Technische Universität Graz

A parallel space-time boundary element method for the heat equation

S. Dohr, J. Zapletal, G. Of, M. Merta, M. Kravcenko

Berichte aus dem Institut für Angewandte Mathematik

Bericht 2018/2

Technische Universität Graz

A parallel space-time boundary element method for the heat equation

S. Dohr, J. Zapletal, G. Of, M. Merta, M. Kravcenko

Berichte aus dem Institut für Angewandte Mathematik

Bericht 2018/2

Technische Universität Graz
Institut für Angewandte Mathematik
Steyrergasse 30
A 8010 Graz

WWW: `http://www.applied.math.tugraz.at`

A parallel space-time boundary element method for the heat equation

Stefan Dohr[a], Jan Zapletal[b,c,*], Günther Of[a], Michal Merta[b,c], Michal Kravcenko[b,c]

[a] *Institute of Applied Mathematics, Graz University of Technology. Steyrergasse 30, A-8010 Graz, Austria*
[b] *IT4Innovations, VŠB – Technical University of Ostrava. 17. listopadu 2172/15, 708 33 Ostrava-Poruba, Czech Republic*
[c] *Department of Applied Mathematics, VŠB – Technical University of Ostrava. 17. listopadu 2172/15, 708 33 Ostrava-Poruba, Czech Republic*

Abstract

In this paper we introduce a new parallel solver for the weakly singular space-time boundary integral equation for the heat problem. The space-time boundary mesh is decomposed into a given number of submeshes. Pairs of the submeshes represent blocks in the system matrices, which are distributed among computational nodes by an algorithm based on a cyclic decomposition of complete graphs ensuring load balance. In addition, we employ vectorization and threading in shared memory to ensure intra-node efficiency. We present scalability experiments on different CPU architectures to evaluate the performance of the proposed parallelization techniques. All levels of parallelism allow us to tackle large problems and lead to an almost optimal speedup.

Keywords: space-time boundary element method, heat equation, parallelization, vectorization
2010 MSC: 65N38, 65Y05, 68W10

1. Introduction

Boundary integral equations and related boundary element methods have been applied for the solution of the linear heat equation for decades [1, 2, 3, 4]. A survey on boundary element methods for the heat and the wave equation is provided in [5]. One can use Laplace transform methods like the convolution quadrature method [6], time-stepping methods [7], and space-time integral equations. Besides the Nyström [8] and collocation methods [9], the Galerkin approach [1, 2, 3, 10, 11] can be applied for the discretization of space-time integral equations.

*Corresponding author
Email address: jan.zapletal@vsb.cz (Jan Zapletal)

The matrices related to the discretized space-time integral equations are dense and their dimension is much higher than in the case of stationary problems. Even with fast methods, see, e.g. [10, 12], the computational times and the memory requirements of the huge space-time system are demanding. Thus the solution of even moderately sized problems requires the use of computer clusters. Although there is a simple parallelization by OpenMP in the FMM code of [11], parallelization of boundary element methods for the heat equation in HPC environments has not been closely investigated yet, to the best of our knowledge. In this paper we concentrate on hybrid parallelization in shared and distributed memory.

The global space-time nature of the system matrices leads to improved parallel scalability in distributed memory systems in contrast to time-stepping methods where the parallelization is usually limited to spatial dimensions. For this reason, parallel-in-time algorithms have been considered suitable for tackling the problems of the upcoming exascale era when more than 100 million way concurrency will be required [13, 14, 15]. Methods such as parareal [16] or space-time parallel multigrid [17] are gaining in popularity. We present a method for parallelization of space-time BEM for the heat equation based on a modification of the approach presented in [18, 19] for spatial problems. The method is based on a decomposition of the input mesh into submeshes of approximately the same size and a distribution of corresponding blocks of the system matrices among processors. To ensure proper load balancing during the assembly of system matrices and matrix-vector multiplication, a distribution of matrix blocks based on a cyclic graph decomposition is used. We modify the original approach to support the special structure of the space-time system matrices.

Numerical or semi-analytic evaluation of the surface integrals is one of the most time-consuming parts of space-time BEM. The high computational intensity of the method makes it well suited for current multi- and many-core processors equipped with wide SIMD (Single Instruction Multiple Data) registers. Vector instruction set extensions in modern CPUs (AVX512, AVX2, SSE) support simultaneous operations with up to eight double precision operands, contributing significantly to the theoretical peak performance of a processor. While current compilers support automatic vectorization to some extent, one has to use low level approaches (assembly language, compiler intrinsic functions), external libraries (Vc [20], Intel MKL Vector Mathematical Functions [21], etc.), or OpenMP pragmas [22] to achieve a reasonable speed-up. We focus on the OpenMP approach due to its portability and relative ease of use. Moreover, we also utilize OpenMP for thread parallelization in shared memory.

