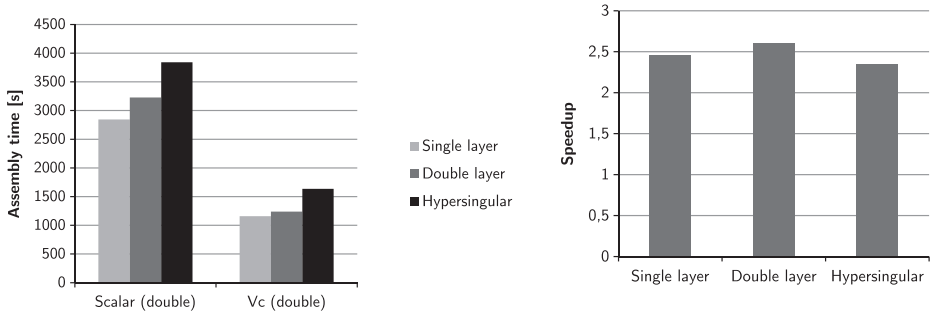


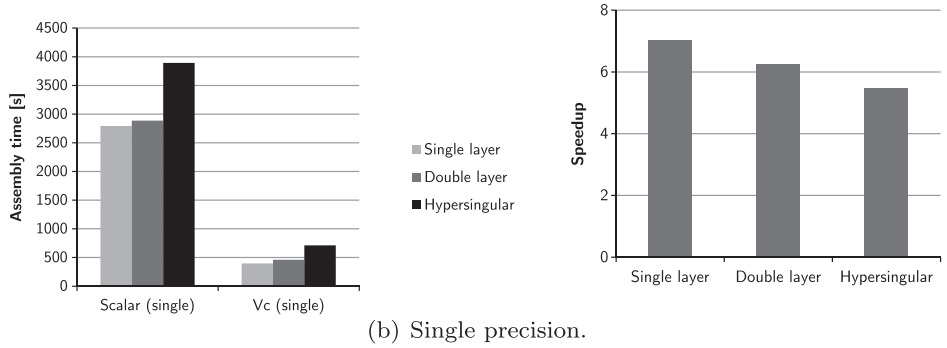
Fig. 12. Evaluation of the representation formula.

The results for the fully numerical approach are depicted in Figs. 8–11. For the experiments shown in Figs. 8–10 three quadrature points in each dimension were used ($3^4 = 81$ quadrature points in total), the AVX experiments in Fig. 11 were performed with $4^4 = 256$ quadrature points. Although the computation is not accelerated as significantly as in the case of the semi-analytic approach, the gains of vectorization are still apparent especially when using single precision arithmetic. Moreover, contrary to the semi-analytic approach, the AVX instruction set further accelerates the computation especially when requiring a higher precision quadrature, which is usually necessary for more complicated geometries (see Fig. 11).

In Fig. 12 we provide computational times for the evaluation of the representation formula (2). The vectorization approach

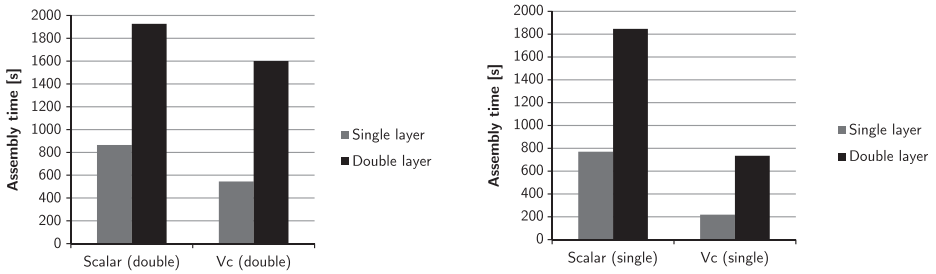


(a) Double precision.

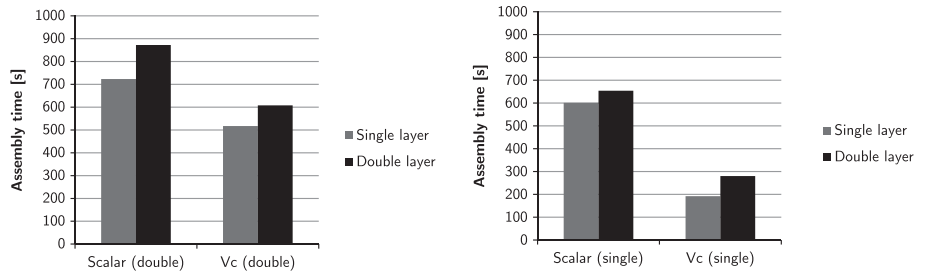


(b) Single precision.

Fig. 11. Numerical (256 quadrature points, AVX) assembly times for $V_{\kappa,h}$, $K_{\kappa,h}$, $D_{\kappa,h}$, scalar and vectorized versions (left) with corresponding speedup (right).



(a) Semi-analytic quadrature.



(b) Numerical quadrature.

Fig. 13. Setup of the nonadmissible parts of $V_{\kappa,h}$, $K_{\kappa,h}$, scalar and vectorized versions.

described above leads to a significant speedup, namely 1.62 in the case of double precision arithmetic and 4.00 in the case of single precision arithmetic. The formula was evaluated in 5,000 nodes.

Finally, the benefits of vectorization for fast BEM are demonstrated in Fig. 13. The graphs provide the computational times necessary for the assembly of nonadmissible parts of system matrices $V_{\kappa,h}$, $K_{\kappa,h}$. The assembly of these parts forms a major portion of the FMM matrices computation and takes an important amount of time when assembling ACA matrices. We observe significant speedup especially in single precision arithmetic. The tests were performed using a surface mesh consisting of 103,680 elements.

6. Conclusion

In this work we have presented a new library of parallel solvers based on the boundary element method. In addition to the shared memory parallelization by OpenMP, the library features explicit vectorization of the semi-analytic and numerical quadrature of boundary integrals. The numerical experiments have demonstrated a relatively high efficiency of the vectorized code for both quadrature strategies. As our early experiments with AVX did not prove a significant improvement of the computational times especially with the semi-analytic approach, most of the tests were performed using the SSE instruction set extension. The potential of AVX was however demonstrated on higher precision fully numerical quadrature, where the speedup reached the value of seven. Moreover, the extended length of the AVX registers in the next generation processors promises similar speedup for double precision arithmetic.

Besides the wave scattering module presented in the paper, the BEM4I library currently implements solvers for the 3D Laplace, Helmholtz, and time-dependent wave equations and a module for shape optimization based on the shape derivative approach. The solver for the time-dependent wave equation is parallelized by MPI and the scalability has been tested on the Anselm cluster up to thousands cores. Other results obtained with the BEM4I library will be discussed in future papers.

Acknowledgements

This work was supported by the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, as well as Czech Ministry of Education, Youth and Sports via the project Large Research, Development and Innovations Infrastructures (LM2011033). The work was also supported by VŠB-TU Ostrava under the Grant SGS SP2015/160.

The authors would like to express gratitude to Dr. Günther Of, TU Graz, for providing the sequential code for the analytic evaluation of the collocation integrals as described in [2].

References

[1] Sauter S, Schwab C. Boundary element methods. Springer series in computational mathematics. Springer; 2010.
[2] Rjasanow S, Steinbach O. The fast solution of boundary integral equations. Springer; 2007.
[3] Of G. Fast multipole methods and applications. In: Schanz M, Steinbach O, editors. Boundary element analysis. Lecture notes in applied and computational mechanics, vol. 29. Berlin, Heidelberg: Springer; 2007. p. 135–60. http://dx.doi.org/10.1007/978-3-540-47533-0_6.
[4] Greengard L, Rokhlin V. A fast algorithm for particle simulations. J Comput Phys 1997;135(2):280–92. http://dx.doi.org/10.1006/jcph.1997.5706.
[5] Rokhlin V. Rapid solution of integral equations of classical potential theory. J Comput Phys 1985;60(2):187–207. http://dx.doi.org/10.1016/0021-9991(85)90002-6.
[6] Bebendorf M, Rjasanow S. Adaptive low-rank approximation of collocation matrices. Computing 2003;70(1):1–24. http://dx.doi.org/10.1007/s00607-002-1469-6.
[7] Reinders J. Additional AVX-512 instructions; 2014. <https://software.intel.com/en-us/blogs/additional-avx-512-instructions>.
[8] Intel Corporation. Intel Math Kernel Library. <https://software.intel.com/en-us/intel-mkl> [accessed 16.02.15].
[9] Piparo D, Innocente V, Hauth T. Speeding up hep experiment software with a library of fast and auto-vectorisable mathematical functions. J Phys: Conf Ser 2014;513(5):052027. http://dx.doi.org/10.1088/1742-6596/513/5/052027.
[10] Kretz M, Lindenstruth V. Vc: a C++ library for explicit vectorization. Softw: Pract Exper 2012;42(11):1409–30. http://dx.doi.org/10.1002/spe.1149.

[11] Cunha M, Telles J, Ribeiro F. Streaming SIMD extensions applied to boundary element codes. Adv Eng Softw 2008;39(11):888–98. http://dx.doi.org/10.1016/j.advengsoft.2008.01.003.
[12] Iemma U. On the use of a SIMD vector extension for the fast evaluation of boundary element method coefficients. Adv Eng Softw 2010;41(3):451–63. http://dx.doi.org/10.1016/j.advengsoft.2009.10.001 [advances in optimum engineering design].
[13] Colton D, Kress R. Inverse acoustic and electromagnetic scattering theory. Applied mathematical sciences. Springer-Verlag; 1998.
[14] McLean W. Strongly elliptic systems and boundary integral equations. Cambridge University Press; 2000.
[15] Steinbach O. Numerical approximation methods for elliptic boundary value problems: finite and boundary elements. Texts in applied mathematics. Springer; 2008.
[16] Zapletal J, Bouchala J. Effective semi-analytic integration for hypersingular Galerkin boundary integral equations for the Helmholtz equation in 3d. Appl Math 2014;59(5):527–42. http://dx.doi.org/10.1007/s10492-014-0070-6.
[17] Zapletal J, Merta M. BEM4I Library. <http://industry.it4i.cz/en/products/bem4i/> [accessed 16.02.15].
[18] Hofmann J, Treibig J, Hager G, Wellein G. Comparing the performance of different x86 SIMD instruction sets for a medical imaging application on modern multi- and manycore chips. In: Proceedings of the 2014 workshop on programming models for SIMD/vector processing, WPMVP '14. New York, NY, USA: ACM; 2014. p. 57–64. http://dx.doi.org/10.1145/2568058.2568068.

Numer Algor (2015) 70:807–824
DOI 10.1007/s11075-015-9974-9

ORIGINAL PAPER

A parallel fast boundary element method using cyclic graph decompositions

Dalibor Lukáš · Petr Kovář · Tereza Kovářová · Michal Merta

Received: 3 July 2013 / Accepted: 3 February 2015 / Published online: 26 April 2015

Abstract We propose a method of a parallel distribution of densely populated matrices arising in boundary element discretizations of partial differential equations. In our method the underlying boundary element mesh consisting of n elements is decomposed into N submeshes. The related $N \times N$ submatrices are assigned to N concurrent processes to be assembled. Additionally we require each process to hold exactly one diagonal submatrix, since its assembling is typically most time consuming when applying fast boundary elements. We obtain a class of such optimal parallel distributions of the submeshes and corresponding submatrices by cyclic decompositions of undirected complete graphs. It results in a method the theoretical complexity of which is $O((n/\sqrt{N}) \log(n/\sqrt{N}))$ in terms of time for the setup, assembling, matrix action, as well as memory consumption per process. Nevertheless, numerical experiments up to $n = 2744832$ and $N = 273$ on a real-world geometry document that the method exhibits superior parallel scalability $O((n/N) \log n)$ of the overall time, while the memory consumption scales accordingly to the theoretical estimate.

Keywords Boundary element method · Parallel computing · Graph decomposition

D. Lukáš (✉) · P. Kovář · T. Kovářová · M. Merta
VŠB–Technical University of Ostrava, 17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
e-mail: dalibor.lukas@vsb.cz

P. Kovář
e-mail: petr.kovar@vsb.cz

T. Kovářová
e-mail: tereza.kovarova@vsb.cz

M. Merta
e-mail: michal.merta@vsb.cz

1 Introduction

Boundary element methods (BEM) [17] have become an efficient tool for solution of partial differential equations. When compared to more popular volume discretization techniques, BEM eliminates interior degrees of freedom and reduces the problem formulation to the boundary, which is particularly efficient in case of unbounded computational domains or in shape optimization [7, 12]. Unfortunately, when implementing BEM, we have to deal with two difficulties: an integration of singular kernels and a dense matter of system matrices.

Concerning the singular integrals, semi-analytical integration techniques have been developed [17] for many types of kernels. Here one calculates inner collocation integrals analytically while quadrature rules are employed for outer Galerkin integrals. Alternatively, the whole integration domain is decomposed into simplices, singularities are transferred to corners and then removed by Duffy's substitution [6]. The resulting regularized kernels are proved to be analytical [22] so that a Gauss quadrature converges exponentially.

As far as the density of system matrices is considered, a lot of effort has been devoted to their sparsification, which reduces the original quadratic complexity to linear or almost linear $O(n \log n)$, where n denotes the number of degrees of freedom. Such methods are then referred to as fast BEM. The idea of sparsification relies on the fact that for far field contributions to the system matrix the integral kernel is smooth and it can be replaced by low-rank approximations. This technique goes back to the fast multipole method [18] or the panel clustering [10], where low-rank function approximations were proposed and analyzed. We can also proceed in a pure algebraic manner [1] in terms of low-rank matrix approximants. Finally, several approaches have already been proposed for preconditioning [13, 16] of resulting systems so that the overall computational complexity of an iterative solution method is still $O(n \log n)$.

Yet, a parallel implementation of the boundary element method remains an issue. There are many papers, cf. [9], dealing with parallel implementations of various tree algorithms such as Burnes-Hut method [4] or the fast multipole method, but, to our best knowledge, only one paper [2] is dealing with efficient parallel assembling and action of the matrix. The general problem is an optimal assignment of leaves (jobs) of the tree to processes. The simplest but load-unbalanced method is the list scheduling, which assigns next not yet executed job to the idle process. On the other hand, a well load-balanced method is the largest process time (LPT), where jobs are sorted according to their costs. However, LPT suffers from an expensive setup and a parallely unscalable complexity of the matrix action, which is due to that the maximum number of processes sharing a row or a column, the so-called sharing constant, is $O(N)$. As a remedy Bebendorf and Kriemann in [2] propose a global sorting of elements by utilizing space-filling curves [20]. The latter in combination with sequence partitioning [15] leads to a reduction of the sharing constant to $O(\sqrt{N})$, which is in [2] numerically documented to lead to the optimal scalability $O((n/N) \log n + n/\sqrt{N})$ of the matrix assembly as well as matrix-vector product. However, the scheduling phase suffers from $O(n)$ memory consumption and $O(N(n - N))$ computational complexity for the sequence partitioning [15].

In this paper we propose a novel approach, which we prove to enjoy the parallel scalability $O((n/\sqrt{N})\log(n/\sqrt{N}))$ of the memory consumption per process, time for the setup, matrix action as well as matrix assembly, while the latter prevails. Numerical experiments exhibit the superior scalability $O((n/N)\log(n))$ of the total time. We consider a decomposition of the underlying mesh into N parts, subsequently, the system matrix is decomposed into $N \times N$ blocks. We employ N concurrent processes, to each of which we assign N blocks of the matrix to be assembled. The assignment is done in such a way that we minimize the amount of the related mesh parts, thus, the total memory consumption for storing the mesh and related structures is minimal. Additionally, in order to balance the load each process holds exactly one diagonal block since these are typically most time and memory consuming within a fast BEM. It turns out that the problem can be formulated in terms of graph theory as a decomposition of an undirected complete graph K_N into N complete subgraphs K_M, where $M(M-1)+1 = N$. This is a combinatorial complexity optimization problem. Fortunately, when one restricts to cyclic decompositions the optima are known for certain values of $N = 3, 7, 13, 21, 31$, etc. Provided that each submesh consists of n/N elements, the sharing constant takes the optimal value $M = \frac{1}{2}(1+\sqrt{4N-3})$, see (10). We base our parallel fast BEM algorithm on these graph decompositions.

The rest of the paper is organized as follows. In Section 2 we recall a boundary element discretization of the Laplace equation in 3 dimensions. In Section 3 we describe two fast BEM techniques. In Section 4 we give an introduction to cyclic decompositions of graphs and list some known optimal results, on top of which we build our parallel implementation in Section 5. In Section 6 we document the expected theoretical complexity on numerical tests using geometry of a shaft up to almost 3 millions of degrees of freedom and 273 processes. In Section 7 we conclude.

2 Boundary element methods

For simplicity we consider a 3-dimensional bounded Lipschitz domain Ω. We look for a solution $u \in H^1(\Omega)$ to the following Dirichlet boundary value problem for the Laplace operator:

$$\begin{aligned} -\triangle u &= 0 \text{ in } \Omega, \\ u &= g \text{ on } \Gamma := \partial\Omega, \end{aligned} \tag{1}$$

where $g \in H^{1/2}(\Gamma)$ denotes a prescribed Dirichlet datum. Our exposition can be easily modified to more general settings as those involving mixed boundary conditions, exterior domains as well as to other elliptic operators, e.g., Lamé, Stokes, Helmholtz, Maxwell, for which boundary integral formulations are available.

The solution of (1) can be represented as follows:

$$u(x) = \int_\Gamma \frac{\partial u}{\partial n}(y)\, G(x,y)\, dS(y) - \int_\Gamma u(y)\, \frac{\partial G}{\partial n(y)}(x,y)\, dS(y), \quad x \in \Omega, \tag{2}$$

where $G(x,y) := 1/(4\pi\|x-y\|)$ denotes the fundamental solution to the Laplace equation. The terms on the right-hand side of (2) are referred to as the single-layer

operator $\tilde{V}$ and the double-layer operator W, respectively. They can be extended continuously to linear bounded operators $\tilde{V} : H^{-1/2}(\Gamma) \to H^1(\Omega)$ and $W : H^{1/2}(\Gamma) \to H^1(\Omega)$. It remains to calculate the Neumann datum $t := \frac{\partial u}{\partial n} \in H^{-1/2}(\Gamma)$. We can apply the trace operator $\gamma_{\mathrm{D}} : H^1(\Omega) \to H^{1/2}(\Gamma)$ to both sides of (2) and we arrive at the first-kind boundary integral equation

$$u(x) = V(t)(x) - \left(-\frac{1}{2}u(x) + K(u)(x)\right),$$

where $V := \gamma_{\mathrm{D}} \circ \tilde{V} : H^{-1/2}(\Gamma) \to H^{1/2}(\Gamma)$ and $K : H^{1/2}(\Gamma) \to H^{1/2}(\Gamma)$ are linear continuous operators, additionaly, the former is $H^{-1/2}(\Gamma)$-elliptic. For $t, u \in L^\infty(\Gamma)$ these operators take the following forms:

$$V(t)(x) := \int_\Gamma t(y)G(x, y)\, dS(y), \quad K(u)(x) := \int_\Gamma u(y)\frac{\partial G}{\partial n(y)}(x, y)\, dS(y),$$

respectively, where $x \in \Gamma$ and the integrals are considered in the Lebesgue sense. Let $\langle ., . \rangle$ denote the duality pairing between $H^{-1/2}(\Gamma)$ and $H^{1/2}(\Gamma)$ with respect to the pivot space $L^2(\Gamma)$. We shall find the missing Neumann datum $t \in H^{-1/2}(\Gamma)$ by means of the following well-posed weak formulation:

$$\langle v, V(t)\rangle = \langle v, ((1/2)I + K)(g)\rangle \quad \forall v \in H^{-1/2}(\Gamma). \tag{3}$$

Now we shall approximate the solution of (3) by a boundary element method. Without a loss of generality, let Γ be polygonal and let $\tau^h = (T_i)_{i=1}^n$ be its shape-regular triangulation into n triangles. We approximate the space $H^{-1/2}(\Gamma)$ by its finite-dimensional subspace V^h consisting of the discontinuous functions that are piecewise constant on τ^h. We introduce the standard element base $(\psi_i^h(x))_{i=1}^n$ of V^h. For the sake of simplicity, we approximate the space $H^{1/2}(\Gamma)$ by V^h again. The approximation is nonconforming in sense of $V^h \not\subset H^{1/2}(\Gamma)$. Assume we are given $g^h \in V^h$ as an approximation of g. The latter introduces an additional error, which can be analyzed by means of Strang's lemma. Let us denote the coordinates of g^h in the element base by $\mathbf{g} \in \mathbb{R}^n$. The Galerkin approximation of (3) then leads to the following linear system of equations:

$$\mathbf{V}\,\mathbf{t} = ((1/2)\mathbf{M} + \mathbf{K})\,\mathbf{g}, \tag{4}$$

where $\mathbf{t} \in \mathbb{R}^n$ are the coordinates of the approximated solution $t^h(x) \in V^h$, the diagonal matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ with entries $(\mathbf{M})_{i,i} = |T_i|$, where by $|T_i|$ we denote the area of T_i, is the Galerkin approximation of the identity I, and the matrices $\mathbf{V}, \mathbf{K} \in \mathbb{R}^{n\times n}$ are the Galerkin approximations of V, K

$$(\mathbf{V})_{i,j} = \int_{T_i}\int_{T_j} G(x, y)\, dS(y)\, dS(x), \quad (\mathbf{K})_{i,j} = \int_{T_i}\int_{T_j} \frac{\partial G}{\partial n(y)}(x, y)\, dS(y)\, dS(x). \tag{5}$$

3 Fast boundary element methods

The matrices $\mathbf{V}$ and $\mathbf{K}$ are densely populated and we shall approximate them by means of hierarchical matrices [3]. For this purpose we introduce a hierarchical clustering of τ^h. On the first level, we decompose τ^h into two disjoint clusters $\mathcal{C}_1, \mathcal{C}_2$ by a separating plane. The plane crosses the boundary mass center

$$c(\tau^h) := \frac{1}{|\Gamma|} \int_\Gamma x \, dS(x) = \frac{1}{|\Gamma|} \sum_{T_k \in \tau^h} |T_k| \, c_k, \tag{6}$$

where c_k denotes the mid-point of T_k. The normal vector to the plane is an eigenvector related to the largest eigenvalue of the 3-by-3 covariance matrix

$$\begin{aligned} (\mathbf{C}(\tau^h))_{ij} &:= \sum_{T_k \in \tau^h} |T_k| \, (c_k - c(\tau^h))_i \, (c_k - c(\tau^h))_j \\ &\approx \int_\Gamma (x - c(\tau^h))_i \, (x - c(\tau^h))_j \, dS(x). \end{aligned} \tag{7}$$

In other words, the normal direction is, up to a quadrature error, the least inertia axis. Further we proceed recursively. At the next level the decomposition is applied to the cluster $\mathcal{C}_1$, i.e. the separating plane is determined by $c(\mathcal{C}_1)$ and $\mathbf{C}(\mathcal{C}_1)$ using (6) and (7), respectively, so that we arrive at $\mathcal{C}_{11}$, $\mathcal{C}_{12}$ and the decomposition of $\mathcal{C}_2$ results in $\mathcal{C}_{21}$, $\mathcal{C}_{22}$. The recursion stops when the number of triangles in a cluster to be decomposed is less than or equal to a prescribed $n_{\min}$.