The structure of the paper is as follows. In Section 2 we introduce the two-dimensional model problem, derive its boundary integral formulation, and discretize it to obtain a BEM system. Section 3 is devoted to the description of our parallel and vectorized implementation of the matrix assembly and solution of the system of linear equations based on OpenMP and MPI. In Section 4 we provide results of numerical and scalability experiments validating the suggested approach and we conclude in Section 5.

2. Boundary integral equations for the heat problem

2.1. Model problem and boundary integral equations

Let $\Omega \subset \mathbb{R}^2$ be a bounded domain with a Lipschitz boundary $\Gamma := \partial\Omega$ and $T > 0$. As a model problem we consider the initial Dirichlet boundary value problem for the heat equation

$$\begin{aligned} \alpha\partial_t u - \Delta_x u &= 0 \quad \text{in } Q := \Omega \times (0,T), \\ u &= g \quad \text{on } \Sigma := \Gamma \times (0,T), \\ u &= u_0 \quad \text{in } \Omega \end{aligned} \tag{2.1}$$

with the heat capacity constant $\alpha > 0$, the given initial datum u_0, and the boundary datum g. The solution of (2.1) can be expressed by using the representation formula for the heat equation [23], i.e. for $(x,t) \in Q$ we have

$$u(x,t) = (\widetilde{M}_0 u_0)(x,t) + (\widetilde{V}\partial_n u)(x,t) - (Wg)(x,t) \tag{2.2}$$

with the intial potential

$$(\widetilde{M}_0 u_0)(x,t) := \int_\Omega U^\star(x-y,t)u_0(y)\,\mathrm{d}y,$$

the single-layer potential

$$(\widetilde{V}\partial_n u)(x,t) := \frac{1}{\alpha}\int_\Sigma U^\star(x-y,t-\tau)\frac{\partial}{\partial n_y}u(y,\tau)\,\mathrm{d}s_y\,\mathrm{d}\tau,$$

and the double-layer potential

$$(Wg)(x,t) := \frac{1}{\alpha}\int_\Sigma \frac{\partial}{\partial n_y}U^\star(x-y,t-\tau)g(y,\tau)\,\mathrm{d}s_y\,\mathrm{d}\tau.$$

The function $U^\star$ denotes the fundamental solution of the two-dimensional heat equation given by

$$U^\star(x-y,t-\tau) = \begin{cases} \dfrac{\alpha}{4\pi(t-\tau)}\exp\left(\dfrac{-\alpha|x-y|^2}{4(t-\tau)}\right) & \text{for } \tau < t, \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

Hence, it suffices to determine the unknown Neumann datum $\partial_n u_{|\Sigma}$ to compute the solution of (2.1). It is well known [3, 24] that for $u_0 \in L^2(\Omega)$ and $g \in H^{1/2,1/4}(\Sigma)$ the problem (2.1) admits a unique solution $u \in H^{1,1/2}(Q, \alpha\partial_t - \Delta_x)$ with the anisotropic Sobolev space

$$H^{1,1/2}(Q,\alpha\partial_t - \Delta_x) := \left\{ u \in H^{1,1/2}(Q) \colon (\alpha\partial_t - \Delta_x)u \in L^2(Q) \right\}.$$

The unknown density $w := \partial_n u_{|\Sigma} \in H^{-1/2,-1/4}(\Sigma)$ can be found by applying the interior Dirichlet trace operator $\gamma_0^{\mathrm{int}} \colon H^{1,1/2}(Q) \to H^{1/2,1/4}(\Sigma)$ to the representation formula (2.2) leading to

$$g(x,t) = (M_0 u_0)(x,t) + (Vw)(x,t) + ((\frac{1}{2}I - K)g)(x,t) \quad \text{for } (x,t) \in \Sigma.$$