The resulting binary tree of clusters generates a quad-tree of submatrices of $\mathbf{V}$. The idea of sparse approximation of $\mathbf{V}$ by a hierarchical matrix relies on the observation that submatrices of $\mathbf{V}$ related to well-separated cluster pairs can be well-approximated by a low-rank matrix. Such pairs of clusters $(\mathcal{C}_x, \mathcal{C}_y)$ are called admissible and we indicate the admissibility by the following condition:

$$\min\{\operatorname{diam} \mathcal{C}_x, \operatorname{diam} \mathcal{C}_y\} \le \eta \operatorname{dist}(\mathcal{C}_x, \mathcal{C}_y),$$

where $\eta < 1$ is given, $\operatorname{diam} \mathcal{C}$ and $\operatorname{dist}(\mathcal{C}_x, \mathcal{C}_y)$ respectively denote the diameter of $\mathcal{C}$ and the distance between $\mathcal{C}_x$ and $\mathcal{C}_y$. Unfortunately, evaluation of the condition has a quadratic complexity, therefore, we replace it by the following stronger admissibility condition with a linear complexity:

$$2 \min\{\operatorname{rad} \mathcal{C}_x, \operatorname{rad} \mathcal{C}_y\} \le \eta \left(\operatorname{dist}(c(\mathcal{C}_x), c(\mathcal{C}_y)) - \operatorname{rad} \mathcal{C}_x - \operatorname{rad} \mathcal{C}_y \right), \tag{8}$$

where $\operatorname{rad} \mathcal{C} := \max_{T_k \in \mathcal{C}} \max_{x \in T_k} \|x - c(\mathcal{C})\|$.

Based on (8) and the binary tree of clusters we decompose the matrix $\mathbf{V}$ into blocks. They are either nonadmissible, in the case a block is related to a nonadmissible pair of leaves of the cluster tree, or admissible if it is related to two vertices of the cluster tree that satisfy (8). Moreover, in order to sparsify $\mathbf{V}$ efficiently, the admissible blocks should be as large as possible, it means that the parents of the related admissible pairs of clusters creates a nonadmissible pair. The nonadmissible blocks are assembled as full matrices, however, they can be glued together and stored as a sparse matrix $\mathbf{V}^{\text{non}} \in \mathbb{R}^{n \times n}$.

3.1 Fast multipole method

We shall approximate each admissible block by a low-rank matrix. First we describe the fast multipole method (FMM) [18]. It relies on an expansion of the integral kernel $G(x, y)$ into spherical harmonics. For a point x^* with $\|x - x^*\| < \|y - x^*\|$ we have

$$G(x, y) = \frac{1}{4\pi \|y - x^*\|} \sum_{k=0}^{\infty} \left(\frac{\|x - x^*\|}{\|y - x^*\|} \right)^k P_k \left(\frac{x - x^*}{\|x - x^*\|}, \frac{y - x^*}{\|y - x^*\|} \right),$$

where the Legendre polynomials $P_k(e(x), e(y))$ admit separation of variables

$$P_k(e(x), e(y)) = \sum_{m=-k}^{k} Y_k^m(e(x))\, Y_k^{-m}(e(y)),$$

with Y_k^m being the spherical harmonic functions. We approximate $G(x, y)$ by a finite sum $G_p(x, y)$, for which we have the following error estimate:

$$\left| G(x, y) - \underbrace{\frac{1}{4\pi \|y - x^*\|} \sum_{k=0}^{p} \left(\frac{\|x - x^*\|}{\|y - x^*\|} \right)^k P_k \left(\frac{x - x^*}{\|x - x^*\|}, \frac{y - x^*}{\|y - x^*\|} \right)}_{=:G_p(x,y)} \right|$$
$$\leq \frac{1}{4\pi \ (\|y - x^*\| - \|x - x^*\|)} \left(\frac{\|x - x^*\|}{\|y - x^*\|} \right)^{p+1}.$$

Provided $\operatorname{rad} \mathcal{C}_x \leq \operatorname{rad} \mathcal{C}_y$, $x^* := c(\mathcal{C}_x)$, the error can further be estimated by

$$\frac{1}{4\pi \ \left(\|c(\mathcal{C}_x) - c(\mathcal{C}_y)\| - \operatorname{rad} \mathcal{C}_y \right)} \left[\eta \left(1 - \frac{\operatorname{rad} \mathcal{C}_x + 2\operatorname{rad} \mathcal{C}_y}{\|c(\mathcal{C}_x) - c(\mathcal{C}_y)\| + \operatorname{rad} \mathcal{C}_y} \right) \right]^{p+1}.$$

The approximation with a precision $\varepsilon > 0$ requires $p = O(\log \varepsilon / \log \eta)$ terms in the sum, each of which admits the separation of variables

$$G_p(x, y) = \frac{1}{4\pi} \sum_{k=0}^{p} \sum_{m=-k}^{k} \left(\|x - x^*\|^k\, Y_k^m(e(x)) \right) \left(\|y - x^*\|^{-k-1}\, Y_k^{-m}(e(y)) \right).$$

Therefore, the double integral can be separated into a product of two single integrals and the matrix-vector multiplication $\mathbf{s} = \mathbf{V}\,\mathbf{t}$ can be evaluated by

$$(\mathbf{s})_i = \sum_{j \in \mathrm{NF}(i)} (\mathbf{V})_{i,j}\, (\mathbf{t})_j + \sum_{k=0}^{p} \sum_{m=-n}^{n} \hat{M}_k^m(O, \psi_i) \tilde{L}_k^m(O, \mathrm{FF}(i)).$$

Here

$$\hat{M}_k^m(O, \psi_i) = \int_\Gamma \|x - x^*\|^{k-1}\, Y_k^m(e(x)) \psi_i(x) dS(x)$$

are the multipole coefficients associated with the element T_i, and

$$\tilde{L}_k^m(O, \mathrm{FF}(i)) = \sum_{j \in \mathrm{FF}(i)} \frac{(\mathbf{t})_j}{4\pi} \int_\Gamma \|y - x^*\|^{-k}\, Y_k^{-m}(e(y)) \psi_j(y) ds(y)$$

are the coefficients of local expansion. The set of admissible and nonadmissible clusters to the cluster containing the element T_i is denoted by FF(i) and NF(i), respectively. The efficient computation of $\tilde{L}_k^m(O, \mathrm{FF}(i))$ exploits the existing tree structure:

1. *Upward pass* – multipole moments are computed on the finest level of the tree and translated to the higher levels by multipole-to-multipole translations.
2. *Downward pass* – coefficients of a local expansion are computed on the highest posssible level by translation of multipole moments, and translated to the lower levels of the tree by local-to-local translations.

Since the multipole coefficients depend on the vector $\mathbf{t}$, these tree traversals have to be repeated in each iteration of an iterative solver. For details see e.g. [14].

The overall algorithmic complexity for the approximate assembling and an action of $\mathbf{V}$ is $O(p^2\, n\, \log n)$. A similar procedure is applied to the case of $\mathbf{K}$, where one additionaly differentiate each y-term in $G_p(x, y)$ subject to the normal direction $n(y)$.

3.2 Adaptive cross approximation

Another method that we describe is the adaptive cross approximation (ACA) [1]. It approximates an admissible block $\mathbf{A} := \mathbf{V}_{\mathcal{C}_x, \mathcal{C}_y} \in \mathbb{R}^{n_x \times n_y}$ by the product of low-rank matrices

$$\mathbf{A} \approx \mathbf{U}_p\, \mathbf{V}_p^T,$$

where $\mathbf{U} \in \mathbb{R}^{n_x \times p}$, $\mathbf{V} \in \mathbb{R}^{n_y \times p}$ with $p\,(n_x + n_y) < n_x\, n_y$ to guarantee a memory reduction. If the latter condition cannot be fulfilled, the block is classified as nonadmissible. The best low-rank matrix approximation in the spectral norm is the truncated singular value decomposition. However, this is impractical as the computational complexity is cubic.

Instead the ACA can be thought as a Lagrange interpolation of the matrix entries subject to properly chosen pivot rows and columns. Let $(i_1, j_1), \ldots, (i_p, j_p)$ be indices of the pivot rows and columns. By $\mathbf{P}_x \in \mathbb{R}^{n_x \times n_x}$ we denote a permutation of the unit matrix, which reorders rows in $\mathbf{A}$ so that they start with $(i_1, \ldots, i_p)$. Similarly $\mathbf{P}_y \in \mathbb{R}^{n_x \times n_x}$ denotes the column permutation of the unit matrix, which shifts the columns $(j_1, \ldots, j_p)$ of $\mathbf{A}$ forward. The ACA then approximates the admissible block $\mathbf{A}$ as follows:

$$\begin{aligned}\mathbf{P}_x\, \mathbf{A}\, \mathbf{P}_y =: \begin{pmatrix} \widetilde{\mathbf{A}}_{11} & \widetilde{\mathbf{A}}_{12} \\ \widetilde{\mathbf{A}}_{21} & \widetilde{\mathbf{A}}_{22} \end{pmatrix} &\approx \begin{pmatrix} \widetilde{\mathbf{A}}_{11} & \widetilde{\mathbf{A}}_{12} \\ \widetilde{\mathbf{A}}_{21} & \widetilde{\mathbf{A}}_{21}\, \widetilde{\mathbf{A}}_{11}^{-1}\, \widetilde{\mathbf{A}}_{12} \end{pmatrix} \\ &= \underbrace{\begin{pmatrix} \widetilde{\mathbf{A}}_{11} \\ \widetilde{\mathbf{A}}_{21} \end{pmatrix}}_{=:\mathbf{U}_p} \underbrace{\left\{ \widetilde{\mathbf{A}}_{11}^{-1}\, \left(\widetilde{\mathbf{A}}_{11}, \widetilde{\mathbf{A}}_{12} \right) \right\}}_{=:\mathbf{V}_p^T}. \end{aligned} \tag{9}$$

The ACA approximation (9) is constructed adaptively using a partial pivoting in the actual remainder $\mathbf{R}_p := \widetilde{\mathbf{A}} - \mathbf{U}_p\, \mathbf{V}_p^T$, which is the Schur complement with respect to $\widetilde{\mathbf{A}}_{22}$ completed by zero blocks. The method starts to search in the first row $i_1 := 1$ of $\mathbf{R}_0 := \widetilde{\mathbf{A}}$ for the first pivot $(\mathbf{R}_0)_{i_1, j_1}$ being the largest entry in modulus. The first

approximation is the cross of the column $\mathbf{U}_1 = \mathbf{u}_1 := (\mathbf{R}_0)_{*,j_1}$ and the row $\mathbf{V}_1 = \mathbf{v}_1 := (1/(\mathbf{R}_0)_{i_1,j_1})\,(\mathbf{R}_0)_{i_1,*}$. Then, we proceed iteratively by searching for further crosses. The second pivoting row i_2 is determined by an entry in the column j_1 of $\mathbf{R}_1$ having the maximal modulus. The pivoting column j_2 is then given by an entry with the maximal modulus in the row i_2 of $\mathbf{R}_1$. The ACA approximation is updated by the cross of the column $\mathbf{u}_2 := (\mathbf{R}_1)_{*,j_2}$ and the row $\mathbf{v}_2 := (1/(\mathbf{R}_1)_{i_2,j_2})\,(\mathbf{R}_1)_{i_2,*}$ so that $\mathbf{U}_2 := (\mathbf{U}_1, \mathbf{u}_2)$ and $\mathbf{V}_2 := (\mathbf{V}_1, \mathbf{v}_2)$. Notice that ACA does not need $\widetilde{\mathbf{A}}_{22}$. The approximation error is measured in the Frobenius norm. In [2] it is shown that, provided $\|\mathbf{R}_k\|_F \le \eta \|\mathbf{R}_{k-1}\|_F$, the stopping criterion

$$\|\mathbf{u}_p\,\mathbf{v}_p^T\|_F \le \frac{1-\eta}{1+\varepsilon}\|\mathbf{U}_{p-1}\,\mathbf{V}_{p-1}^T\|_F$$

implies the relative error reduction $\|\mathbf{R}_p\|_F \le \varepsilon\,\|\widetilde{\mathbf{A}}\|_F$. This makes the cross approximation adaptive. The complexity of the ACA approximation to $\mathbf{V}_{\mathcal{C}_x,\mathcal{C}_y}$ is again $O(p^2\,(n_x+n_y))$.

A straightforward application of ACA to $\mathbf{K}$ may fail, as in some admissible blocks $\mathbf{K}_{\mathcal{C}_x,\mathcal{C}_y}$ there might be zero parts when some pairs of triangles in $\mathcal{C}_x$ and $\mathcal{C}_y$ belong to a common plane. As a remedy, which we were advised by Professor Mario Bebendorf, we apply ACA to another matrix

$$\left(\int_{T_i^x}\int_{T_j^y}\frac{1}{|x-y|^2}\,dS(y)\,dS(x)\right)_{i,j} \in \mathbb{R}^{n_x\times n_y},$$

where T_i^x and T_j^y is the i-th triangle in $\mathcal{C}_x$ and the j-th triangle in $\mathcal{C}_y$, respectively. We remember the resulting pivot indices $(i_1, j_1), \ldots, (i_p, j_p)$. The pivots are then used to calculate the Lagrange interpolation of $1/|x-y|^3$ with the idea of separating x and y in an approximation of the kernel of $\mathbf{K}$ via a product of low-rank functions as follows:

$$\frac{\partial G}{\partial n(y)}(x,y) = \underbrace{\underbrace{(x-y)\,n(y)}_{\text{rank-4 function}}\quad \underbrace{\frac{1}{|x-y|^3}}_{\text{ACA}\rightsquigarrow\text{rank-}p\text{ function}}}_{\text{rank-}4p\text{ function}}.$$

Since $(x-y)\,n(y)$ is reproduced exactly, the modified ACA preserves the zero parts. The action of $\mathbf{K}_{\mathcal{C}_x,\mathcal{C}_y}$ is performed as follows:

$$\mathbf{K}_{\mathcal{C}_x,\mathcal{C}_y}\cdot\mathbf{g}_{\mathcal{C}_y} \approx \frac{1}{4\pi}\sum_{r=1}^{4}\mathbf{U}_p^r\cdot\left(\left(\mathbf{M}_p\right)^{-T}\cdot\left(\left(\mathbf{V}_p^r\right)^T\cdot\mathbf{g}_{\mathcal{C}_y}\right)\right),$$

where the entries of $\mathbf{M}_p \in \mathbb{R}^{p\times p}$, $\mathbf{U}_p^r \in \mathbb{R}^{m_x\times p}$, and $\mathbf{V}_p^r \in \mathbb{R}^{m_y\times p}$ reads

$$(\mathbf{M}_p)_{i,j} := \frac{1}{\left|c(T_{i_k}^x) - c(T_{j_k}^y)\right|^3},\quad (\mathbf{U}_p^r)_{i,k} := \int_{T_i^x}\frac{f^r(x)}{\left|x - c(T_{j_k}^y)\right|^3}\,dS(x),\quad \text{and}$$

$$(\mathbf{V}_p^r)_{j,k} := \int_{T_j^y}\frac{g^r(y)}{\left|c(T_{i_k}^x) - y\right|^3}\,dS(y),$$

respectively, with

$$\sum_{r=1}^{4} f^r(x)\, g^r(y) := x_1\, n_1(y) + x_2\, n_2(y) + x_3\, n_3(y) + (-1)\, y\, n(y) = (x - y)\, n(y).$$

4 Cyclic decompositions of undirected graphs

We wish to assign $N \times N$ matrix blocks to N processes such that each process is assigned exactly one diagonal block, which we expect to balance the load per process when employing a fast BEM. At the same time, we wish the maximum of the number of block row or column indices per process to be minimized, which minimizes the memory per process. We shall formulate this combinatorial problem in terms of graph theory. For certain values of N an optimal solution using a cyclic decomposition of the complete undirected graph on N vertices into complete subgraphs will be obtained. Decompositions that are not cyclic are rarely to be obtained in an easy and systematic way. Nevertheless, any decomposition of a complete graph into complete subgraphs leads to a perfect distribution of memory blocks among processes.

We recall a few notions. By a simple undirected graph G we understand a pair (V, E), where V is a set of vertices and E is a set of two element subsets of V called edges. The simple undirected graph with each pair of vertices connected by an edge is called a complete graph and denoted by K_N. We set the correspondence between the $N \times N$ block matrix and the complete graph K_N as follows. The N row (and column) indices of the matrix correspond to the N vertices of the complete graph K_N and each edge $\{w, z\} \in E(K_N)$ corresponds to a pair of off-diagonal blocks of the $N \times N$ matrix, first block with indices z, w and second block with indices w, z.

A graph can be decomposed into smaller graphs with respect to edges. By a G-decomposition of a complete graph K_N we understand such a system of pairwise edge disjoint subgraphs $G_0, G_1, \ldots, G_q$, where each G_i is isomorphic to G, that each edge of K_N belongs to exactly one copy G_i of G. A decomposition is cyclic if there exists an ordering $z_1, z_2, \ldots, z_N$ of vertices of K_N and there exist isomorphisms $\phi_i : G_0 \to G_i$, where $i = 0, 1, \ldots, q$, such that $\phi_i(z_j) = z_{j+i}$ for every $j = 1, 2, \ldots, N$, where the indices are taken modulo N. We adopt the usual convention $N = 0$ of modular arithmetic.

Among popular tools used for graph decompositions there are graph labelings. Rosa [19] proved, that there exists a cyclic G-decomposition of K_{2k+1} into $2k + 1$ copies of G with k edges if and only if G has the so called ρ-labeling of $V(G)$. Loosely saying a vertex labeling is a mapping of nonnegative integers to the vertices of a graph, while labels of each edge can be induced from the labels of its end vertices, see Fig. 1 (left). A ρ-labeling of a graph G with k edges is such an injective mapping $f : V(G) \to \{0, 1, \ldots, 2k\}$ that the set of edge labels induced by $l(z, w) = \min\{|f(w) - f(z)|, 2k + 1 - |f(w) - f(z)|\}$ is $\{1, 2, \ldots, k\}$. We refer to Fig. 1 (middle) for an example of a cyclic K_4-decomposition of K_{13} based on a ρ-labeling, which is depicted in Fig. 1 (left).

Recall that our objective is to distribute all the $N \times N$ blocks to N parallel processes each with as few indices as possible. In terms of graphs, we employ a cyclic

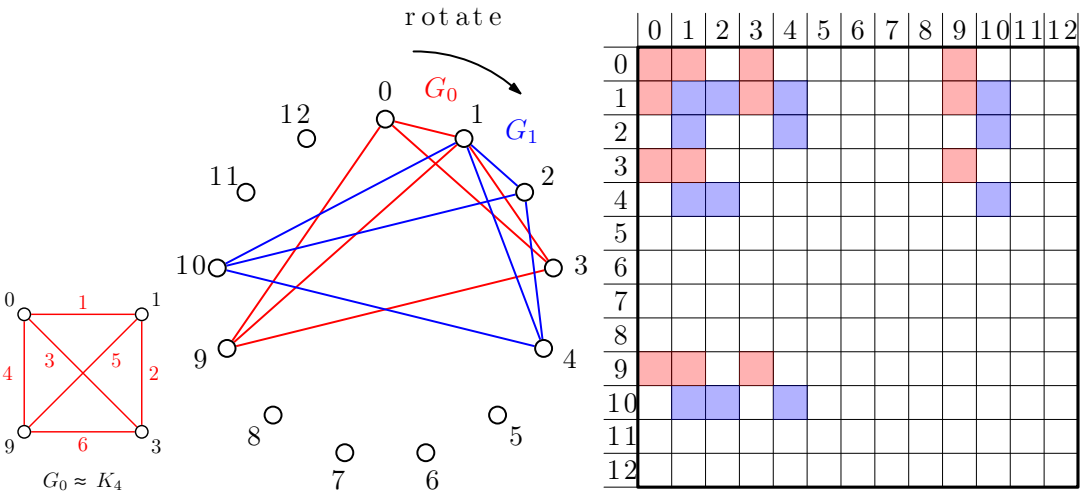


Fig. 1 Example of a ρ-labeling of K_4 with induced edge labels (*left*); a cyclic decomposition of K_{13} into 13 copies of K_4 (middle); a 13×13 block matrix with highlighted blocks corresponding to copies G_0, G_1 of K_4 with ρ-labeling $\{0, 1, 3, 9\}$ (*right*)

G-decomposition of K_N, which exists only for certain odd N, as follows. Each copy G_i, where $i = 0, 1, \ldots, N-1$, assigns to the i-th process one diagonal block and $N-1$ off-diagonal blocks with indices given by the labels of end-vertices of the edges in G_i. To minimize the total memory consumption the optimal graph G shall have $(N-1)/2$ edges and minimum vertices. Obviously, G is again a complete graph K_M with

$$\frac{M(M-1)}{2} = \frac{N-1}{2} \tag{10}$$

edges. Provided K_M has a ρ-labeling, we can construct a cyclic K_M-decomposition of K_N into $N = M^2 - M + 1$ copies of K_M. The difficulty is that a complete graph K_M allows a ρ-labeling only for certain values of M. Unfortunately, a complete classification is not known, see [5, 8]. E.g. the complete graphs K_7 and K_{11} do not have a ρ-labeling, thus no K_7-decomposition of K_{43} nor a K_{11}-decomposition of K_{111} exist. The blocks highlighted in Fig. 1 (right) correspond to the first two copies G_0 and G_1 of a K_4-decomposition of K_{13} from Fig. 1 (middle).

On the other hand, there are sufficient conditions known based on which a K_M-decomposition of K_N into N copies can be constructed for infinitely many values of N. A perfect difference set introduced by Singer in [23] with M elements corresponds immediately to a ρ-labeling of a complete graph K_M. If we restrict ourselves to the case when $M-1$ is a prime power, we can use the construction of perfect difference sets [23], see Table 1, to find a ρ-labeling of K_M and construct a cyclic K_M-decomposition of K_N.

Notice, that each copy of the complete subgraph G_i in the cyclic decomposition corresponds to blocks, that differ in their indices by $1, 2, \ldots, (N-1)/2$, while each of the differences appears precisely once. Therefore, the corresponding submeshes are likely to be distributed throughout the mesh in a sense evenly. This supports a good load balance in average.

Table 1 Selected set of labels for K_M-decompositions of K_N, where $M = \frac{1}{2}\left(1+\sqrt{4N-3}\right)$

N	M	set of labels in a ρ-labeling of K_M
3	2	{0, 1}
7	3	{0, 1, 3}
13	4	{0, 1, 3, 9}
21	5	{0, 1, 4, 14, 16}
31	6	{0, 1, 3, 8, 12, 18}
57	8	{0, 1, 3, 13, 32, 36, 43, 52}
73	9	{0, 1, 3, 7, 15, 31, 36, 54, 63}
91	10	{0, 1, 3, 9, 27, 49, 56, 61, 77, 81}
133	12	{0, 1, 3, 12, 20, 34, 38, 81, 88, 94, 104, 109}
183	14	{0, 1, 3, 16, 23, 28, 42, 76, 82, 86, 119, 137, 154, 175}
273	17	{0, 1, 3, 7, 15, 31, 63, 90, 116, 127, 136, 181, 194, 204, 233, 238, 255}

5 Parallel implementation, scalability

The idea of the parallel implementation is now straightforward. It relies on ρ-labelings, examples of which are given in Table 1. For a feasible N we decompose the mesh τ^h consisting of n triangles into N parts $\tau_0^h, \ldots, \tau_{N-1}^h$ so that the elements in a submesh τ_i^h are geometrically close and the sizes of the submeshes and the numbers of elements do not differ much from n/N. For this purpose we employ the software package Metis [11]. Each process with an index $P \in \{0, 1, \ldots, N-1\}$ then translates the set of labels $\{i_1, \ldots, i_M\}$ to the index vector of submeshes

$$\mathbf{s}^P := ((i_1 + P) \operatorname{rem} N, \ldots, (i_M + P) \operatorname{rem} N),$$

where $a \operatorname{rem} b$ gives the remainder of the division of a by b. This simple expression takes the advantage of the cyclic decomposition.

The setup phase starts so that each process P concurrently uploads the associated submeshes $\mathcal{S}_i^P := \tau^h_{(\mathbf{s}^P)_i}$ for $i = 0, \ldots, M-1$. Next, nodes and edges (pairs of nodes) on boundaries of submeshes are identified and sent to the master $P := 0$. The master process introduces a global sorting of nodal indices. The setup phase is completed by creating trees of boundary elements.

The second phase is the assembling. For $P \in \{0, 1, \ldots, N-1\}$ and $i, j \in \{0, 1, \ldots, M-1\}$ we denote by $\mathbf{V}_{i,j}^P$, $\mathbf{K}_{i,j}^P$ the blocks of the matrices $\mathbf{V}$, $\mathbf{K}$, respectively, associated to the process P and related to the submeshes $\mathcal{S}_i^P$ and $\mathcal{S}_j^P$. Similarly, we denote by $\mathbf{g}_i^P$ the block of the Dirichlet datum $\mathbf{g}$. Each P assembles the following blocks by means of a fast BEM:

- one diagonal block $\mathbf{V}_{0,0}^P$,
- $M(M-1)$ or $M(M-1)/2$ off-diagonal blocks $\mathbf{V}_{i,j}^P$, for $i \neq j$ or $i < j$, respectively, in case of FMM or ACA,
- one diagonal block $\mathbf{K}_{0,0}^P$,

- $M(M-1)$ off-diagonal blocks $\mathbf{K}^P_{i,j}$, $i \neq j$,
- and one block $\mathbf{g}^P_0$.

After the assembling phase the master process gathers the vector $\mathbf{g}$ and requests for the action $\mathbf{K} \cdot \mathbf{g}$. This is summed up from the following parallel contributions:

$$\sum_{j=0}^{M-1} \mathbf{K}^P_{0,j} \cdot \mathbf{g}^P_j \quad \text{and} \quad \sum_{\substack{j=0 \\ j\neq i}}^{M-1} \mathbf{K}^P_{i,j} \cdot \mathbf{g}^P_j, \ i = 1, \ldots, M-1,$$

and transformed by the master to the right-hand side of (4). Finally, for a solution of (4) the conjugate gradient method is employed so that in each iteration the master asks for the action $\mathbf{V} \cdot \mathbf{r}$. Each process P calculates the contributions

$$\sum_{j=0}^{M-1} \mathbf{V}^P_{0,j} \cdot \mathbf{r}^P_j \quad \text{and} \quad \sum_{\substack{j=0 \\ j\neq i}}^{M-1} \mathbf{V}^P_{i,j} \cdot \mathbf{r}^P_j, \ i = 1, \ldots, M-1,$$

while, in case of the ACA, it makes use of the symmetry of $\mathbf{V}$.

We shall estimate computational time for the setup, assembling, and matrix action as well as memory-per-process consumption. We can derive these for a slightly modified variant of the implementation, which in practice delivers still optimal, though a bit worse time and memory demands. During the assembling phase, identically to the implementation above, each process assembles the diagonal blocks $\mathbf{V}^P_{0,0}$ and $\mathbf{K}^P_{0,0}$. This enjoys the complexity $O((n/N)\log(n/N))$. Now the difference is that rather than assembling the remaining off-diagonal blocks, each process assembles only two larger hierarchical matrices $\mathbf{V}^P$ and $\mathbf{K}^P$ related to the union $\mathcal{S}^P$ of the submeshes $\mathcal{S}^P_0, \ldots, S^P_{M-1}$. The action $\mathbf{V} \cdot \mathbf{r}$ comprises the following contributions $\mathbf{v}^P$ of processes $P \in \{0, 1, \ldots, N-1\}$:

$$\mathbf{v}^P := \mathbf{V}^P \cdot \mathbf{r}^P, \quad \mathbf{v}^P_0 := \mathbf{v}^P_0 - (M-1)\,\mathbf{V}^P_{0,0} \cdot \mathbf{r}^P_0.$$

The latter subtraction is due to that each diagonal block is repeated in M matrices $\mathbf{V}^P$. The action $\mathbf{K} \cdot \mathbf{g}$ proceeds similarly. During the setup phase, we still use the same cyclic decompositions as in the implementation above. The only difference is that we construct different trees. We conclude that the theoretical complexity of the CPU-time for the setup, assembling the hierarchical matrices, a matrix action as well as the memory-per-process is

$$O\left(\frac{Mn}{N}\log\frac{Mn}{N} + \frac{n}{N}\log\frac{n}{N}\right) = O\left(\frac{n}{\sqrt{N}}\log\frac{n}{\sqrt{N}}\right). \tag{11}$$

6 Numerical results

We shall document the parallel efficiency of our method. In our experiments the theoretical estimate (11) of the memory-per-process consumption will be confirmed, while the same theoretical complexity of the setup, assembling, and matrix action

Table 2 A parallel ACA for **V** using the LPT method

	Setup time [s]; its parallel efficiency E_{CPU} [%]					
	Assembling time [s]; E_{CPU} [%]					
	Time [s] for 100 matrix actions; E_{CPU} [%]					
	Average memory per process [MB]; E_{Mem} [%]					
l	$N := 1$	3	7	13	21	31
	1;100	0;-	0;-	0;-	1;5	0;-
	8;100	3;89	1;114	0;-	0;-	0;-
0	1;100	0;-	0;-	1;8	0;-	0;-
	194;100	234;48	229;32	228;24	238;18	233;15
	4;100	3;44	2;29	2;15	2;10	2;6
	53;100	18;98	8;95	4;102	2;126	2;85
1	7;100	2;117	1;100	1;54	1;33	0;-
	332;100	304;63	279;45	271;34	273;27	271;22
	39;100	26;50	25;22	22;14	19;10	21;6
	315;100	105;100	45;100	26;93	16;94	11;92
2	38;100	13;97	6;90	3;97	2;90	2;61
	1069;100	633;97	492;82	444;67	428;55	416;46
		448;100	382;50	311;33	291;22	298;15
		520;100	224;99	112;107	76;98	54;93
3		65;100	32;87	17;88	12;77	10;63
		2111;100	1412;98	1170;87	1068;75	1011;65

will be surpassed. For a fixed number of elements n we measure the time and memory parallel efficiency, respectively, as a function of the number of processes N as follows:

$$E_{\mathrm{CPU}}(N) := \frac{N'\,\mathrm{CPU}(N')}{N\,\mathrm{CPU}(N)}\,100\ \% \quad \text{and} \quad E_{\mathrm{Mem}}(N) := \frac{\sqrt{N'}\,\mathrm{Mem}(N')}{\sqrt{N}\,\mathrm{Mem}(N)}\,100\ \%,$$

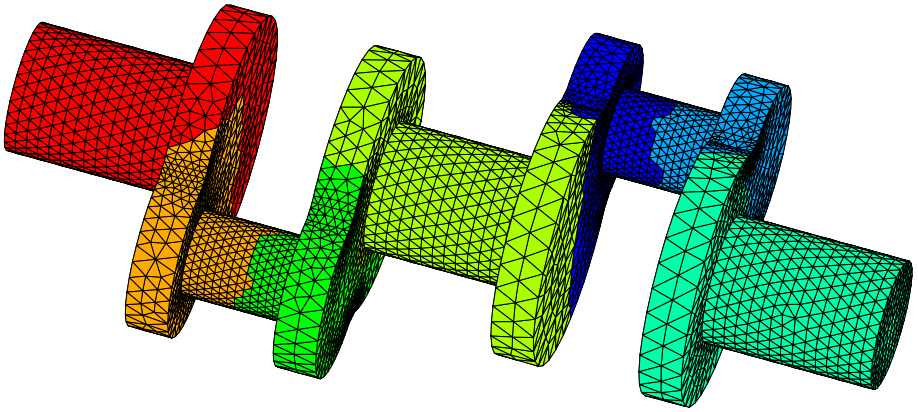


Fig. 2 Triangulation of the shaft decomposed into 7 subdomains

where N' is the smallest number of processes for which we can run the programme. Note that the efficiency can exceed 100 %, since different N lead to different trees, thus, in slightly different approximations of (4).

The numerical experiments were carried out on the cluster Anselm at VŠB-Technical University of Ostrava, Czech Republic. The cluster consists of 209 compute nodes, each equipped with two eight-core 2.4 GHz Intel Sandy Bridge

Table 3 A parallel ACA for **V** using the cyclic graph decompositions

Setup time [s]; its parallel efficiency E_{CPU} [%]
Assembling time [s]; E_{CPU} [%]
Time [s] for 100 matrix actions; E_{CPU} [%]
Time [s] for 100 matrix actions free of communication; E_{CPU} [%]
Average memory per process [MB]; E_{Mem} [%]

l	$N := 1$	3	13	21	73	133	273
	1.1;100	0.9;43	1.3;7	1.4;4	2.5;1	4.3;0	5.7;0
	106;100	35;102	7;111	4;119	1;133	1;139	0;119
0	9.9;100	4.0;82	1.2;65	0.8;59	0.3;41	0.4;21	0.4;8
	9.9;100	4.0;82	1.1;69	0.7;70	0.2;77	0.1;78	0.1;59
	503;100	321;91	237;59	235;47	228;26	229;19	230;13
	10.9;100	4.2;87	2.0;42	1.9;28	2.6;6	5.0;2	6.1;1
	551;100	192;96	43;98	26;100	6;132	7;61	2;130
1	50;100	20;81	6;61	4;58	2;41	2;19	1;14
	50;100	20;81	6;63	4;62	1;71	1;35	0;61
	1749;100	724;139	334;145	301;127	260;79	255;59	254;42
	158;100	54;97	12;103	8;98	4;50	5;22	7;9
	2381;100	855;93	204;90	123;92	29;111	15;118	8;111
2	212;100	88;80	27;61	17;58	7;44	5;33	5;16
	212;100	88;80	26;63	16;62	5;64	3;65	1;58
	6861;100	2461;161	764;249	594;252	399;201	366;182	352;118
				95;100	26;103	17;86	13;59
				465;100	123;109	67;109	38;96
3				63;100	25;72	20;51	20;24
				59;100	16;105	10;94	6;74
				1775;100	972;98	830;85	758;65
					371;100	198;103	99;100
					508;100	290;96	148;92
4					101;100	79;70	85;32
					65;100	41;87	27;64
					3365;100	2755;90	2436;71

processors and 64 GB of RAM. Compute nodes are interconnected by InfiniBand network. The theoretical peak performance is 82 Tflop/s.

In Table 2 we illustrate that a parallel method based on scheduling the jobs with respect to their costs, the so-called largest process time (LPT), suffers from the setup phase as well as from parallely unscalable memory-per-process consumption. The computational domain was $\Omega := (0, 1)^3$ discretized into $n = 3072$, 12288, 49152, and 196608 triangles at the levels $l = 0$, 1, 2, and 3, respectively. We employed the ACA method with the following parameters: $\eta := 1.1$, $\varepsilon := 10^{-4}$, and $n_{\min} := 10\,(l + 1)$ for the respective levels l. The matrix entries (5) were calculated by the second order semi-analytical quadrature method, see [17].

In all of our following experiments we discretize the boundary of the shaft depicted in Fig. 2 into $n = 10722$, 42888, 171552, 686208, and 2744832 triangles that we denote by uniform refinement levels $l = 0$, 1, 2, 3, and 4, respectively.

In Table 3 we display scalability of the parallel ACA using the cyclic decompositions applied to the matrix $\mathbf{V}$. The parameters and quadrature method remain the same as in Table 2. In contrary to the LPT parallelization, we observe that the overall time of the setup and assembling enjoys the parallel scalability $O(1/N)$ as far as the local number of elements n/N is large enough. The parallel complexity of the memory-per-process consumption follows the theoretical estimate $O(1/\sqrt{N})$ as far

Table 4 A parallel FMM for $\mathbf{V}$ using the cyclic graph decompositions

Assembling time [s]; its parallel efficiency E_{CPU} [%]
Time [s] for 100 matrix actions without communication; E_{CPU} [%]
Average memory per process [MB]; E_{Mem} [%]

l	$N := 1$	3	13	21	73	133	273
	26;100	11;78	3;60	2;63	2;67	0;67	0;60
0	53;100	26;68	10;41	8;31	6;13	5;8	5;4
	335;100	287;67	235;39	236;31	231;17	232;12	235;9
	226;100	89;85	29;59	19;57	4;74	2;76	1;66
1	131;100	67;65	30;34	24;26	18;10	16;6	15;3
	988;100	532;107	315;87	294;73	266;44	263;33	266;23
	1067;100	418;85	127;65	81;63	23;63	12;66	7;55
2	440;100	201;73	98;35	84;25	63;10	59;6	57;3
	3712;100	1603;34	654;157	544;149	424;106	391;82	390;58
				297;100	88;98	52;91	34;68
3				291;100	235;36	221;21	215;10
				1558;100	991;84	916;68	899;48
					385;100	197;103	99;100
4					890;100	855;57	837;28
					3445;100	3057;83	2959;60

as it is sufficiently larger than 200 MB, which was always consumed by the system. The parallel scalability is satisfactory up to the scalability of the matrix action, which deteoriates because the communication dominates the computation. While the computational time of the matrix action without the communication is $O(1/N)$, see the fourth lines in cells of the table, this is not the case when including the communication, see third lines, since time for the communication dominates.

Similarly to Table 3, in Table 4 we present scalability of the parallel FMM using the cyclic decompositions applied to the matrix $\mathbf{V}$. The FMM parameters were chosen similarly with the only difference that the ACA precision ε is replaced by the FMM expansion order $p := 4$. We do not present setup times as they were equal to those in Table 3. By construction only the nonadmissible blocks are assembled while assembling the admissible blocks is replaced by the FMM expansions during each matrix action. The FMM assembling times are higher than ACA since we cannot make use of the symmetry. Unfortunately, unlike ACA, the FMM action without communication behaves rather as $O(1/\sqrt{N})$ than $O(1/N)$, which might be caused by the fact that the ACA rank adapts block by block while it is globally fixed to p in FMM.

Table 5 A parallel solution to (4) using the ACA for $\mathbf{V}$, the FMM for $\mathbf{K}$, and the cyclic graph decompositions

Time [s] for assembling $\mathbf{V}$ and $\mathbf{K}$; its parallel efficiency E_{CPU} [%]
Time [s] for the CG-iterations; E_{CPU} [%]
Overall time [s]; E_{CPU} [%]
Average memory per process [MB]; E_{Mem} [%]

l	$N := 1$	7	13	31	73	133	273
	376;100	53;102	33;88	12;103	5;107	2;116	3;47
	49;100	9;74	5;69	2;69	1;54	1;39	1;14
0	**427;100**	**63;97**	**40;81**	**16;87**	**9;67**	**8;40**	**11;14**
	882;100	**301;111**	**260;94**	**243;65**	**236;44**	**237;32**	**240;22**
	4307;100	745;83	489;68	172;81	70;85	37;87	20;79
	351;100	68;74	43;62	18;63	9;53	6;44	6;23
1	**4671;100**	**816;82**	**535;67**	**192;78**	**82;78**	**48;73**	**32;53**
	4016;100	**693;219**	**465;239**	**341;211**	**293;160**	**283;123**	**283;86**
		3605;100	2224;87	875;93	391;88	257;74	184;50
		370;100	239;83	99;84	48;74	31;62	40;24
2		**3999;100**	**2478;87**	**981;92**	**445;86**	**295;71**	**236;44**
		2454;100	**1389;130**	**780;150**	**539;141**	**475;119**	**458;86**
					7229;100	1811;219	722;359
					1385;100	515;148	213;174
3					**8649;100**	**2353;202**	**957;242**
					2683;100	**1633;122**	**1293;107**

In Table 5 we show the scalability of the parallel fast BEM using the cyclic decompositions to solution of (4) by the conjugate gradient (CG) method. The right-hand side corresponds to the chosen analytical solution $u(x) := 1/|x - x^*|$, where $x^* \notin \overline{\Omega}$. We apply the ACA to the matrix $\mathbf{V}$ and the FMM to assemble the right-hand side using the matrix $\mathbf{K}$. This combination turned to be most efficient. The ACA parameters for the respective levels $l = 0, 1, 2, 3$ were chosen as follows: $\varepsilon := 10^{-8}, 10^{-8}, 10^{-8}, 10^{-10}$, $\eta := 1.1$, $n_{\min} := 10\,(l + 1)$. We employed the second order semi-analytical quadrature method. The FMM expansion orders were $p := 5, 6, 6, 7$. The entries of $\mathbf{K}$ were calculated using Sauter-Schwab quadrature method, see [21], of orders 3,4,4,4. The CG relative precision was 10^{-6}. The approximation error

$$\text{error} := \sqrt{\frac{(t - t^h, t - t^h)_{L^2(\Gamma)}}{(t, t)_{L^2(\Gamma)}}}. \tag{12}$$

took the values $3.76 \cdot 10^{-2}$, $1.71 \cdot 10^{-2}$, $9.41 \cdot 10^{-3}$, and $5.73 \cdot 10^{-3}$, which was typically achieved in 280, 350, 420, and 500 CG iterations almost independently of N at the respective levels. Note that the time for the CG-solution, which is not optimally scalable, is marginal when compared to the well-scalable time of the assembling phase.