The operator $M_0 \colon L^2(\Omega) \to H^{1/2,1/4}(\Sigma)$, the single-layer boundary integral operator $V \colon H^{-1/2,-1/4}(\Sigma) \to H^{1/2,1/4}(\Sigma)$, and the double-layer boundary integral operator $\frac{1}{2}I - K \colon H^{1/2,1/4}(\Sigma) \to H^{1/2,1/4}(\Sigma)$ are obtained by composition of the potentials in (2.2) with the Dirichlet trace operator γ_0^{int}, see [3, 23]. We solve the variational formulation to find $w \in H^{-1/2,-1/4}(\Sigma)$ such that

$$\langle Vw, \tau \rangle_\Sigma = \langle (\frac{1}{2}I + K)g, \tau \rangle_\Sigma - \langle M_0 u_0, \tau \rangle_\Sigma \quad \text{for all } \tau \in H^{-1/2,-1/4}(\Sigma), \tag{2.4}$$

where $\langle \cdot, \cdot \rangle_\Sigma$ denotes the duality pairing on $H^{1/2,1/4}(\Sigma) \times H^{-1/2,-1/4}(\Sigma)$. The single-layer boundary integral operator V is bounded and elliptic [2, 3], i.e. there exists a constant $c_1^V > 0$ such that

$$\langle Vw, w \rangle_\Sigma \geq c_1^V \, \|w\|^2_{H^{-1/2,-1/4}(\Sigma)} \quad \text{for all } w \in H^{-1/2,-1/4}(\Sigma).$$

Thus, the variational formulation (2.4) is uniquely solvable.

2.2. Boundary element method

For the Galerkin boundary element discretization of the variational formulation (2.4) we consider a space-time tensor product decomposition of Σ [1, 10]. For a given triangulation $\Gamma_h = \{\gamma_i\}_{i=1}^{N_\Gamma}$ of the boundary Γ and a given decomposition $I_h = \{\tau_j\}_{j=1}^{N_I}$ of the time interval $I := (0, T)$ we define $\Sigma_h := \{\sigma = \gamma_i \times \tau_j \colon i = 1, ..., N_\Gamma;\, j = 1, ..., N_I\}$, i.e. we have $\Sigma_h = \{\sigma_\ell\}_{\ell=1}^N$ and

$$\overline{\Sigma} = \bigcup_{\ell=1}^N \overline{\sigma}_\ell$$

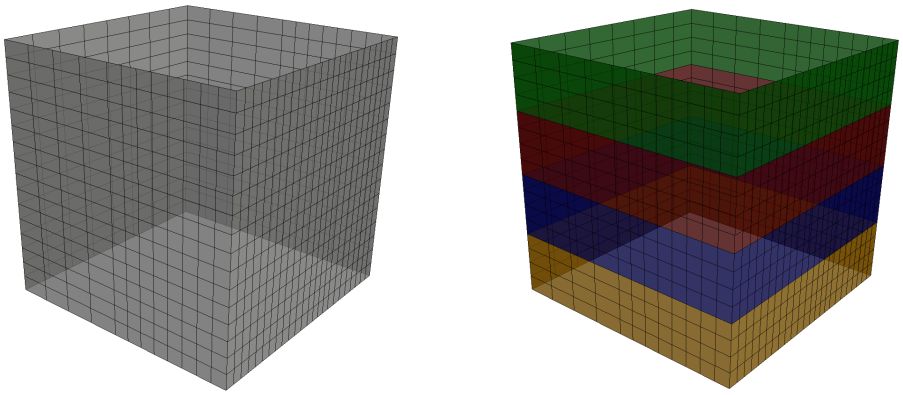
with $N := N_\Gamma N_I$. In the two-dimensional case the space-time boundary elements σ are rectangular. A sample decomposition of the space-time boundary of $Q = (0, 1)^3$ is shown in Fig. 2.1a.

For the discretization of (2.4) we use the space $X_h^{0,0}(\Sigma_h) := \operatorname{span}\{\varphi_\ell^0\}_{\ell=1}^N$ of piecewise constant basis functions φ_ℓ^0, which is defined with respect to the decomposition Σ_h. For the approximation of the Dirichlet datum g we consider the space $X_h^{1,0}(\Sigma_h) := \operatorname{span}\{\varphi_i^{10}\}_{i=1}^N$ of functions that are piecewise linear and globally continuous in space and piecewise constant in time, while the initial datum u_0 is discretized by using the space of piecewise linear and globally continuous functions $S_h^1(\Omega_h) = \operatorname{span}\{\varphi_i^1\}_{i=1}^M$, which is defined with respect to a given triangulation $\Omega_h := \{\omega_i\}_{i=1}^{N_\Omega}$ of the domain Ω. This leads to the system of linear equations

$$\mathsf{V}_h \boldsymbol{w} = \left(\frac{1}{2}\mathsf{M}_h + \mathsf{K}_h\right) \boldsymbol{g} - \mathsf{M}_h^0 \boldsymbol{u_0} \tag{2.5}$$

where

$$\mathsf{V}_h[\ell, k] := \frac{1}{\alpha} \int_{\sigma_\ell} \int_{\sigma_k} U^\star(x - y, t - \tau) \, \mathrm{d}s_y \, \mathrm{d}\tau \, \mathrm{d}s_x \, \mathrm{d}t, \tag{2.6}$$



(a) Tensor product decomposition. (b) Submeshes.