7 Conclusion

We developed a novel method of a parallel distribution of hierarchical matrices arising in fast BEM methods, namely, in the ACA and FMM. Our method relies on a distribution of $N \times N$ matrix blocks among N processes to be assembled concurrently so that to each process exactly one diagonal block is assigned and the amount of block-row or column indices per process is minimal. This problem turns out to be equivalent to a decomposition of the complete graph K_N into complete subgraphs K_M, where N and M are related by (10). We restricted ourselves to cyclic decompositions, the constructions of which are known for an infinite number of N. Under reasonable assumptions we prove that the method enjoys the parallel scalability $O(1/\sqrt{N})$ of the memory-per-process consumption as well as the overall computational time. However, in our numerical experiments up to $n = 2744832$ boundary elements and $N = 273$ computational cores on a real-world geometry the computational time scales with $O(1/N)$, while the memory demands correspond to theory.

Our method partly relies on heuristics. A rigorous analysis of the load balance, which is documented only numerically, is missing. Another open question is the parallel scalability of the matrix action, which in our experiments deteoriates due to the fact that the communication dominates over the short computational phase.

Acknowledgments This work was supported by the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070) and by the project SPOMECH - Creating a multidisciplinary R&D team for reliable solution of mechanical problems (CZ.1.07/2.3.00/20.0070) funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, as well as Czech Ministry of Education, Youth and Sports via the

project Large Research, Development and Innovations Infrastructures (LM2011033). The work was also supported by VŠB—Technical University of Ostrava under the grant SGS SP2013/191.

References

1. Bebendorf, M.: Approximation of boundary element matrices. Numer. Math. **86**, 565–589 (2000)
2. Bebendorf, M., Kriemann, R.: Fast parallel solution of boundary integral equations and related problems. Comp. Vis. Sci. **8**, 121–135 (2005)
3. Bebendorf, M.: Hierarchical Matrices. Springer, Berlin (2008)
4. Burnes, J., Hut, P.: A hierarchical $O(N \log N)$ force calculation algorithm. Nature **324**, 446–449 (1986)
5. Colbourn, C.J., Dinitz, J.H.: The CRC Handbook of Combinatorial Designs, 2nd edn. Chapman & Hall/CRC, London (2007)
6. Duffy, M.G.: Quadrature over a pyramid or cube of integrands with a singularity at a vertex. SIAM J. Numer. Anal. **19**, 1260–1262 (1982)
7. Eppler, K., Harbrecht, H.: Second-order shape optimization using wavelet BEM. Optim. Methods Softw. **21**, 135–153 (2006)
8. Gallian, J.A.: Graph Labeling. Electron. J. Comb., Dynamic Survey **6** (2013)
9. Grama, A., Kumar, V., Same, A.: Parallel hierarchical solvers and preconditioners for boundary element methods. SIAM J. Sci. Comput. **20**, 337–358 (1998)
10. Hackbusch, W., Nowak, Z.P.: On the fast matrix multiplication in the boundary element methods by panel clustering. Numer. Math. **54**, 463–491 (1989)
11. Karypis, G., Kumar, V.: A fast and highly quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**, 359–392 (1999)
12. Lukáš, D., Postava, K., Životský, O.: A shape optimization method for nonlinear axisymmetric magnetostatics using a coupling of finite and boundary elements. Math. Comp. **82**, 1721–1731 (2012)
13. McLean, W., Tran, T.: A preconditioning strategy for boundary element Galerkin methods. Numer. Meth. Partial Differential Equations **13**, 283–301 (1997)
14. Of, G.: Fast multipole methods and applications. Lecture Notes in Applied and Computational Mechanics **29**, 135–160 (2007)
15. Olstad, B., Manne, F.: Efficient partitioning of sequences. IEEE Trans. Comp. **44**, 1322—1325 (1995)
16. von Petersdorff, T., Stephan, E.: Multigrid solvers and preconditioners for first kind integral equations. Numer. Meth. Partial Differential Equations **8**, 443–450 (1992)
17. Rjasanow, S., Steinbach, O.: The Fast Solution of Boundary Integral Equations. Springer, Berlin (2007)
18. Rokhlin, V.: Rapid solution of integral equations of classical potential theory. J. Comput. Phys. **60**, 187–207 (1985)
19. Rosa, A.: On certain valuations of the vertices of a graph. In Theory of Graphs, International Symposium, Rome, July 1966. Gordon and Breach, pp. 349–355 (1967)
20. Sagan, H.: Space-Filling Curves. Springer, Berlin (1994)
21. Sauter, S., Schwab, C.: Quadrature for hp-Galerkin BEM in $\mathbb{R}^3$. Numer. Math. **78**, 211–258 (1997)
22. Sauter, S., Schwab, C.: Boundary Element Methods. Springer, Berlin (2010)
23. Singer, J.: A theorem in finite projective geometry and some applications to number theory. Trans. AMS **43**, 377–385 (1937)

INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng (2015)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nme.5187

Efficient solution of time-domain boundary integral equations arising in sound-hard scattering

Alexander Veit[1], Michal Merta[2,3,*,†], Jan Zapletal[2,3] and Dalibor Lukáš[2,3]

[1]*Department of Computer Science, The University of Chicago, 1100 E. 58th Street, Chicago, IL 60637, USA*
[2]*IT4Innovations National Supercomputing Center, VŠB-Technical University of Ostrava, 17. listopadu 15/2172, 708 33 Ostrava, Czech Republic*
[3]*Department of Applied Mathematics, VŠB-Technical University of Ostrava, 17. listopadu 15/2172, 708 33 Ostrava, Czech Republic*

SUMMARY

We consider the efficient numerical solution of the three-dimensional wave equation with Neumann boundary conditions via time-domain boundary integral equations. A space-time Galerkin method with C^∞-smooth, compactly supported basis functions in time and piecewise polynomial basis functions in space is employed. We discuss the structure of the system matrix and its efficient parallel assembly. Different preconditioning strategies for the solution of the arising systems with block Hessenberg matrices are proposed and investigated numerically. Furthermore, a C++ implementation parallelized by OpenMP and MPI in shared and distributed memory, respectively, is presented. The code is part of the boundary element library BEM4I. Results of numerical experiments including convergence and scalability tests up to a thousand cores on a cluster are provided. The presented implementation shows good parallel scalability of the system matrix assembly. Moreover, the proposed algebraic preconditioner in combination with the FGMRES solver leads to a significant reduction of the computational time.

Received 29 March 2015; Revised 26 November 2015; Accepted 28 November 2015

KEY WORDS: variational methods; wave equation; boundary element method; time domain; retarded potential integral equation

1. INTRODUCTION

We are concerned with the efficient numerical solution of time-dependent scattering phenomena in unbounded domains. Specifically, we consider the time-dependent, three-dimensional wave equation in the case of a sound-hard scatterer modelled by Neumann boundary conditions. We formulate and solve the arising problem in terms of time-domain boundary integral equations (TDBIEs). The main advantage of this approach is that the problem (originally posed in a three-dimensional unbounded domain) is reduced to the two-dimensional (bounded) surface of the scatterer. Different discretization techniques have been introduced and intensively studied to efficiently solve TDBIEs. These methods are typically based on the Galerkin discretization in space and utilize collocation, convolution quadrature or a Galerkin scheme for the time discretization. We refer to [1–5] and the references therein for an overview of existing methods and to [6] for a detailed introduction to the theory of TDBIEs.

Here, we will employ the space-time Galerkin method to solve the arising equations numerically. This approach originated from the groundbreaking work of Bamberger and Ha Duong [7, 8]. They introduced coercive space-time variational formulations for acoustically soft and hard scatterers

*Correspondence to: Michal Merta, IT4Innovations National Supercomputing Center, VŠB-Technical University of Ostrava, 17. listopadu 15/2172, 708 33 Ostrava, Czech Republic.
†E-mail: michal.merta@vsb.cz

and showed stability and convergence of the resulting Galerkin schemes with piecewise polynomial ansatz spaces. The main difficulty in this approach is the accurate computation of the matrix entries. If standard piecewise polynomial basis functions are used, special quadrature techniques taking into account the complicated shape of the arising four-dimensional integration domains are necessary. To circumvent this problem, we use C^∞-smooth and compactly supported basis functions for the time discretization. These were introduced in [9, 10] for the Dirichlet problem, and it was shown that this choice allows an accurate approximation of the matrix entries using standard quadrature schemes. As the consequence of the simplified computation of the system matrix, the use of nonequidistant timesteps and higher-order approximation spaces in time became feasible [11]. In [9, 11], we also provide a comparison of our method to the widely used convolution quadrature method and collocation methods.

In this paper, we employ the same type of ansatz functions for the Neumann problem using the variational formulation derived in [8]. We focus on equidistant timesteps and investigate the implications on the structure of the system matrix. Because of the overlap of the temporal basis functions, the resulting linear system that needs to be solved admits a block Hessenberg structure. We compare a conventional GMRES solver with a GMRES solver preconditioned by deflations [12] and show numerically that the necessary number of iterations is significantly reduced. In Section 4.2, we furthermore introduce an experimental algebraic preconditioner that exploits the block Hessenberg structure of the system matrix. We perform various numerical experiments that show the performance of this preconditioner in combination with the FGMRES method developed in [13].

We present an efficient parallel implementation of the aforementioned discretization and solution strategies in Section 5. The code is a part of the boundary element library BEM4I [14]. It is based on C++ and uses hybrid parallelization by OpenMP and MPI to accelerate the evaluation of the discretized space-time boundary integral operators. The implementation leverages the repeating pattern of the system matrices to minimize computational cost as well as memory requirements. We briefly describe the structure of the BEM4I library and provide details about the parallel system matrix assembly in Section 5.1. Results of numerical experiments demonstrating scalability of the computation in a distributed memory architecture are provided in Section 6.

2. INTEGRAL FORMULATION OF THE WAVE EQUATION

Let $\Omega \subset \mathbb{R}^3$ be a Lipschitz domain with the boundary denoted by Γ. We consider the homogeneous wave equation

$$\partial_t^2 u - \Delta u = 0 \quad \text{in } \Omega \times [0, T] \tag{1a}$$

with homogeneous initial conditions

$$u(\cdot, 0) = \partial_t u(\cdot, 0) = 0 \quad \text{in } \Omega \tag{1b}$$

and Neumann boundary conditions

$$\partial_n u := \frac{\partial u}{\partial n} = g \quad \text{on } \Gamma \times [0, T] \tag{1c}$$

on a time interval $[0, T]$ for $T > 0$, where n denotes the unit outward normal vector. In applications, Ω is often the unbounded exterior of a bounded domain. For such problems, the method of boundary integral equations is an elegant tool where the partial differential equation is transformed to an equation on the bounded surface Γ.

We employ an ansatz as a *double layer potential* for the solution u,

$$\begin{aligned} u(x,t) := D\phi(x,t) := -\frac{1}{4\pi} \int_\Gamma \frac{n_y \cdot (x-y)}{\|x-y\|} \left(\frac{\phi(y, t-\|x-y\|)}{\|x-y\|^2} \right. \\ \left. + \frac{\partial_t \phi(y, t-\|x-y\|)}{\|x-y\|} \right) \mathrm{d}\Gamma_y, \quad (x,t) \in \Omega \backslash \Gamma \times [0,T] \end{aligned} \tag{2}$$

DOI: 10.1002/nme

with an unknown density function ϕ. D is also referred to as *retarded double layer potential* because of the retarded time argument $t - \|x - y\|$ connecting the time and space variables.

The ansatz (2) satisfies the wave equation (1a) and the initial conditions (1b). Therefore, the unknown density function ϕ has to be determined such that the Neumann boundary conditions (1c) are satisfied. For this, the normal derivative of the double layer potential has to be extended to the boundary Γ, which can be carried out continuously for sufficiently smooth functions across smooth points of Γ. We therefore define the hypersingular operator

$$Wv(x,t) := \lim_{x^+ \in \Omega \to x} n_x \cdot \nabla_{x^+} Dv(x^+, t) \tag{3}$$

for $(x,t) \in \Gamma \times [0,T]$, where the limit is taken in the sense of distributions. In order to find the unknown density function ϕ in (2) such that (1c) is satisfied, we thus consider the boundary integral equation

$$W\phi = g \quad \text{on } \Gamma \times [0,T]. \tag{4}$$

To solve this equation numerically, we introduce a weak formulation of (4) following [8]. A suitable space-time variational formulation is given by the following: Find ϕ in a Sobolev space V such that

$$\begin{aligned} a(\phi,\zeta) := \int_0^T \int_\Gamma \int_\Gamma \Bigg\{ & \frac{n_x \cdot n_y}{4\pi\|x-y\|} \ddot{\phi}(y, t-\|x-y\|) \dot{\zeta}(x,t) \\ & + \frac{\overrightarrow{\operatorname{curl}}_\Gamma \phi(y, t-\|x-y\|) \cdot \overrightarrow{\operatorname{curl}}_\Gamma \dot{\zeta}(x,t)}{4\pi\|x-y\|} \Bigg\} \, \mathrm{d}\Gamma_y \, \mathrm{d}\Gamma_x \, \mathrm{d}t \\ = & \int_0^T \int_\Gamma g(x,t) \dot{\zeta}(x,t) \, \mathrm{d}\Gamma_x \, \mathrm{d}t =: b(\zeta) \end{aligned} \tag{5}$$

for all $\zeta \in V$, where we denote by $\dot{\phi}$ and $\ddot{\phi}$ the first and second derivatives with respect to time. We refer to [8] for a precise definition of V. Here, $\overrightarrow{\operatorname{curl}}_\Gamma \phi$ is the tangential rotation of the function ϕ defined as

$$\overrightarrow{\operatorname{curl}}_\Gamma \phi(x,t) := n_x \times \nabla_x \tilde{\phi}(x,t),$$

where the prolongation $\tilde{\phi}$ is defined in a tubular neighbourhood of Γ [8, 15].

3. NUMERICAL DISCRETIZATION

We discretize the variational problem (5) using a Galerkin method in space and time. Therefore, we replace the infinite-dimensional space V by a finite-dimensional subspace V_{Galerkin} spanned by L basis functions $\{b_i\}_{i=1}^L$ in time and M basis functions $\{\varphi_j\}_{j=1}^M$ in space. This leads to the discrete ansatz

$$\phi_{\text{Galerkin}}(x,t) = \sum_{i=1}^{L} \sum_{j=1}^{M} \alpha_i^j \varphi_j(x) b_i(t), \quad (x,t) \in \Gamma \times [0,T], \tag{6}$$

with the unknown coefficients α_i^j. Plugging (6) into the variational formulation (5) and using the basis functions b_k and φ_l as test functions lead to the linear system

$$\underline{\underline{\mathbf{A}}} \cdot \underline{\boldsymbol{\alpha}} = \underline{\mathbf{g}}, \tag{7}$$

DOI: 10.1002/nme

where the block matrix $\underline{\underline{\mathbf{A}}} \in \mathbb{R}^{LM\times LM}$, the unknown coefficient vector $\underline{\boldsymbol{\alpha}} \in \mathbb{R}^{LM}$ and the right-hand side vector $\underline{\mathbf{g}} \in \mathbb{R}^{LM}$ can be partitioned according to

$$\underline{\underline{\mathbf{A}}} := \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \cdots & \mathbf{A}_{1,L} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & \cdots & \mathbf{A}_{2,L} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{L,1} & \mathbf{A}_{L,2} & \cdots & \mathbf{A}_{L,L} \end{bmatrix}, \qquad \underline{\boldsymbol{\alpha}} := \begin{bmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \\ \vdots \\ \boldsymbol{\alpha}_L \end{bmatrix}, \qquad \underline{\mathbf{g}} := \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_L \end{bmatrix}, \tag{8}$$

with

$$\mathbf{A}_{k,i} \in \mathbb{R}^{M\times M}, \quad \boldsymbol{\alpha}_i \in \mathbb{R}^{M}, \quad \mathbf{g}_k \in \mathbb{R}^{M} \quad \text{for } i,k \in \{1,\cdots,L\}.$$

Individual entries are given by

$$\begin{aligned} \mathbf{A}_{k,i}(j,l) = \int_0^T \int_\Gamma \int_\Gamma \Bigg\{ & \frac{n_x \cdot n_y}{4\pi\|x-y\|} \varphi_j(y) \ddot{b}_i(t-\|x-y\|)\, \varphi_l(x) \dot{b}_k(t) \\ & + \frac{\overrightarrow{\operatorname{curl}}_\Gamma \varphi_j(y) \cdot \overrightarrow{\operatorname{curl}}_\Gamma \varphi_l(x)}{4\pi\|x-y\|} b_i(t-\|x-y\|) \dot{b}_k(t) \Bigg\} \, \mathrm{d}\Gamma_y \, \mathrm{d}\Gamma_x \, \mathrm{d}t \end{aligned} \tag{9}$$

and

$$\boldsymbol{\alpha}_i(j) = \left(\alpha_i^j\right)_{j=1}^M, \quad \mathbf{g}_k(l) = \int_0^T \int_\Gamma g(x,t)\, \varphi_l(x)\, \dot{b}_k(t)\, \mathrm{d}\Gamma_x \, \mathrm{d}t,$$

respectively. We rewrite (9) by introducing univariate functions

$$\psi_{k,i}(r) := \int_0^T \ddot{b}_i(t-r)\dot{b}_k(t)\, \mathrm{d}t, \qquad \tilde{\psi}_{k,i}(r) := \int_0^T b_i(t-r)\dot{b}_k(t)\, \mathrm{d}t \tag{10}$$

[11] and obtain

$$\begin{aligned} \mathbf{A}_{k,i}(j,l) = \int_\Gamma \int_\Gamma \Bigg\{ & \frac{n_x \cdot n_y}{4\pi\|x-y\|} \varphi_j(y)\, \varphi_l(x) \psi_{k,i}(\|x-y\|) \\ & + \frac{\overrightarrow{\operatorname{curl}}_\Gamma \varphi_j(y) \cdot \overrightarrow{\operatorname{curl}}_\Gamma \varphi_l(x)}{4\pi\|x-y\|} \tilde{\psi}_{k,i}(\|x-y\|) \Bigg\} \, \mathrm{d}\Gamma_y \, \mathrm{d}\Gamma_x. \end{aligned} \tag{11}$$

To compute $\overrightarrow{\operatorname{curl}}_\Gamma \varphi_j$, we define the prolongation

$$\tilde{\varphi}_j(x + \varepsilon n_x) := \varphi_j(x)$$

for $x \in \Gamma$. Using this prolongation for $\varphi_j \in C(\Gamma)$ on Lipschitz domains, the $\overrightarrow{\operatorname{curl}}_\Gamma \varphi_j(x)$ is well defined almost everywhere on Γ.

The accurate computation of the matrix entries (11) is problematic in the space-time Galerkin approach. If piecewise polynomial basis functions in time are used as proposed in [8], the integrand in (11) is only piecewise smooth, which makes standard quadrature techniques prohibitively expensive. In this case, special routines (e.g. based on exact integration) have to be used to obtain accurate approximations of (11). In [9], C^∞-smooth and compactly supported temporal shape functions b_i were proposed. It could be shown [10, 11, 16] that this choice significantly simplifies the accurate approximation of integrals as in (11) and as a consequence allows the use of nonequidistant stepsizes as well as higher-order approximation spaces in time. For the sake of simplicity, we restrict

DOI: 10.1002/nme

here to the case of constant time steps, whereas the construction of fast solvers for variable time stepping is a topic of future research. We denote the timesteps by

$$t_i := i \cdot \Delta t, \quad \text{with } \Delta t := \frac{T}{N-1},\ i = 0, \dots N-1,$$

where N is the number of timesteps. In the following, we briefly recall the definition of the temporal basis functions for the special case of equidistant timesteps. We define

$$f(t) := \begin{cases} \frac{1}{2}\operatorname{erf}(2\operatorname{artanh} t) + \frac{1}{2} & |t| < 1, \\ 0 & t \leqslant -1, \\ 1 & t \geqslant 1 \end{cases}$$

and note that $f \in C^\infty(\mathbb{R})$ (see [9, 11] for a detailed analysis of this function) . We scale and shift f in order to obtain a (left) cut-off function

$$f_i(t) := f\left(2\frac{t-t_i}{\Delta t} - 1\right), \quad \text{where } f_i(t) = \begin{cases} 0 & t \leqslant t_i, \\ 1 & t \geqslant t_{i+1}. \end{cases}$$

We obtain a bump function on the interval $[t_{i-1}, t_{i+1}]$ with midpoint t_i by

$$\rho_i(t) := \begin{cases} f_{i-1}(t) & t_{i-1} \leqslant t \leqslant t_i, \\ 1 - f_i(t) & t_i \leqslant t \leqslant t_{i+1}, \\ 0 & \text{otherwise.} \end{cases}$$

A smooth partition of unity of the interval $[0, T]$ is then defined by

$$\mu_1 := 1 - f_0, \quad \mu_N := f_{N-2}, \quad \forall i \in \{2, \dots, N-1\} : \mu_i := \rho_{i-1}.$$

Smooth and compactly supported basis functions in time can now be obtained by multiplying these partition of unity functions with suitably scaled Legendre polynomials P_m of degree m (see [9] for details):

$$\begin{aligned} b_{1,m} &:= 8\mu_1(t)\left(\frac{t}{\Delta t}\right)^2 P_m\left(\frac{2}{\Delta t}t - 1\right) && m = 0, \dots, p, \\ b_{i,m} &:= \mu_i(t) P_m\left(\frac{t-t_{i-2}}{\Delta t} - 1\right) && m = 0, \dots, p,\ i = 2, \dots, N-1, \\ b_{N,m} &:= \mu_N(t) P_m\left(2\frac{t-t_{N-2}}{\Delta t} - 1\right) && m = 0, \dots, p, \end{aligned} \tag{12}$$

where p controls the order of the method in time. We will use the aforementioned basis functions in time for the Galerkin approximation (6). Note that the definition is slightly different to the one in [9]. Here, we simply use $p + 1$ basis functions in the first interval. This choice leads to a better asymptotic convergence rate of the method in the interval $[t_0, t_1]$ because we have the same number of basis functions as for the other time intervals but additionally use the a priori knowledge about the solution $u(\cdot, 0) = \partial_t u(\cdot, 0) = 0$, which is reflected in the factor t^2 in the basis functions [17]. This choice will simplify the implementation. In order to use the functions (12) as basis functions in the discrete ansatz (6), a suitable numeration has to be introduced. Here, we use

$$b_i := b_{\left\lceil \frac{i}{p+1} \right\rceil, \operatorname{mod}(i-1, p+1)} \quad \text{for} \quad i = 1, \dots, L = N \cdot p.$$

Recall that the basis functions b_i are based on the partition of unity method developed in [17]. It could be shown there that these functions have the same approximation properties as typical piecewise polynomials basis functions.

DOI: 10.1002/nme

For the discretization in space, we use standard piecewise polynomial (typically linear) basis functions φ_j. The use of higher-order methods in space is a topic of future research.

3.1. Efficient representation and evaluation of $\psi_{k,i}$ and $\tilde{\psi}_{k,i}$

An efficient handling of the functions $\psi_{k,i}$ and $\tilde{\psi}_{k,i}$ in (10) is crucial for a successful implementation of the algorithm because they have to be evaluated numerous times during the approximation of the matrix entries (11). In [11], we propose to approximate this type of functions for each $k, i \in \{1, \dots, L\}$ with piecewise Chebyshev polynomials. This approximation is efficient because of the smoothness of $\psi_{k,i}$ and $\tilde{\psi}_{k,i}$, and it can be evaluated efficiently with Clenshaw's recurrence formula [18]. Here, we furthermore exploit the fact that we only use constant timesteps. The number of approximations that have to be precomputed in this case is only $\mathcal{O}(p^2)$ compared with $\mathcal{O}(N^2 p^2)$ for variable stepsizes in time.

Let the indices $i, k \in \{1, \dots, L\}$ be arbitrary but fixed, and let $\tilde{i}, \tilde{k} \in \{1, \dots, N\}$, $m_1, m_2 \in \{0, \dots, p\}$ be such that

$$b_i \equiv b_{\tilde{i},m_1}, \quad b_k \equiv b_{\tilde{k},m_2}. \tag{13}$$

We first consider the case where $2 \leqslant \tilde{i}, \tilde{k} \leqslant N-1$, that is, basis functions that are associated with 'inner' timesteps. Then simple calculus shows that

$$\begin{aligned} \tilde{\psi}_{k,i}(r) &= \int_{t_{\tilde{k}-2}}^{t_{\tilde{k}}} b_{\tilde{i},m_1}(t-r)\dot{b}_{\tilde{k},m_2}(t)\,\mathrm{d}t \\ &= \int_{t_{\tilde{k}-2}}^{t_{\tilde{k}}} b_{2,m_1}\left(t - t_{\tilde{i}-2} - r\right)\dot{b}_{2,m_2}\left(t - t_{\tilde{k}-2}\right)\,\mathrm{d}t \\ &=: \tilde{\xi}_{m_1,m_2}\left(r + t_{\tilde{i}-2} - t_{\tilde{k}-2}\right), \end{aligned} \tag{14}$$

where

$$\tilde{\xi}_{m_1,m_2}(\alpha) := \int_0^{2\Delta t} b_{2,m_1}(t-\alpha)\dot{b}_{2,m_2}(t)\,\mathrm{d}t.$$

Thus, the task of approximating $\tilde{\psi}_{k,i}$ for $p+2 \leqslant k, i \leqslant L-p-2$ is reduced to approximating $\tilde{\xi}_{m_1,m_2}$ for $0 \leqslant m_1, m_2 \leqslant p$ on its support $[-2\Delta t, 2\Delta t]$ and evaluating these functions according to (14). Completely similarly, we obtain that

$$\psi_{k,i}(r) = \xi_{m_1,m_2}\left(r + t_{\tilde{i}-2} - t_{\tilde{k}-2}\right), \quad \text{with} \quad \xi_{m_1,m_2}(\alpha) := \int_0^{2\Delta t} \ddot{b}_{2,m_1}(t-\alpha)\dot{b}_{2,m_2}(t)\,\mathrm{d}t.$$

Figure 1 illustrates the prototype functions $\tilde{\xi}_{m_1,m_2}$ and ξ_{m_1,m_2} for $\Delta t = 1$. It suggests that these functions are either even or odd depending on m_1 and m_2. Indeed, it is easy to see that

$$\tilde{\xi}_{m_1,m_2}(\alpha) = (-1)^{m_1+m_2+1}\tilde{\xi}_{m_1,m_2}(-\alpha) \tag{15}$$

because of the symmetry of the basis functions $b_{2,m}(t)$ with respect to $t = \Delta t$. The same formula holds for ξ_{m_1,m_2}. As a consequence, approximations of ξ_{m_1,m_2} and $\tilde{\xi}_{m_1,m_2}$ (e.g. with piecewise Chebyshev polynomials) have only to be computed and stored for $\alpha \in [0, 2\Delta t]$ because

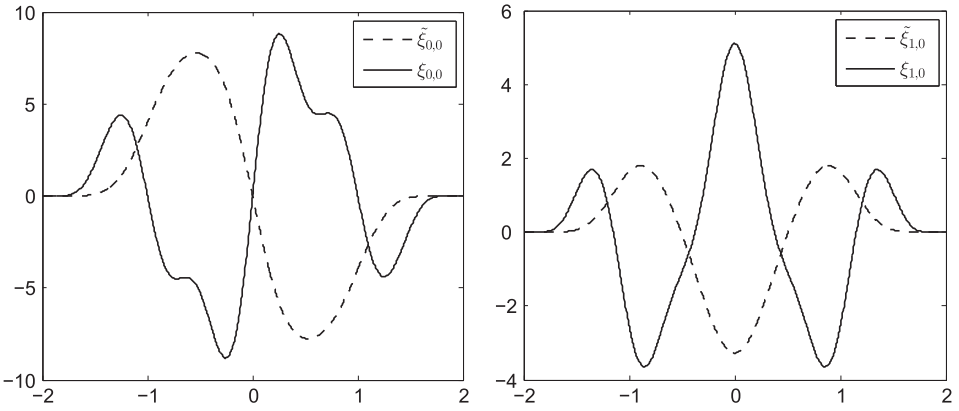
$$\tilde{\psi}_{k,i}(r) = \operatorname{sign}\left(r + t_{\tilde{i}-2} - t_{\tilde{k}-2}\right)^{m_1+m_2+1} \tilde{\xi}_{m_1,m_2}\left(\left|r + t_{\tilde{i}-2} - t_{\tilde{k}-2}\right|\right), \tag{16}$$

where $k, \tilde{k}, m_2$ and $i, \tilde{i}, m_1$ are related via (13). An analogous formula holds for $\psi_{k,i}$.

Lastly, we want to point out a symmetry with respect to m_1 and m_2. Partial integration yields

$$\tilde{\xi}_{m_1,m_2}(-\alpha) = -\tilde{\xi}_{m_2,m_1}(\alpha) \quad \text{and} \quad \xi_{m_1,m_2}(-\alpha) = -\xi_{m_2,m_1}(\alpha). \tag{17}$$

DOI: 10.1002/nme



(a) $m_1 = 0, m_2 = 0$ (b) $m_1 = 1, m_2 = 0$

Figure 1. ξ_{m_1,m_2} and $\tilde{\xi}_{m_1,m_2}$ for $\Delta t = 1$. For visualization purposes, $\tilde{\xi}_{0,0}$ and $\tilde{\xi}_{1,0}$ were multiplied by a factor 8.

Therefore, only $\frac{1}{2}(p+1)(p+2)$ functions $\tilde{\xi}_{m_1,m_2}$ and ξ_{m_1,m_2} have actually to be approximated. More importantly, this relation has an impact on the structure of the system matrix $\underline{\underline{\mathbf{A}}}$ as we will show in Section 3.2.

So far, we only considered the case $2 \leqslant \tilde{i}, \tilde{k} \leqslant N-1$. If the basis functions are associated with the first or the last timestep, similar prototype functions can be defined. Because this is analogous to the procedure described in the preceding text, we omit the details here.

3.2. Structure of the system matrix $\underline{\underline{\mathbf{A}}}$

The solution of (5) using the discrete ansatz (6) leads to a linear system with $L \cdot M$ unknowns. The special choice of basis functions in time as well as the use of equidistant timesteps has several implications on the structure of the system matrix $\underline{\underline{\mathbf{A}}}$. Throughout this section, we assume again that $k, \tilde{k}, m_2$ and $i, \tilde{i}, m_1$ are related via (13). We denote the matrix block $\mathbf{A}_{k,i}$ from (9) by $\mathbf{A}_{\tilde{k},\tilde{i}}^{m_2,m_1}$ to highlight the dependence on the timestep and the order of the involved basis functions in time. Furthermore, we define the matrix block

$$\tilde{\mathbf{A}}_{\tilde{k},\tilde{i}} := \begin{bmatrix} \mathbf{A}_{\tilde{k},\tilde{i}}^{0,0} & \cdots & \mathbf{A}_{\tilde{k},\tilde{i}}^{0,p} \\ \vdots & \ddots & \vdots \\ \mathbf{A}_{\tilde{k},\tilde{i}}^{p,0} & \cdots & \mathbf{A}_{\tilde{k},\tilde{i}}^{p,p} \end{bmatrix} \in \mathbb{R}^{(p+1)M \times (p+1)M}. \tag{18}$$

We first remark that

$$\operatorname{supp} \psi_{k,i} = \operatorname{supp} \tilde{\psi}_{k,i} = [t_{\tilde{k}-2} - t_{\tilde{i}}, t_{\tilde{k}} - t_{\tilde{i}-2}] \cap \mathbb{R}_{\geqslant 0},$$

where we formally set $t_{-1} = 0$ and $t_N = T$. Thus,

$$\psi_{k,i} \equiv \tilde{\psi}_{k,i} \equiv 0 \text{ on } \mathbb{R} \quad \text{for } t_{\tilde{k}} \leqslant t_{\tilde{i}-2}.$$

This shows that the resulting system matrix $\underline{\underline{\mathbf{A}}} \in \mathbb{R}^{LM \times LM}$ admits the block Hessenberg structure

$$\underline{\underline{\mathbf{A}}} = \begin{bmatrix} \tilde{\mathbf{A}}_{1,1} & \tilde{\mathbf{A}}_{1,2} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \tilde{\mathbf{A}}_{2,1} & \tilde{\mathbf{A}}_{2,2} & \tilde{\mathbf{A}}_{2,3} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \mathbf{0} \\ \tilde{\mathbf{A}}_{N-1,1} & \tilde{\mathbf{A}}_{N-1,2} & \cdots & \cdots & \cdots & \tilde{\mathbf{A}}_{N-1,N} \\ \tilde{\mathbf{A}}_{N,1} & \tilde{\mathbf{A}}_{N,2} & \cdots & \cdots & \cdots & \tilde{\mathbf{A}}_{N,N} \end{bmatrix}, \tag{19}$$

DOI: 10.1002/nme

where $\mathbf{0} \in \mathbb{R}^{(p+1)M \times (p+1)M}$ denotes the zero matrix. An important consequence of (16) is that for $2 \leqslant \tilde{i}, \tilde{k} \leqslant N-1$, the functions $\psi_{k,i}$ and $\tilde{\psi}_{k,i}$ only depend on the difference $t_{\tilde{i}-2} - t_{\tilde{k}-2}$ and therefore

$$\tilde{\mathbf{A}}_{\tilde{k},\tilde{i}} = \tilde{\mathbf{A}}_{\tilde{k}-\tilde{i}+2,2} \text{ for } \tilde{k} - \tilde{i} \geqslant 0 \quad \text{and} \quad \tilde{\mathbf{A}}_{\tilde{k},\tilde{i}} = \tilde{\mathbf{A}}_{2,3} \text{ for } \tilde{k} - \tilde{i} = -1. \tag{20}$$

Thus, only $\mathcal{O}(N)$ matrix blocks $\tilde{\mathbf{A}}_{\tilde{k},\tilde{i}}$ have actually to be computed and stored. In order to further reduce the complexity, we remark that for $2 \leqslant \tilde{i}, \tilde{k} \leqslant N-1$, the formula (17) together with (15) shows that

$$\mathbf{A}^{m_1,m_2}_{\tilde{k},\tilde{i}} = (-1)^{m_1+m_2} \mathbf{A}^{m_2,m_1}_{\tilde{k},\tilde{i}}. \tag{21}$$

In order to represent $\tilde{\mathbf{A}}_{\tilde{k},\tilde{i}}$, we thus only have to compute and store $\frac{1}{2}(p+1)(p+2)$ matrix blocks $\mathbf{A}^{m_2,m_1}_{\tilde{k},\tilde{i}}$. Finally, we remark that the blocks $\mathbf{A}^{m_2,m_1}_{\tilde{k},\tilde{i}}$ themselves are sparse (see also [9, 10]) because

$$\mathbf{A}^{m_2,m_1}_{\tilde{k},\tilde{i}}(j,l) = 0 \quad \text{if} \quad \operatorname{supp} \psi_{k,i} \cap [\operatorname{mindist}_{j,l}, \operatorname{maxdist}_{j,l}] = \emptyset, \tag{22}$$

where

$$\begin{aligned} \operatorname{mindist}_{j,l} &:= \min \left\{ \|x-y\|,\ x \in \operatorname{supp} \varphi_l,\ y \in \operatorname{supp} \varphi_j \right\} \\ \operatorname{maxdist}_{j,l} &:= \max \left\{ \|x-y\|,\ x \in \operatorname{supp} \varphi_l,\ y \in \operatorname{supp} \varphi_j \right\}. \end{aligned}$$

Note that in the case $T > \operatorname{diam}(\Gamma)$, (22) implies that $\mathbf{A}^{m_2,m_1}_{\tilde{k},\tilde{i}} = \mathbf{0}$ and therefore $\tilde{\mathbf{A}}_{\tilde{k},\tilde{i}} = \mathbf{0}$ if $t_{\tilde{k}-2} - t_{\tilde{i}} > \operatorname{diam}(\Gamma)$.

It is clear that a computational and memory-efficient implementation should avoid the explicit construction of $\underline{\underline{\mathbf{A}}}$. Instead, only those matrix blocks $\mathbf{A}^{m_2,m_1}_{\tilde{k},\tilde{i}}$ should be computed that are necessary to recover arbitrary entries of $\underline{\underline{\mathbf{A}}}$ via (20) and (21). Furthermore, these blocks have to be stored in a suitable sparse matrix storage format. Because the solution of the arising linear system (7) is typically computed using iterative solvers, it is also necessary to develop a routine for the efficient multiplication of $\underline{\underline{\mathbf{A}}}$ with a vector that incorporates the special structure of $\underline{\underline{\mathbf{A}}}$.

4. SOLUTION OF THE LINEAR SYSTEM

In contrast to the classical schemes using piecewise polynomial temporal basis functions with nonoverlapping supports, the resulting system matrix is not lower block triangular with block Toeplitz structure. Therefore, FFT-type methods and marching-on-time methods cannot be used for the solution of the linear system. An efficient iterative solver has to be utilized, and an appropriate preconditioner of the system has to be developed. The GMRES algorithm is one of the possible choices [19]. However, because the system matrix $\underline{\underline{\mathbf{A}}}$ is global in time and space and therefore of large dimension, the restarted version of the algorithm GMRES(m) is usually necessary to keep the computational and memory requirements reasonable. To speed up the convergence, we present two possible preconditioning techniques based on deflation and a recursive algebraic approach.

4.1. Restarted GMRES preconditioned by deflation

It is known that restarts of the algorithm lead to a slower convergence than in the case of the full GMRES because of the loss of information on the smallest Ritz values [20]. The restarted GMRES preconditioned by deflation (DGMRES(m, l)) aims to keep this information by approximating the invariant subspace corresponding to the smallest eigenvalues of the system matrix. After each restart, the invariant subspace is increased by l new vectors. It has been observed that DGMRES(m, l) can restore the convergence even in cases where the original restarted algorithm stalls [12].

DOI: 10.1002/nme

Let $|\lambda_1| \leqslant |\lambda_2| \leqslant \ldots \leqslant |\lambda_{LM}|$ be the eigenvalues of $\underline{\underline{\mathbf{A}}}$. The desired preconditioner has the form

$$\widehat{\mathbf{M}} := \mathbf{I}_{LM} + \mathbf{U}(1/|\lambda_{LM}|\mathbf{T} - \mathbf{I}_r)\mathbf{U}^T,$$

where $\mathbf{T} := \mathbf{U}^T \underline{\underline{\mathbf{A}}}\mathbf{U}$, $\mathbf{I}_{LM}$, $\mathbf{I}_r$ are identity matrices of appropriate dimensions and $\mathbf{U}$ is an orthonormal basis of the invariant subspace corresponding to the smallest r eigenvalues of $\underline{\underline{\mathbf{A}}}$. It can be proven that

$$\widehat{\mathbf{M}}^{-1} = \mathbf{I}_{LM} + \mathbf{U}(|\lambda_{LM}|\mathbf{T}^{-1} - \mathbf{I}_r)\mathbf{U}^T$$

and the eigenvalues of $\underline{\underline{\mathbf{A}}}\widehat{\mathbf{M}}^{-1}$ are $\lambda_{r+1}, \lambda_{r+2}, \ldots, \lambda_{LM}, |\lambda_{LM}|$. Thus, the smallest r eigenvalues of $\underline{\underline{\mathbf{A}}}$ are removed and replaced by $|\lambda_{LM}|$ with multiplicity r.

Because it is inefficient to assemble the preconditioner $\widehat{\mathbf{M}}$ accurately, we aim to set up its suitable approximation $\mathbf{M}$. The GMRES algorithm produces the basis $\mathbf{V}_k$ of the current Krylov subspace and the Hessenberg matrix $\mathbf{H}_k = \mathbf{V}_k^T \underline{\underline{\mathbf{A}}}\mathbf{V}_k$ representing the restriction of $\underline{\underline{\mathbf{A}}}$ onto this subspace. In order to construct $\mathbf{M}$, the Schur decomposition of $\mathbf{H}_k$ is performed. Using the Schur vectors $\mathbf{S}$ corresponding to the smallest eigenvalues of $\mathbf{H}_k$, we approximate the Schur vectors of $\underline{\underline{\mathbf{A}}}$ as $\mathbf{U} \approx \mathbf{V}_k\mathbf{S}$. Moreover, the largest eigenvalue of $\mathbf{H}_k$ is used to approximate the largest eigenvalue of $\underline{\underline{\mathbf{A}}}$.

Using this procedure, the algorithm increases $\mathbf{U}$ by l vectors after each restart until the maximal dimension r of the invariant subspace is achieved. Because the matrices $\mathbf{H}_k$ are dense and of relatively small dimensions, the Schur decomposition is performed efficiently using the BLAS routines. Moreover, it is not necessary to explicitly assemble the matrix $\mathbf{M}^{-1}$, and its actions are carried out as a sequence of dense matrix-vector multiplications. For more detailed description, we refer the reader to [12].

4.2. An algebraic preconditioner for block Hessenberg systems

In this section, we propose a preconditioner for the linear system (7) that makes use of the block Hessenberg structure of $\underline{\underline{\mathbf{A}}}$. Let

$$\underline{\underline{\mathbf{A}}}\underline{\underline{\mathbf{M}}}^{-1}(\underline{\underline{\mathbf{M}}} \cdot \underline{\boldsymbol{\alpha}}) = \underline{\mathbf{g}} \tag{23}$$

be the preconditioned system where $\underline{\underline{\mathbf{M}}}$ denotes the preconditioner. We assume that $\underline{\underline{\mathbf{A}}}$ is partitioned according to (19). We choose $\underline{\underline{\mathbf{M}}}$ to be the matrix that coincides with $\underline{\underline{\mathbf{A}}}$ except that the matrix block $\tilde{\mathbf{A}}_{\lceil N/2\rceil,\lceil N/2\rceil+1}$ is replaced by a zero matrix of the corresponding size, that is,

$$\underline{\underline{\mathbf{M}}} := \begin{bmatrix} \underline{\underline{\mathbf{M}}}_{1,1} & \underline{\underline{\mathbf{0}}} \\ \underline{\underline{\mathbf{M}}}_{2,1} & \underline{\underline{\mathbf{M}}}_{2,2} \end{bmatrix},$$

where

$$\begin{aligned} \underline{\underline{\mathbf{M}}}_{1,1} &:= \left(\tilde{\mathbf{A}}_{\tilde{k},\tilde{i}}\right)_{\tilde{k},\tilde{i}=1\ldots\lceil N/2\rceil}, \\ \underline{\underline{\mathbf{M}}}_{2,1} &:= \left(\tilde{\mathbf{A}}_{\tilde{k},\tilde{i}}\right)_{\tilde{k}=\lceil N/2+1\rceil\ldots N,\,\tilde{i}:=1\ldots\lceil N/2\rceil}, \\ \underline{\underline{\mathbf{M}}}_{2,2} &= \left(\tilde{\mathbf{A}}_{\tilde{k},\tilde{i}}\right)_{\tilde{k},\tilde{i}=\lceil N/2+1\rceil\ldots N}. \end{aligned}$$

This choice of $\underline{\underline{\mathbf{M}}}$ as a preconditioner in (23) is motivated by the Woodbury matrix identity, which states that the inverse of a rank-k perturbed matrix can be computed by doing a rank-k correction to the inverse of the original matrix. Because $\underline{\underline{\mathbf{M}}}$ is a rank-$(p+1)M$ perturbation of $\underline{\underline{\mathbf{A}}}$, it follows that $\underline{\underline{\mathbf{A}}}\underline{\underline{\mathbf{M}}}^{-1}$ is a rank-$(p+1)M$ perturbation of the identity matrix. This suggests that the application of an iterative solver like GMRES is more efficient for the preconditioned system (23) than for the original one.

DOI: 10.1002/nme

In order to apply the preconditioner within an iterative solver, $\underline{\underline{\mathbf{M}}}^{-1}$ is not needed explicitly. It is, however, crucial that the actions of $\underline{\underline{\mathbf{M}}}^{-1}$ on a vector $\underline{\mathbf{r}} = \left(\underline{\mathbf{r}}_1^T, \underline{\mathbf{r}}_2^T\right)^T$ can be computed efficiently. Because of the block-triangular structure of $\underline{\underline{\mathbf{M}}}$, it is easy to see that

$$\underline{\underline{\mathbf{M}}}^{-1}\underline{\mathbf{r}} = \begin{bmatrix} \underline{\underline{\mathbf{M}}}_{1,1}^{-1}\underline{\mathbf{r}}_1 \\ \underline{\underline{\mathbf{M}}}_{2,2}^{-1}\left(\underline{\mathbf{r}}_2 - \underline{\underline{\mathbf{M}}}_{2,1}\underline{\underline{\mathbf{M}}}_{1,1}^{-1}\underline{\mathbf{r}}_1\right) \end{bmatrix}.$$

Therefore, the problem of computing $\underline{\underline{\mathbf{M}}}^{-1}\underline{\mathbf{r}}$ boils down to the problem of evaluating $\underline{\underline{\mathbf{M}}}_{1,1}^{-1}\underline{\mathbf{v}}$ and $\underline{\underline{\mathbf{M}}}_{2,2}^{-1}\underline{\mathbf{w}}$ for some vectors $\underline{\mathbf{v}}$ and $\underline{\mathbf{w}}$. This is equivalent to the solution of linear systems of the form

$$\underline{\underline{\mathbf{M}}}_{1,1}\underline{\mathbf{x}} = \underline{\mathbf{v}} \quad \text{and} \quad \underline{\underline{\mathbf{M}}}_{2,2}\underline{\mathbf{x}} = \underline{\mathbf{w}}. \tag{24}$$

Note that $\underline{\underline{\mathbf{M}}}_{1,1}$ and $\underline{\underline{\mathbf{M}}}_{2,2}$ are again block Hessenberg matrices with basically half the size of the original matrix $\underline{\underline{\mathbf{A}}}$. Thus, these smaller systems can again be solved iteratively using a preconditioner of the same form. This strategy can be applied recursively until the matrices that have to be inverted are just the diagonal blocks $\tilde{\mathbf{A}}_{k,k}$. On the lowest level, the resulting linear systems then should be solved either exactly or with a standard iterative solver.

Algorithm 1 Solution of the system using the recursive preconditioner

Input: System matrix $\underline{\underline{\mathbf{A}}}$, RHS vector $\underline{\mathbf{g}}$, relative precision ε, max. number of iterations m_{it}
Set max. recursion depth of the preconditioner m_{r}
Set max. number of inner iterations $m_{\mathrm{it}}^{\mathrm{in}}$
Initialize the current recursion level $r := 1$
AssemblePreconditioner($\underline{\underline{\mathbf{A}}}, \underline{\underline{\mathbf{M}}}$)
FGMRES($\underline{\underline{\mathbf{A}}}, \underline{\boldsymbol{\alpha}}, \underline{\mathbf{g}}, \varepsilon, m_{\mathrm{it}}, \underline{\underline{\mathbf{M}}}$)
Output: Solution vector $\underline{\boldsymbol{\alpha}}$

function AssemblePreconditioner($\mathbf{A}$, $\mathbf{M}$)
▷ assembles the preconditioner $\mathbf{M}$ for the block Hessenberg matrix $\mathbf{A}$
$\mathbf{M}_{1,1} := (\mathbf{A}_{k,i})_{k,i=1,\ldots,\lceil N/2\rceil}$
$\mathbf{M}_{2,1} := (\mathbf{A}_{k,i})_{k=\lceil N/2+1\rceil,\ldots,N,i=1,\ldots\lceil N/2\rceil}$
$\mathbf{M}_{2,2} := (\mathbf{A}_{k,i})_{k,i=\lceil N/2+1\rceil,\ldots,N}$
$\mathbf{M} := \begin{pmatrix} \mathbf{M}_{1,1} & \mathbf{0} \\ \mathbf{M}_{2,1} & \mathbf{M}_{2,2} \end{pmatrix}$

function ApplyPreconditioner($\mathbf{M}$, $\mathbf{x}$, $\mathbf{y}$)
▷ applies the preconditioner $\mathbf{M}$ on a vector $\mathbf{x}$ and stores the result in $\mathbf{y}$
$\mathbf{M}_{1,1} := (\mathbf{M}_{k,i})_{k,i=1,\ldots,\lceil N/2\rceil}$
$\mathbf{M}_{2,1} := (\mathbf{M}_{k,i})_{k=\lceil N/2+1\rceil,\ldots,N,i=1,\ldots\lceil N/2\rceil}$
$\mathbf{M}_{2,2} := (\mathbf{M}_{k,i})_{k,i=\lceil N/2+1\rceil,\ldots,N}$
$\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2]^T, \mathbf{y} = [\mathbf{y}_1, \mathbf{y}_2]^T$
if $r \leqslant m_{\mathrm{r}}$ **then**
AssemblePreconditioner($\mathbf{M}_{1,1}$, $\mathbf{M}_1$)
AssemblePreconditioner($\mathbf{M}_{2,2}$, $\mathbf{M}_2$)
else
$\mathbf{M}_1 = \mathbf{M}_2 := \mathbf{I}$
$r := r + 1$
FGMRES($\mathbf{M}_{1,1}$, $\mathbf{y}_1$, $\mathbf{x}_1$, ε, $m_{\mathrm{it}}^{\mathrm{in}}$, $\mathbf{M}_1$)
FGMRES($\mathbf{M}_{2,2}$, $\mathbf{y}_2$, $\mathbf{x}_2 - \mathbf{M}_{2,1}\mathbf{y}_1$, ε, $m_{\mathrm{it}}^{\mathrm{in}}$, $\mathbf{M}_2$)
return $\mathbf{y} = [\mathbf{y}_1, \mathbf{y}_2]^T$

function FGMRES($\mathbf{A}$, $\mathbf{x}$, $\mathbf{y}$, ε, m_{it}, $\mathbf{M}$)
▷ solves the system $\mathbf{Ax} = \mathbf{y}$ using FGMRES algorithm with relative precision ε, maximum number of iterations m_{it}, and preconditioner $\mathbf{M}$
⋮

If the inner systems (24) and subsequent smaller systems in the recursive process are solved very accurately, the application of this preconditioner is typically too expensive, and the resulting solver is too time-consuming. Thus, we suggest to only employ a limited number of iterations of an inner solver to approximate the solution of (24). Because the classical GMRES algorithm does not allow

DOI: 10.1002/nme

the preconditioner to change at every iteration (and thus forbids the usage of an inexact solver inside the preconditioner), we use a flexible inner–outer preconditioned variant of GMRES (FGMRES) presented in [13] to solve both (7) and the systems (24). FGMRES allows a variable preconditioner at the cost of double memory requirements but with the same arithmetic complexity. Numerical experiments in Section 6 indicate that only a few iterations of the inner FGMRES solver during the application of $\underline{\underline{\mathbf{M}}}^{-1}$ are necessary to significantly reduce the number of outer iterations and the solution time.

A simplified description of the solver using the recursive preconditioner is listed in Algorithm 1. First, the maximum recursion level of the preconditioner and the number of iterations of the inner FGMRES solver are set. Next, the preconditioner is assembled, and the FGMRES is called to solve the outer system (7). During its application, the preconditioner itself employs $m_{\text{it}}^{\text{in}}$ iterations of the preconditioned FGMRES to approximate the solution of the systems (24). This repeats recursively until the maximum level of recursion is reached. On the lowest level, the inner systems are solved with FGMRES without a preconditioner.

The application of this recursive preconditioner is experimental. Although the numerical benchmarks in Section 6 show promising results, a rigorous convergence analysis is necessary. Moreover, its parallel version is still under development and has to be properly optimized.

5. IMPLEMENTATION

A parallel solver for sound-scattering problems based on the approach described in the previous sections has been implemented in the BEM4I library [14]. The code is written in C++ in an object-oriented way and utilizes OpenMP and MPI for parallelization in shared and distributed memory, respectively. All classes are templated to support various indexing and scalar types.

The structure of the solver is depicted in Figure 2. Its core consists of a set of classes responsible for the assembly of system matrices.

(1) The `BESpace` class holds the information about the order of spatial and temporal basis and test functions. It also stores the object of the underlying surface mesh.
(2) The purpose of the `BEBilinearFormWave` class is to assemble appropriate system matrices. Several matrix types are supported, including non-distributed sparse matrices (suitable for small problems solved by a direct solver), non-distributed block sparse matrices and block sparse matrices distributed among computational nodes using MPI. The usage of block system matrices reduces memory and computational requirements because it does not duplicate assembly and storage of identical blocks. The assembly of individual blocks is parallelized by OpenMP.
(3) The `BEIntegratorWave` class is responsible for the assembly of local system matrices, that is, the quadrature over pairs of elements. It takes care of evaluation of smooth temporal basis and testing functions and their Chebyshev approximations.

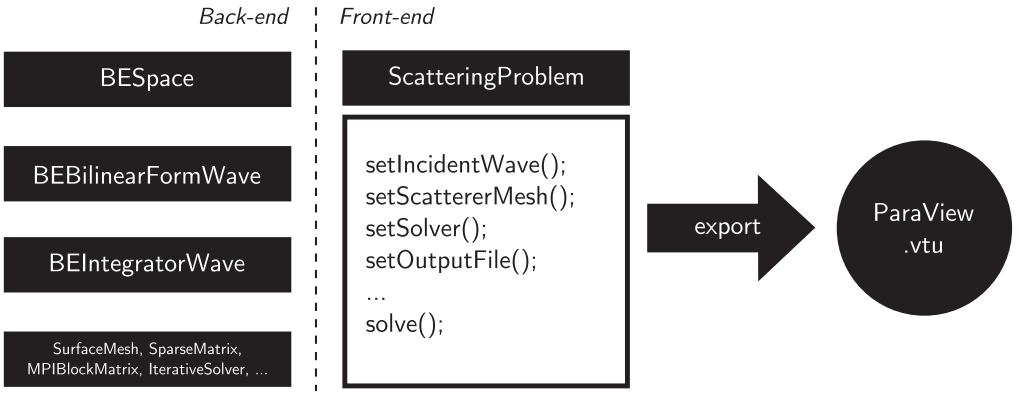


Figure 2. Structure of the solver for wave scattering problems in the BEM4I library.

DOI: 10.1002/nme

Besides these main classes, the solver utilizes classes representing surface meshes, full, sparse and block matrices, direct and iterative solvers (including GMRES, DGMRES and FGMRES), classes for the evaluation of potential operators, and so on. A user solves a problem either by direct manipulation of the aforementioned classes or using the interface provided in the `ScatteringProblem` class. The solution is exported to the ParaView `.vtu` file format.

5.1. Assembly of distributed system matrix

Special care has to be taken during the parallel assembly of the system matrix to exploit its structure and minimize memory and computational requirements. One has to take into account its properties described in Section 3.2, that is, the block Hessenberg format (19), duplication of blocks following from (20), sparsity and the special structure of individual blocks (21), as well as the fact that $\tilde{\mathbf{A}}_{\tilde{k},\tilde{i}} = \mathbf{0}$ for $t_{\tilde{k}-2,\tilde{i}} - t_{\tilde{i}} > \operatorname{diam}(\Gamma)$. In the following, we describe the assembly of a distributed block system matrix. Similar principles can be applied to assemble a non-distributed block matrix suitable for problems of smaller dimension.

To represent a system matrix, we use the `MPIBlockMatrix` class of the BEM4I library. The class serves as a simple distributed container for non-distributed matrices, such as `FullMatrix`, `SparseMatrix` or any linear operator implementing the `Apply` method on a vector. When an instance of the class is created in the MPI environment, each MPI process is only assigned a subset of matrix blocks. The matrix-vector multiplication is performed in parallel, and the local results are gathered, summed and distributed among all processes. The multiplied vector is replicated on every process (Figure 3).

In the first step of the matrix assembly, the distribution of non-zero blocks $\tilde{\mathbf{A}}_{\tilde{k},\tilde{i}}$ among P available MPI processes is determined. Assuming as in the preceding text that the time interval $[0, T]$ is discretized into equidistant timesteps of the length $\Delta t := T/(N-1)$, the number of inner non-zero blocks $\tilde{\mathbf{A}}_{\tilde{k},\tilde{i}}, \tilde{k}, \tilde{i} \in \{2, \dots, N-1\}$ can be obtained as

$$n_z := \frac{1}{2}(N^2 - N - 4) - \frac{N - \tilde{n}_z - 2}{2} \max\{N - \tilde{n}_z - 1, 0\}, \tag{25}$$

where

$$\tilde{n}_z := \min\left\{N, \left\lceil \frac{2 \cdot \operatorname{diam}(\Gamma)}{\Delta t} \right\rceil + 2\right\} \tag{26}$$

is the number of non-zero blocks in the first matrix column. Each MPI process is assigned approximately n_z/P inner non-zero blocks. The exact distribution of blocks among processes is defined by a mapping $R : \mathbb{N} \times \mathbb{N} \to \mathbb{N}_0$ assigning the MPI rank of the owner to every block. Two factors have to be considered when defining R – proper load balance during the matrix-vector multiplication and duplication of blocks $\tilde{\mathbf{A}}_{\tilde{k},2}, \tilde{k} \in \{2, \dots, N-2\}$ and $\tilde{\mathbf{A}}_{2,3}$ due to the relation (20). Therefore, the distribution is performed diagonal-wise in order to store identical blocks on the same process in one memory location. However, some blocks are duplicated among several processes to ensure balanced load during the matrix-vector multiplication. An example of the distribution of inner blocks among processes is depicted in Figure 4. After the assignment of inner blocks, the blocks in the first and last rows and columns are mapped to processes with the smallest number of blocks assigned. Note that this distribution does not take into account differences in fill-in of various blocks.

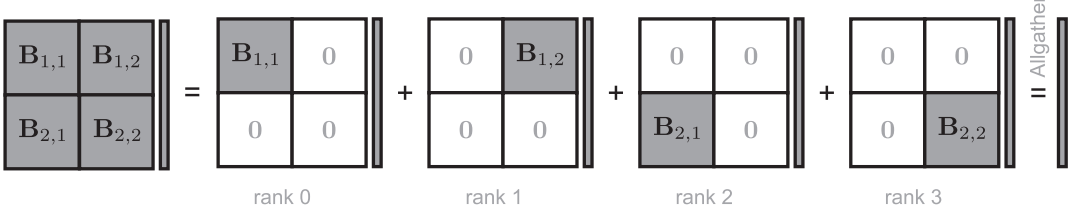


Figure 3. Multiplication by a distributed matrix `MPIBlockMatrix`.

DOI: 10.1002/nme

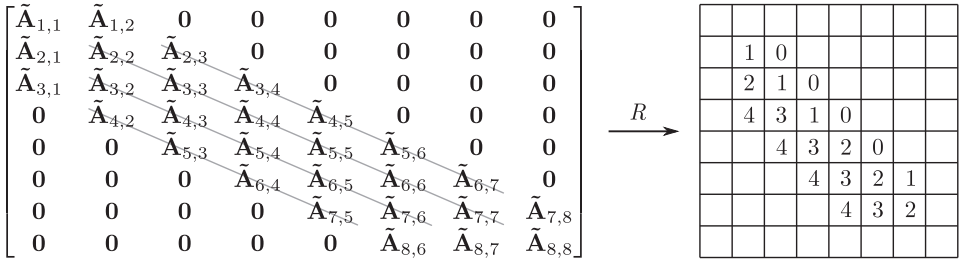


Figure 4. Example of a system matrix and distribution of its inner blocks $\tilde{\mathbf{A}}_{2,2}, \ldots, \tilde{\mathbf{A}}_{N-1,N-1}$ among five processes with ranks $\{0, 1, \ldots, 4\}$. Identical blocks are connected by a line (here $N = 8, \tilde{n}_z = 3$).

After the mapping R is defined, the blocks $\tilde{\mathbf{A}}_{\tilde{k},\tilde{i}}$ are assembled. Because of the relation (20), the total number of blocks to be assembled is at most $3N$ (the first two columns, the last row and the blocks $\tilde{\mathbf{A}}_{2,3}$ and $\tilde{\mathbf{A}}_{N-1,N}$); therefore, the complexity is reduced from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$. Each unique non-zero block is assembled and distributed in the following way:

(1) An instance of the `SparseMatrix` class representing the current block $\tilde{\mathbf{A}}_{\tilde{k},\tilde{l}}$ is created.
(2) Approximations of temporal basis functions associated with the current block are computed using the relations from [11] and Section 3.1.
(3) Using (22), the non-zero pattern of the current block $\tilde{\mathbf{A}}_{k,l}$ is precomputed together with pairs of mesh elements contributing to the non-zero entries. This computation is parallelized by MPI and OpenMP.
(4) The pairs of elements are evenly split among available MPI processes.
(5) Every MPI process iterates through its element pairs and through the order of the temporal basis functions and computes its local contribution to the block $\tilde{\mathbf{A}}_{\tilde{k},\tilde{i}}$ using (11). In order to further save computational time and memory property, (21) can be employed. The iteration through element pairs is parallelized in shared memory by OpenMP.
(6) Each process sends its partial results to the owner(s) of the block determined by the mapping R. The owner combines the results to form the current block. If any identical block is owned by the process, only a reference to the assembled block is copied to the appropriate memory location, and the data are not duplicated.
(7) All other processes delete their partial results.

6. NUMERICAL EXPERIMENTS

In the following section, we present results of numerical experiments performed on the Anselm cluster located at the IT4Innovations National Supercomputing Centre, Czech Republic. The cluster consists of 188 compute nodes, each of them equipped with two 8-core Intel Xeon E5-2665 processors and 64 GB of RAM. The theoretical peak performance of the cluster is 82 Tflop/s.

6.1. Convergence tests

In this section, we test the convergence of the space-time Galerkin method introduced in the preceding text. In [21], analytic solutions of direct and indirect formulations for Dirichlet and Neumann problems on the unit sphere were derived. Such solutions turned out to be very useful for numerical experiments and the validation of the algorithm and also serve here as reference solutions. We recall these solutions for the problem (4) for the sake of completeness.

Let us assume that the scatterer is the unit sphere, that is, $\Gamma = \mathbb{S}^2$, and that the right-hand side in (4) is of the form

$$g(x,t) = g(t)Y_n^m(x) \quad \text{for } (x,t) \in \mathbb{S}^2 \times [0,T], \tag{27}$$

DOI: 10.1002/nme

where $Y_n^m(x)$ are spherical harmonics of degree n and order m. We assume $g(t)$ to be causal, that is, $g(t) = 0$ for $t \leqslant 0$ and that $\dot{g}(0) = 0$. It can be shown [15] that $Y_n^m(x)$ are eigenfunctions of the hypersingular operator in the frequency domain, that is,

$$\mathcal{L}(W)(s)Y_n^m(x) = \lambda_n(s)Y_n^m(x),$$

where $\mathcal{L}$ denotes the Laplace transform. The function $\lambda_n(s)$ can be expressed explicitly in terms of spherical Bessel functions and spherical Hankel functions of the first kind. Assuming that the solution of (4) for the special right-hand side (27) also admits the form $\phi(t)Y_n^m(x)$ and exploiting the fact that W is a convolution with respect to the time variable, we obtain

$$\phi(t) = \int_0^t g(\tau)\mathcal{L}^{-1}\left\{\frac{1}{\lambda_n}\right\}(t-\tau)\,\mathrm{d}\tau. \tag{28}$$

Evaluating the inverse Laplace transform in the formula in the preceding text leads to explicit formulae for ϕ. In the case $n = 0$, that is, the right-hand side $g(x,t) = g(t)$ is purely time dependent, the solution of (4) is purely time dependent as well and is given by

$$\begin{aligned}\phi(x,t) = &-2\int_0^t g(t-\tau)\cosh(\tau)\,\mathrm{d}\tau \\ &+ 2\sum_{k=1}^{\lfloor t/2\rfloor}\sum_{l=1}^{k}(-1)^{k+1}\int_{2k}^t c_{k,l}\,(\tau-2k)^{k-l+1}\,\mathrm{e}^{\tau-2k}\,\dot{g}(t-\tau)\,\mathrm{d}\tau,\end{aligned}$$

where

$$c_{k,l} := \binom{k-1}{l-1}\frac{2^{k-l}}{(k-l+1)!}.$$

In the case $n = 1$, that is, the right-hand side is of the form $g(x,t) = g(t)Y_1^m(x)$, the solution of (4) is given by

$$\phi(x,t) = \left(-2\int_0^t g(t-\tau)\cosh(\tau)\cos(\tau)\,\mathrm{d}\tau\right)Y_1^m(x)$$

for $t \in [0,2[$. Formulae for large n and t are typically complicated and furthermore difficult to evaluate because of numerical cancellation. In this case, the one-dimensional problem

$$\int_0^t \mathcal{L}^{-1}\{\lambda_n\}(t-\tau)\phi(\tau)\,\mathrm{d}\tau = g(t)$$

can be accurately and efficiently solved using the convolution quadrature in order to obtain approximations of (28) (e.g. [2]).

In the following set of numerical experiments, we test the convergence of the method using the reference solutions introduced in the preceding text. First, we solve (4) on the unit sphere $\mathbb{S}^2$ for $t \in [0,6]$ with the purely time-dependent right-hand side of the form

$$g(x,t) := \sin(3t)t^2\,\mathrm{e}^{-t}\,Y_0^0(x).$$

Note that Y_0^0 is constant on the unit sphere. The error of the numerical solution is measured in the $\mathrm{L}^2(\Gamma;[0,6])$ norm. Using the local (temporal) polynomial approximation space of order $p = 1$, we obtain the convergence rate of N^{-1} (Figure 5); the convergence for $p = 2$ is depicted in Figure 6.

For the second numerical experiment, we consider the right-hand side

$$g(x,t) := \sin(2\pi t)t^3\,\mathrm{e}^{-2t}\,Y_1^0(x)$$

DOI: 10.1002/nme

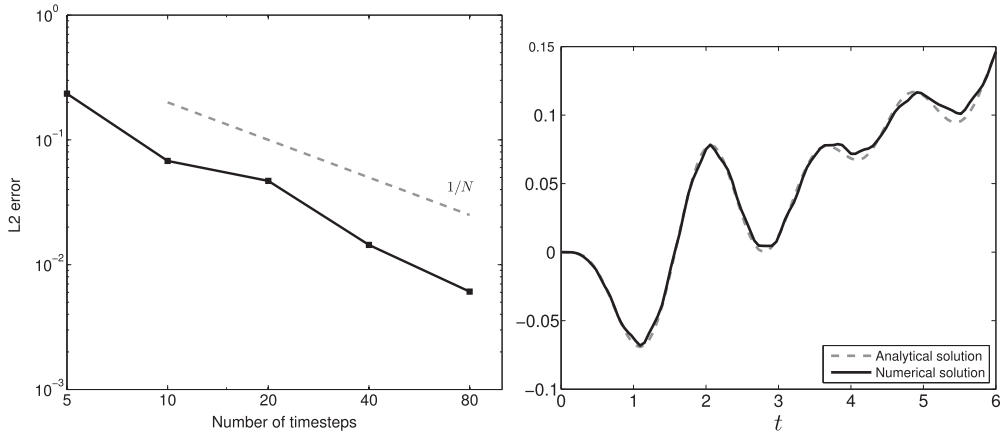


Figure 5. Left: log-log scale plot of $\|\Phi - \Phi_{\text{Galerkin}}\|_{L^2(\Gamma;[0,T])}$ for $g(t) = \sin(3t)t^2\,\mathrm{e}^{-t}, n = 0, T = 6$, and local polynomial approximation space of degree $p = 1$. Right: numerical solution in $x \in \Gamma$ obtained using 20 timesteps and $p = 1$.

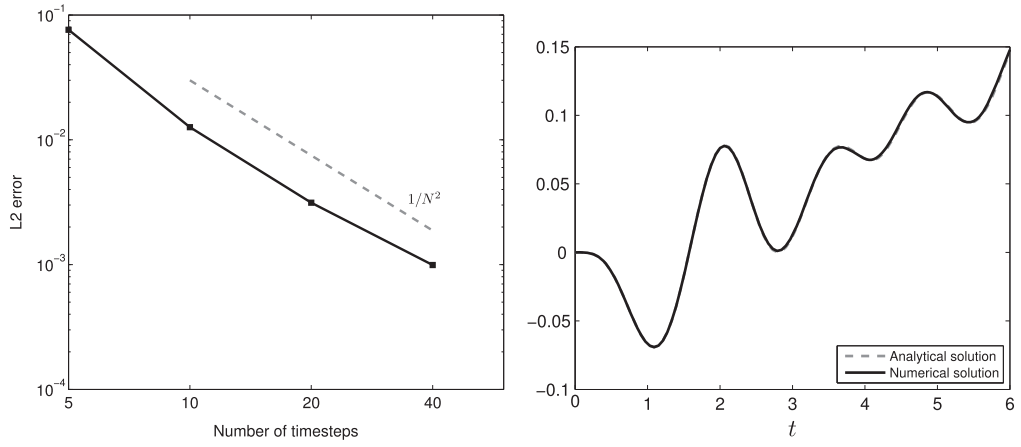


Figure 6. Left: log-log scale plot of $\|\Phi - \Phi_{\text{Galerkin}}\|_{L^2(\Gamma;[0,T])}$ for $g(t) = \sin(3t)t^2\,\mathrm{e}^{-t}, n = 0, T = 6$, and local polynomial approximation space of degree $p = 2$. Right: numerical solution in $x \in \Gamma$ obtained using 20 timesteps and $p = 2$.

and solve (4) with $\Gamma := \mathbb{S}^2$ and $t \in [0, 2]$. The numerical solution evaluated in the point $\Gamma \ni x = (0, 0, 1)$ obtained using the temporal approximation space of order $p = 2$ and its convergence to the analytical solution in the $L^2(\Gamma; [0, 2])$ norm are depicted in Figure 7.

6.2. Convergence of iterative solvers

We demonstrate the performance of the iterative solvers introduced in Section 4 on a set of numerical experiments. The tests were performed using a mesh with 320 surface elements; the relative precision of the solvers is set to $\varepsilon := 10^{-5}$. The results for the local polynomial approximation space of degree $p = 1$ are shown in Tables I and II; in Tables III and IV, we provide results for $p = 2$. The convergence history of all solvers is depicted in Figure 8. The arguments m and l in GMRES(m) and DGMRES(m, l) denote the number of iterations between restarts and the number of vectors added to the basis of the invariant subspace after each restart, respectively. Moreover, by FGMRES($m, r(i_1, \ldots, i_r)$), we denote the solution by flexible GMRES preconditioned as in Section 4.2 with r levels of recursion and $i_1, \ldots, i_r$ iterations of the inner FGMRES solver on each level.

The experiments demonstrate that both preconditioners presented in Section 4 lead to a significant reduction of iterations and computational time. Moreover, the results indicate that only a few

DOI: 10.1002/nme

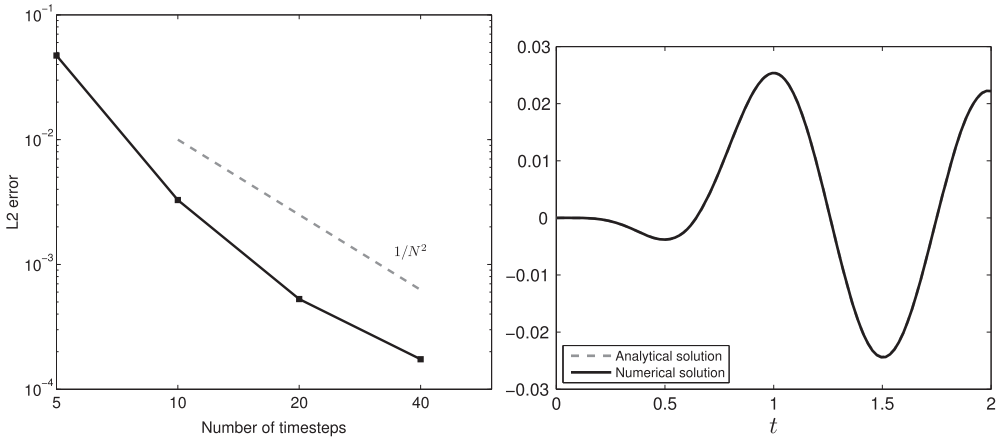


Figure 7. Left: log-log scale plot of $\|\Phi - \Phi_{\text{Galerkin}}\|_{L^2(\Gamma;[0,T])}$ for $g(t) = \sin(2\pi t)t^3\,\mathrm{e}^{-2t}$, $n = 1$, $T = 2$, and local polynomial approximation space of degree $p = 2$. Right: numerical solution in $x \in \Gamma$ obtained using 20 timesteps and $p = 2$.

Table I. Convergence of GMRES and DGMRES for $p = 1$.

	GMRES(50)		DGMRES(50,4)	
N	No. of iterations	Time (s)	No. of iterations	Time (s)
5	595	1.6	246	0.8
10	2121	14.3	421	3.3
15	4021	44.5	580	8.5
20	5448	99.0	510	14.9

Table II. Convergence of FGMRES with recursive preconditioner for $p = 1$.

	FGMRES(50, 1(10))		FGMRES(50, 1(5))		FGMRES(50, 2(2, 10))	
N	No. of iterations	Time (s)	No. of iterations	Time (s)	No. of iterations	Time (s)
5	24	0.7	45	0.9	23	0.8
10	43	3.1	126	6.8	26	3.3
15	51	7.3	205	20.0	28	5.9
20	48	9.7	341	51.2	34	10.6

Table III. Convergence of GMRES and DGMRES for $p = 2$.

	GMRES(50)		DGMRES(50,2)	
N	No. of iterations	Time (s)	No. of iterations	Time (s)
5	1985	14.3	1434	11.6
10	5404	89.1	3834	67.9
15	4634	132.7	1491	52.1
20	6383	293.3	1269	68.1

Table IV. Convergence of FGMRES with recursive preconditioner for $p = 2$.

	FGMRES(50, 1(10))		FGMRES(50, 1(5))		FGMRES(50, 2(2, 10))	
N	No. of iterations	Time (s)	No. of iterations	Time (s)	No. of iterations	Time (s)
5	83	5.5	210	8.9	—	—
10	257	44.3	627	77.3	94	26.6
15	148	48.3	363	82.2	56	24.8
20	91	45.9	399	139.4	63	40.4

DOI: 10.1002/nme

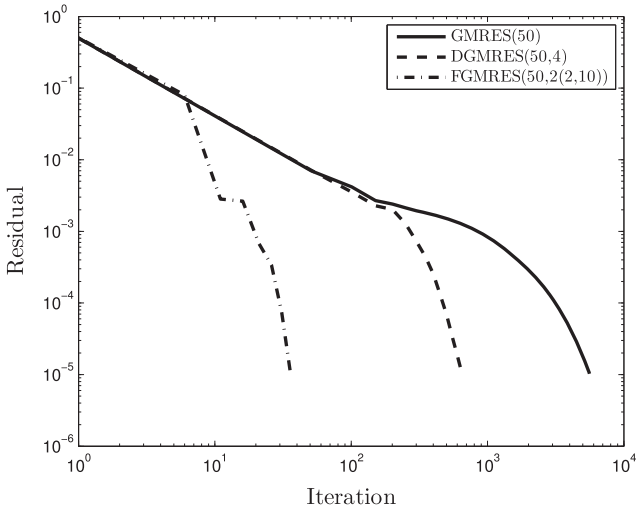


Figure 8. Convergence history of GMRES, DGMRES and FGMRES with recursive preconditioner ($p = 1, N = 20$).

Figure 9. Submarine-shaped computational geometry.

iterations of the inner solver are necessary during the application of the preconditioner in the case of FGMRES. The best convergence was obtained when the solution of (24) was approximated by two iterations of FGMRES on the first level and 10 iterations on the second level of recursion.

6.3. Wave scattering off a submarine

The following numerical experiments serve to demonstrate the behaviour of the solver on more complex geometries, such as the submarine-shaped scatterer depicted in Figure 9 with spatial dimensions 8.94 m × 1.28 m × 1.89 m. We solve the sound-hard scattering problem for $t \in [0, 12]$ with the planar incident wave [6, 22]

$$u^{\text{inc}}(x,t) := \begin{cases} A\cos(kx + \varphi_0 - \omega t), & \omega t - m_f \geqslant kx \geqslant \omega t - m_t, \\ 0, & \text{otherwise}, \end{cases}$$

where $A = 0.02$, $k = (-\pi/\sqrt{2}, 0.0, -\pi/\sqrt{2})$, $\omega = \pi$, $\varphi_0 = 7/2$, $m_f = 6\pi$ and $m_t = 8\pi$. The Neumann boundary condition (1c) is given by

$$\partial_n u = -\frac{\partial u^{\text{inc}}}{\partial n} = \begin{cases} A\sin(kx + \varphi_0 - \omega t)kn, & \omega t - m_f \geqslant kx \geqslant \omega t - m_t, \\ 0, & \text{otherwise}, \end{cases}$$

with n being the unit outward normal vector to Γ. We decompose the boundary of the scatterer into 5604 surface elements, the time interval into 40 equidistant time steps and use the local polynomial approximation space of order $p = 1$.

On the left-hand side of Figure 10, the scalability of the system matrix assembly up to 1024 cores is depicted. The assembly time was reduced from 5702 s on 16 cores to 217 s on 1024 cores. The corresponding parallel efficiency with respect to a single computational node is depicted in the right-hand side of Figure 10. The main bottleneck influencing the scalability is currently caused by the MPI communication during the non-zero pattern computation and during the gathering of matrix contributions from MPI processes.

DOI: 10.1002/nme

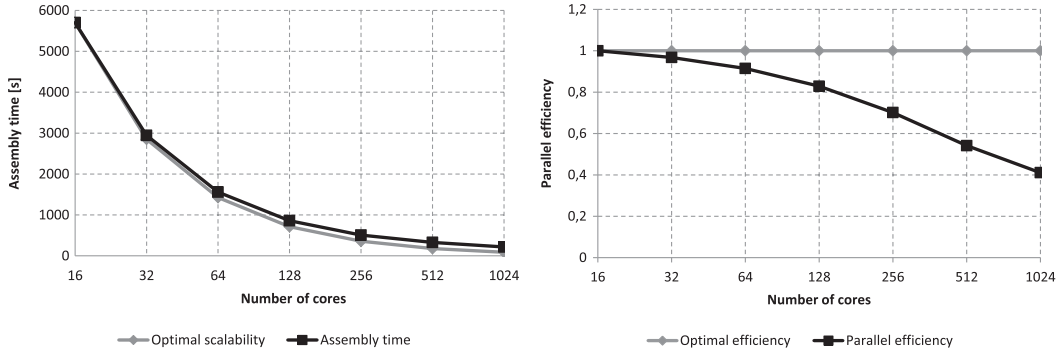


Figure 10. Parallel scalability and efficiency of the system matrix assembly up to 1024 cores (64 nodes).

Table V. Convergence of GMRES/FGMRES for scattering problem.

GMRES(500)		FGMRES(500, 1(15))		FGMRES(500, 1(30))		FGMRES(500, 1(40))	
No. of iterations	Time (s)	No. of iterations	Time (s)	No. of iterations	Time (s)	No. of iterations	Time (s)
9656	1607	614	1217	307	1076	243	962

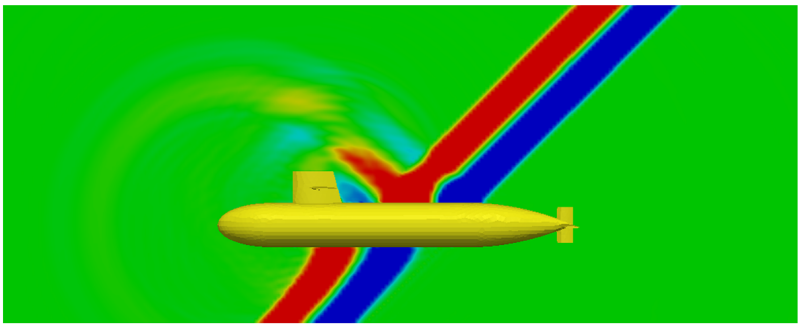


Figure 11. Solution (sum of incident and scattered wave) at time $t = 7.0$ s.

In Table V, we report on the convergence behaviour of GMRES and several variants of FGMRES with initial parallel implementation of the preconditioner presented in Section 4.2. Relative precision of the solvers was set to $\varepsilon := 10^{-4}$. Although the numbers of iterations are significantly reduced using FGMRES, further optimization of the preconditioner is necessary to reduce its application time. Finally, Figure 11 depicts the solution (sum of incident and scattered wave) in the vicinity of the scatterer at time $t = 7.0$ s. The double layer potential was evaluated in 66 049 points for 40 timesteps. The evaluation took 327 s using 64 computational nodes.

7. CONCLUSION

We considered the numerical modelling of time-dependent, three-dimensional sound-hard scattering phenomena in unbounded domains. We used TDBIEs in combination with space-time Galerkin methods to solve the arising problems numerically. The use of C^∞-smooth and compactly supported temporal basis functions simplifies the generation of the system matrix and as a consequence allows the use of higher-order approximation schemes in time which significantly improves the accuracy of the method. Because of the overlap of the basis functions in time, the arising linear systems that need to be solved admit a block Hessenberg structure. We examined the performance of a restarted GMRES solver preconditioned by deflation and proposed an algebraic preconditioner that exploits the block Hessenberg structure of the matrix. Various examples show that both solvers require a considerably lower number of iterations than a conventional GMRES method. Especially the second approach is promising since the overall solution time could be significantly reduced. A

DOI: 10.1002/nme

rigorous analysis and an optimized parallel implementation of the preconditioner, however, will be considered in the future.

We furthermore presented a parallel implementation of the scheme that is based on the boundary element library BEM4I. It exploits the special structure of the system matrix to reduce computational complexity and memory requirements. It uses OpenMP and MPI for parallelization in shared and distributed memory. Numerical experiments show good parallel scalability and efficiency of the computations in a distributed memory architecture with up to 1024 cores. However, because of the relatively costly evaluation of the smooth temporal basis functions and the necessity to assemble additional blocks, compared with the usage of the classical basis functions, the computational times are still relatively high. A possible future treatment of this problem is the usage of a fast boundary element method, such as FMM (see [23] for its application to the Galerkin BEM for transient problems).

ACKNOWLEDGEMENTS

The first author gratefully acknowledges the support given by the Swiss National Science Foundation (SNF), no. P2ZHP2_148705. Latter authors were supported by the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, as well as the Czech Ministry of Education, Youth and Sports via the project Large Research, Development and Innovations Infrastructures (LM2011033). The second and third authors were supported by VŠB-TU Ostrava under the grant SGS SP2015/160. The last author was supported by VŠB-TU Ostrava under the grant SGS SP2015/100.

REFERENCES

1. Costabel M. Time-dependent problems with the boundary integral equation method. *Encyclopedia of Computational Mechanics* 2004. DOI: 10.1002/0470091355.ecm022.
2. Banjai L, Schanz M. Wave propagation problems treated with convolution quadrature and BEM. In *Fast Boundary Element Methods in Engineering and Industrial Applications, Lecture Notes in Applied and Computational Mechanics*, Vol. 63, Langer U, Schanz M, Steinbach O, Wendland WL (eds). Springer Berlin Heidelberg, 2012; 145–184.
3. Geranmayeh A. Time domain boundary integral equations analysis. *Ph.D. Thesis*, Fachbereich Elektrotechnik und Informationstechnik, Technische UniversitŸŁ¡at Darmstadt, 2011.
4. Ha Duong T. On retarded potential boundary integral equations and their discretisation. In *Topics in Computational Wave Propagation: Direct and Inverse Problems*, Vol. 31 of Lect. Notes Comput. Sci. Eng. Springer, Berlin, 2003; 301–336.
5. Ha-Duong T, Ludwig B, Terrasse I. A Galerkin BEM for transient acoustic scattering by an absorbing obstacle. *International Journal for Numerical Methods in Engineering* 2003; **57**(13):1845–1882.
6. Sayas F. *Retarded Potentials and Time Domain Boundary Integral Equations: A Road Map.* Lecture notes, 2013.
7. Bamberger A, Ha Duong T. Formulation variationnelle espace-temps pour le calcul par potentiel retardé de la diffraction d'une onde acoustique (I). *Mathematical Methods in the Applied Sciences* 1986; **8**(3):405–435.
8. Bamberger A, Ha Duong T. Formulation variationnelle pour le calcul de la diffraction d'une onde acoustique par une surface rigide. *Mathematical Methods in the Applied Sciences* 1986; **8**(4):598–608.
9. Sauter S, Veit A. A Galerkin method for retarded boundary integral equations with smooth and compactly supported temporal basis functions. *Numerische Mathematik* 2012:1–32.
10. Sauter S, Veit A. Retarded boundary integral equations on the sphere: exact and numerical solution. *IMA Journal of Numerical Analysis* 2014; **2**(34):675–699.
11. Sauter S, Veit A. Adaptive time discretization for retarded potentials. *Numerische Mathematik* 2015:1–27. DOI: http://dx.doi.org/10.1007/s00211-015-0726-5.
12. Erhel J, Burrage K, Pohl B. Restarted GMRES preconditioned by deflation. *Journal of Computational and Applied Mathematics* May 1996; **69**(2):303–318.
13. Saad Y. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific Computing* March 1993; **14**(2):461–469.
14. Merta M, Zapletal J. *BEM4I Library*. (Available from: http://industry.it4i.cz/en/products/bem4i/) [Accessed on 26 February 2015].
15. Nédélec J. *Acoustic and Electromagnetic Equations: Integral Representations for Harmonic Problems.* No. 144 in Acoustic and electromagnetic equations: integral representations for harmonic problems. Springer: New York, 2001.
16. Khoromskij BN, Sauter S, Veit A. Fast quadrature techniques for retarded potentials based on TT/QTT tensor approximation. *Computer Methods in Applied Mathematics* 2011; **11**(3):342–362.
17. Babuška I, Melenk JM. The partition of unity method. *International Journal for Numerical Methods in Engineering* 1997; **40**:727–758.

18. Clenshaw I. A note on the summation of Chebyshev series. *Mathematics of Computation* 1955; **9**:118–120.
19. Saad Y, Schultz M. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing* 1986; **7**(3):856–869.
20. Huang Y, van der Vorst H. Some observations on the convergence behavior of GMRES. *Reports of the Faculty of Mathematics and Informatics*. Delft University of Technology, 1989.
21. Veit A. Numerical methods for time-domain boundary integral equations. *Ph.D. Thesis*, University of Zurich, 2012.
22. Glaefke M. Adaptive methods for time domain boundary integral equations. *Ph.D. Thesis*, Brunel University, 2012.
23. Sylvand G. Equation des ondes en acoustique : Acceleration des potentiels retardes par la methode multipole temporelle. *Technical Report*, INRIA, 2003.

DOI: 10.1002/nme

Conclusion and outlook

In this thesis we have presented several approaches to parallelization of the boundary element method in both space and time. We have shown the approach to accelerate the computation using the explicit vectorization leveraging the SIMD instruction set extensions of modern processors. We further extend this work and exploit the gained knowledge when accelerating the computation using the Intel Xeon Phi coprocessors installed in the Salomon supercomputer at IT4Innovations National Supercomputing Center.

Furthermore, the new method for a distribution of the sparsified system matrices proved to be suitable for usage combined with the adaptive cross approximation as well as the fast multipole method. The presented approach has been tested with piecewise constant basis functions and its extension to piecewise polynomial basis functions is a topic of current research.

In addition to the stationary problems modelled by the elliptic partial differential equations we have been focusing on the parallelization of the Galerkin boundary element method for solution of the time-dependent wave equation in 3D. To deal with the problems associated with the numerical quadrature during the assembly of the system matrices we employed the novel smooth temporal basis functions introduced by Sauter and Veit [27]. Parallelization is crutial in order to enable solution of large scale problems since this approach significantly increases the computational and memory requirements. To the best of our knowledge the parallelization of this technique has not yet been presented. Moreover, since the special structure of the system matrices requires a usage of an iterative solver, we presented solvers based on the GMRES method preconditioned by deflation and by algebraic recursive preconditioner. In future this approach can be extended to support solution of other time dependent problems, e.g., the heat equation.

Most of the abovementioned techniques have been implemented in the boundary element library BEM4I developed at IT4Innovations National Supercomputing Center. In addition to the described solvers for the Laplace and wave equation the library contains the modules treating the Helmholtz and Lamé equations. It is parallelized by OpenMP and MPI in shared and distributed memory, respecively, utilizes SIMD vectorization, and is accelerated using the Intel Xeon Phi coprocessors.

Due to its high computational demands the boundary element method is a suitable candidate for efficient utilization of current peta-scale and upcoming exa-scale computer architectures. Mainly the space-time Galerkin boundary element methods for time-dependent problems produce very large system matrices which are global in space and time. While this may be from one point of view considered a disadvantage, from the other point of view it enables to introduce another level of parallelism in temporal dimension which would be impossible with most of other methods for solving time-dependent problems. For this reason we believe that there are many promising topics of research that may in future stem from this work.

Author's publications

Papers in journals with impact factor

- A. Veit, M. Merta, J. Zapletal, and D. Lukáš. "Efficient solution of time-domain boundary integral equations arising in sound-hard scattering". In: *International Journal for Numerical Methods in Engineering* (2016). Article in press. IF 2.055 (Q1). DOI: 10.1002/nme.5187.
- D. Lukáš, P. Kovář, T. Kovářová, and M. Merta. "A parallel fast boundary element method using cyclic graph decompositions'. In: *Numerical Algorithms* 70.4 (2015). IF 1.417 (Q1), pp. 807–824. DOI: 10.1007/s11075-015-9974-9.
- M. Merta and J. Zapletal. "Acceleration of boundary element method by explicit vectorization". In: *Advances in Engineering Software* 86 (2015). IF 1.402 (Q2), pp. 70–79. DOI: 10.1016/j.advengsoft.2015.04.008.
- M. Čermák, V. Hapla, D. Horák, M. Merta, and A. Markopoulos. "Total-FETI domain decomposition method for solution of elasto-plastic problems". In: *Advances in Engineering Software* 84 (2015). IF 1.402 (Q2), pp 48–54.
  DOI: 10.1016/j.advengsoft.2014.12.011.
- M. Merta and J. Zapletal "A parallel library for boundary element discretization of engineering problems". Submitted to Mathematics and Computers in Simulation (2014). IF 0.949.

Papers in conference proceedings

- M. Merta and J. Zapletal. "BEM4I applied to shape optimization problems". In: *AIP Conference Proceedings* 1738 (2016). DOI: 10.1063/1.4952145.
- M. Merta, J. Zapletal, J. Jaros. "Many core acceleration of the boundary element method". In: *Lecture Notes in Computer Science* 9611 LNCS (2016), pp. 116–125. DOI: 10.1007/978-3-319-40361-8_8

- M. Čermák, M. Merta, and J. Zapletal. "A novel boundary element library with applications". In: *AIP Conference Proceedings* 1648 (2015). DOI: 10.1063/1.4913036.
- Merta, M., Zapletal, J. "Library of parallel boundary element method based solvers for solution of the time-dependent wave equation". In: *Civil-Comp Proceedings 107* (2015).
- M. Merta, A. Vašatová, V. Hapla, and D. Horák. "Parallel implementation of total-FETI DDM with application to medical image registration". In: *Lecture Notes in Computational Science and Engineering* 98 (2014), pp. 917–925. DOI: 10.1007/978-3-319-05789-7_89
- V. Hapla, D. Horák, and M. Merta. "Use of direct solvers in TFETI massively parallel implementation". In: *Lecture Notes in Computer Science* 7782 LNCS (2013), pp. 192–205. DOI: 10.1007/987-3-642-36803-5_14.
- D. Horák, P. Kabelíková, M. Merta, and V. Vondrák. "The OOSol scalable library based on a domain decomposition method". In: *Civil-Comp Proceedings 95* (2011).

Conferences[1]

- M. Merta, A. Veit, J. Zapletal, and D. Lukáš. "Parallel time-domain boundary element method for 3-dimensional wave equation". MAFELAP 2016, London, United Kingdom, 2016.
- M. Merta and J. Zapletal. "Acceleration of the boundary element library BEM4I using the Intel Xeon Phi coprocessors". CMSE 2016, Rožnov p. Radhoštěm, Czech Republic, 2016 & PRACEDays16, Prague, Czech Republic, 2016 (poster).
- M. Merta. "Introduction to the Intel Xeon Phi programming". PRACE Winter School 2016, Bratislava, Slovakia, 2016.
- M. Merta, J. Zapletal, L. Říha, T. Brzobohatý, A. Markopoulos, and O. Meca. "Acceleration of the numerical libraries using the Intel Xeon Phi coprocessors". 4th IT4Innovations Annual Conference, Přerov, Czech Republic, 2015.
- M. Merta and J. Zapletal. "Acceleration of the BEM4I library using the Intel Xeon Phi coprocessors". 13. Workshop on fast boundary element methods in industrial applications, Hirschegg, Austria, 2015.
- M. Merta. "Acceleration of the boundary element method". WOFEX 2015, Ostrava, Czech Republic, 2015.

[1] Speaker or poster presenter

- M. Merta and J. Zapletal. "Library of parallel boundary element method based solvers for solution of the time dependent wave equation". PARENG 2015, Dubrovnik, Croatia, 2015.
- M. Merta and J. Zapletal. "BEM4I applied to 3D wave scattering problems". EASC 2015, Edinburgh, United Kingdom, 2015 (poster).
- M. Merta and J. Zapletal. "A library of parallel solvers based on the boundary element method". 3rd IT4Innovations Annual Conference, Ostrava, Czech Republic, 2014.
- M. Merta and J. Zapletal. "BEM4I – Parallel boundary element library". Modelling 2014, Rožnov p. Radhoštěm, Czech Republic, 2014 & ESCO 2014, Pilsen, Czech Republic, 2014 & EASC 2014, Stockholm, Sweden, 2014 (poster).
- M. Merta. "Parallel solution of the time dependent wave equation". WOFEX 2014, Ostrava, Czech Republic, 2014.
- M. Merta and J. Zapletal. "BEM4I – Parallel BEM library with applications to the wave equation". Modelling 2014, Rožnov p. Radhoštěm, Czech Republic, 2014.
- M. Merta and J. Zapletal. "BEM4I – Preview of the new BEM library". SPOMECH Workshop 2013, Ostrava, Czech Republic, 2013.
- M. Merta. "Parallel implementation of fast boundary element method". WOFEX 2013, Ostrava, Czech Republic, 2013.
- M. Merta and D. Lukáš. "Parallel implementation of fast boundary element method". SNA 2013, Rožnov p. Radhoštěm, Czech Republic, 2013.
- M. Merta and M. Čermák. "Parallel implementation of TFETI DDM for the solution of elasto-plastic problems". PARENG 2013, Pécs, Hungary, 2013.
- M. Merta. "Introduction to Scientific Computing Using Trilinos". PRACE Spring School 2012, Kraków, Poland, 2012.
- M. Merta, A. Vašatová, V. Hapla, and D. Horák. "Massively parallel implementation of total-FETI method with application to medical image registration". Workshop Fast BEM and BETI, Ostrava, Czech Republic, 2012.
- M. Merta, A. Vašatová, V. Hapla, and D. Horák. "Massively parallel implementation of total-FETI method with application to medical image registration". DD 21, Rennes, France, 2012.

Software

- M. Merta and J. Zapletal. "BEM4I" [Software]. Available from `http://bem4i.it4i.cz`.

Bibliography

[1] A. Bamberger and T. Ha Duong. "Formulation variationnelle espace-temps pour le calcul par potentiel retardé de la diffraction d'une onde acoustique (I)". In: *Mathematical methods in the applied sciences* 8.1 (1986), pp. 405–435.

[2] M. Bebendorf. "Approximation of boundary element matrices". In: *Numer. Math.* 86 (2000).

[3] M. Bebendorf. *Hierarchical Matrices*. Springer, 2008.

[4] M. Bebendorf and R. Kriemann. "Fast parallel solution of boundary integral equations and related problems". In: *Comp. Vis. Sci.* 8 (2005).

[5] M. Bebendorf and S. Rjasanow. "Adaptive Low-Rank Approximation of Collocation Matrices". English. In: *Computing* 70.1 (2003), pp. 1–24. ISSN: 0010-485X. DOI: 10.1007/s00607-002-1469-6. URL: http://dx.doi.org/10.1007/s00607-002-1469-6.

[6] J. Bouchala and J. Zapletal. "Effective Semi-Analytic Integration for Hypersingular Galerkin Boundary Integral Equations for the Helmholtz Equation in 3D". In: *Appl. Math.* (to appear).

[7] E. Bécache. *Equations intégrales pour l'équation des ondes*. Available at http://www-rocq.inria.fr/~becache/cours_eqinteg.ps.gz.. [Online; accessed 24-November-2015]. 1994.

[8] M. Costabel. "Time-Dependent Problems with the Boundary Integral Equation Method". In: *Encyclopedia of Computational Mechanics* (2004).

[9] M.G. Duffy. "Quadrature over a pyramid or cube of integrands with a singularity at a vertex". In: *SIAM J. Numer. Anal.* 19 (1982).

[10] J. El Gharib. "Problèmes de potentiels retardés pour l'acoustique". PhD thesis. École Polytechnique, 1999.

[11] L. Greengard and V. Rokhlin. "A Fast Algorithm for Particle Simulations". In: *Journal of Computational Physics* 135.2 (1997), pp. 280 –292. ISSN: 0021-9991. DOI: http://dx.doi.org/10.1006/jcph.1997.5706. URL: http://www.sciencedirect.com/science/article/pii/S0021999197957065.

[12] T. Ha-Duong. "On retarded potential boundary integral equations and their discretisation". In: *Topics in computational wave propagation*. Springer, 2003, pp. 301–336.

[13] T. Ha Duong, A. Bamberger, and J. C. Nedelec. "Formulation variationnelle pour le calcul de la diffraction d'une onde acoustique par une surface rigide". In: *Mathematical Methods in the Applied Sciences* 8.1 (1986), pp. 598–608. ISSN: 1099-1476. DOI: 10.1002/mma.1670080139. URL: http://dx.doi.org/10.1002/mma.1670080139.

[14] T. Ha-Duong, B. Ludwig, and I. Terrasse. "A Galerkin BEM for transient acoustic scattering by an absorbing obstacle". In: *International Journal for Numerical Methods in Engineering* 57.13 (2003), pp. 1845–1882. ISSN: 1097-0207. DOI: 10.1002/nme.745. URL: http://dx.doi.org/10.1002/nme.745.

[15] W. Hackbusch, Wendy Kress, and S. Sauter. "Boundary Element Analysis: Mathematical Aspects and Applications". In: ed. by Martin Schanz and Olaf Steinbach. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. Chap. Sparse Convolution Quadrature for Time Domain Boundary Integral Formulations of the Wave Equation by Cutoff and Panel-Clustering, pp. 113–134. ISBN: 978-3-540-47533-0. DOI: 10.1007/978-3-540-47533-0_5. URL: http://dx.doi.org/10.1007/978-3-540-47533-0_5.

[16] W. Hackbusch and Z.P. Nowak. "On the fast matrix multiplication in the boundary element methods by panel clustering". In: *Numer. Math.* 54 (1989).

[17] M. Kretz and V. Lindenstruth. "Vc: A C++ library for explicit vectorization". In: *Software: Practice and Experience* 42.11 (2012), pp. 1409–1430. ISSN: 1097-024X. DOI: 10.1002/spe.1149. URL: http://dx.doi.org/10.1002/spe.1149.

[18] J.L. Lions and E. Magenes. *Non-Homogeneous Boundary Value Problems and Applications*. Springer-Verlag, 1972.

[19] Ch. Lubich. "On the multistep time discretization of linear initial-boundary value problems and their boundary integral equations". In: *Numerische Mathematik* 67.3 (1994), pp. 365–389.

[20] D. Lukáš, P. Kovář, T. Kovářová, and M. Merta. "A parallel fast boundary element method using cyclic graph decompositions". In: *Numerical Algorithms* 70.4 (2015). IF 1.417 (Q1), pp. 807–824. DOI: 10.1007/s11075-015-9974-9.

[21] M. Merta and J. Zapletal. "Acceleration of boundary element method by explicit vectorization". In: *Advances in Engineering Software* 86 (2015). IF 1.402 (Q2), pp. 70–79. DOI: 10.1016/j.advengsoft.2015.04.008.

[22] M. Merta and J. Zapletal. *BEM4I*. Available at http://industry.it4i.cz/en/products/bem4i/. [Online; accessed 13-October-2014]. 2014.

[23] M. Merta, J. Zapletal, and J. Jaros. "Many core acceleration of the boundary element method". Accepted to Lecture Notes in Computer Science, 2015.

[24] S. Rjasanow and O. Steinbach. *The Fast Solution of Boundary Integral Equations*. Springer, 2007. ISBN: 978-0-387-34042-5.

[25] V. Rokhlin. "Rapid solution of integral equations of classical potential theory". In: *Journal of Computational Physics* 60.2 (1985), pp. 187 –207. ISSN: 0021-9991. DOI: http://dx.doi.org/10.1016/0021-9991(85)90002-6. URL: http://www.sciencedirect.com/science/article/pii/0021999185900026.

[26] S. Sauter and C. Schwab. *Boundary Element Methods*. Springer Series in Computational Mathematics. Springer, 2010. ISBN: 9783540680932.

[27] S. Sauter and A. Veit. "A Galerkin method for retarded boundary integral equations with smooth and compactly supported temporal basis functions". In: *Numerische Mathematik* 123.1 (2013), pp. 145–176.

[28] S. Sauter and A. Veit. "A Galerkin Method for Retarded Boundary Integral Equations with Smooth and Compactly Supported Temporal Basis Functions. Part II: Implementation and Reference Solutions." In: *Preprint* (2011).

[29] S. Sauter and A. Veit. "Adaptive Time Discretization for Retarded Potentials". In: *arXiv preprint arXiv:1404.2322* (2014).

[30] S. Sauter and A. Veit. "Adaptive time discretization for retarded potentials". In: *Numerische Mathematik* (2015). cited By 1; Article in Press. DOI: 10.1007/s00211-015-0726-5. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-84930150076&partnerID=40&md5=bb62bcad9a0506dc80e0c2111afaad9e.

[31] S. Sauter and A. Veit. "Retarded boundary integral equations on the sphere: exact and numerical solution". In: *IMA Journal of Numerical Analysis* (2013). DOI: 10.1093/imanum/drs059. eprint: http://imajna.oxfordjournals.org/content/early/2013/07/14/imanum.drs059.full.pdf+html. URL: http://imajna.oxfordjournals.org/content/early/2013/07/14/imanum.drs059.abstract.

[32] O. Steinbach. *Numerical Approximation Methods for Elliptic Boundary Value Problems: Finite and Boundary Elements*. Texts in applied mathematics. Springer, 2008. ISBN: 9780387313122.

[33] E. P. Stephan, M. Maischak, and E. Ostermann. "Transient Boundary Element Method and Numerical Evaluation of Retarded Potentials". English. In: *Computational Science*

– *ICCS 2008*. Ed. by Marian Bubak, GeertDick van Albada, Jack Dongarra, and PeterM.A. Sloot. Vol. 5102. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 321–330. ISBN: 978-3-540-69386-4. DOI: 10.1007/978-3-540-69387-1_35. URL: http://dx.doi.org/10.1007/978-3-540-69387-1_35.

[34] A. Veit, M. Merta, J. Zapletal, and D. Lukáš. "Efficient solution of time-domain boundary integral equations arising in sound-hard scattering". In: *International Journal for Numerical Methods in Engineering* (2016). Article in Press. IF 2.055 (Q1). DOI: 10.1002/nme.5187.

Appendix A

Co-authors' statements

I hereby agree with the co-author's description of his contribution to the paper *Acceleration of boundary element method by explicit vectorization* and confirm that my contribution to the paper is 30 %.

In Ostrava on 9 June 2016

.....

Signed Ing. Jan Zapletal

I hereby agree with the co-author's description of his contribution to the paper *A parallel boundary element method using cyclic graph decompositions* and confirm that my contribution to the paper is 50 %.

In Ostrava on 9 June 2016

D. Lukáš

.....

Signed doc. Ing. Dalibor Lukáš, Ph.D.

I hereby agree with the co-author's description of his contribution to the paper *A parallel boundary element method using cyclic graph decompositions* and confirm that my contribution to the paper is 20 %.

In Ostrava on 9 June 2016

.....

Signed doc. Mgr. Petr Kovář, Ph.D.

I hereby agree with the co-author's description of his contribution to the paper *A parallel boundary element method using cyclic graph decompositions* and confirm that my contribution to the paper is 10 %.

In Ostrava on 9 June 2016

.....

Signed Mgr. Tereza Kovářová, Ph.D.

Dr. Alexander Veit
Hügelstrasse 39, 8002 Zürich Switzerland
Tel: +41 77 90 22 167, E-Mail: alexander.veit@credit-suisse.com

Michal Merta
Faculty of Electrical Engineering and Computer Science
VŠB - Technical University of Ostrava
17. listopadu 15
708 33 Ostrava-Poruba
Czech Republic

Zurich, 14.06.2016

Approving statement

Dear Michal,

I hereby agree with the description of your contribution to our paper "*Efficient solution of time-domain boundary integral equation arising in sound hard scattering*" and confirm that my contribution to the paper was 25%.

Sincerely

A. Veit

Dr. Alexander Veit

I hereby agree with the co-author's description of his contribution to the paper *Efficient solution of time-domain boundary integral equations arising in sound-hard scattering* and confirm that my contribution to the paper is 5 %.

In Ostrava on 9 June 2016

.....

Signed Ing. Jan Zapletal

I hereby agree with the co-author's description of his contribution to the paper *Efficient solution of time-domain boundary integral equation arising in sound hard scattering* and confirm that my contribution to the paper is 5 %.

In Ostrava on 9 June 2016

D. Lukáš

.....

Signed doc. Ing. Dalibor Lukáš, Ph.D.