Figure 2.1: Sample space-time boundary decompositions for $Q = (0,1)^3$.

$$\mathsf{K}_h[\ell, j] := \frac{1}{\alpha} \int_{\sigma_\ell} \int_{\Sigma} \frac{\partial}{\partial n_y} U^\star(x - y, t - \tau) \varphi_j^{10}(y, \tau) \,\mathrm{d}s_y \,\mathrm{d}\tau \,\mathrm{d}s_x \,\mathrm{d}t, \tag{2.7}$$

$$\mathsf{M}_h^0[\ell, j] := \int_{\sigma_\ell} \int_{\Omega} U^\star(x - y, t) \varphi_j^1(y) \,\mathrm{d}y \,\mathrm{d}s_x \,\mathrm{d}t, \tag{2.8}$$

and

$$\mathsf{M}_h[\ell, j] := \int_{\sigma_\ell} \int_{\Sigma} \varphi_j^{10}(y, \tau) \,\mathrm{d}s_y \,\mathrm{d}\tau \,\mathrm{d}s_x \,\mathrm{d}t. \tag{2.9}$$

The vectors $\boldsymbol{w}, \boldsymbol{g} \in \mathbb{R}^N$ and $\boldsymbol{u_0} \in \mathbb{R}^M$ in (2.5) represent the coefficients of the trial function $w_h := \sum_{\ell=1}^N w_\ell \varphi_\ell^0$, and the given approximations $g_h = \sum_{\ell=1}^N g_\ell \varphi_\ell^{10}$ and $u_h^0 := \sum_{i=1}^M u_i^0 \varphi_i^1$ of the Dirichlet datum g and the initial datum u_0, respectively. Due to the ellipticity of the single-layer operator V the matrix V_h is positive definite and therefore (2.5) is uniquely solvable.

We assume that the elements of I_h, referred to as time layers, are sorted from $t = 0$ to $t = T$. Due to the causal behaviour of the fundamental solution (2.3) the matrices V_h and K_h are block lower triangular matrices, where each block corresponds to one pair of time layers, see (2.10) in the case of V_h. The structure of K_h is identical to V_h.

$$\mathsf{V}_h = \begin{bmatrix} \mathsf{V}_{0,0} & 0 & \cdots & 0 \\ \mathsf{V}_{1,0} & \mathsf{V}_{1,1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \mathsf{V}_{N_I-1,0} & \mathsf{V}_{N_I-1,1} & \cdots & \mathsf{V}_{N_I-1,N_I-1} \end{bmatrix} \tag{2.10}$$

The structure of the initial matrix M_h^0 is different. The number of its columns depends on the number of vertices of the initial mesh Ω_h, while the number of rows depends on the number of space-time boundary elements σ. Due to the

given sorting of the elements of I_h the matrix can be decomposed into block-rows where each block-row corresponds to one time layer. For the mass matrix M_h we obtain a block-diagonal structure, where each diagonal block represents the local mass matrix of one time layer.

By using the representation formula (2.2) with the computed approximations w_h, g_h and u_h^0, we can compute an approximation $\tilde{u}$ of u, i.e. for $(x,t) \in Q$ we obtain

$$\tilde{u}(x,t) = \sum_{i=1}^{M} u_i^0 (\widetilde{M}_0 \varphi_i^1)(x,t) + \sum_{\ell=1}^{N} w_\ell (\widetilde{V} \varphi_\ell^0)(x,t) - \sum_{\ell=1}^{N} g_\ell (W \varphi_\ell^{10})(x,t). \tag{2.11}$$

For the evaluation of the discretized representation formula (2.11) in Q we define a specific set of evaluation points. Let $\{x_\ell\}_{\ell=1}^{E_\Omega}$ be a set of nodes in the interior of the domain Ω, e.g. the nodes of the already given triangulation Ω_h on a specific level. Moreover let $\{t_k\}_{k=1}^{E_I}$ be an ordered set of time steps distributed on the interval $I = (0,T)$. The set of evaluation points is then given as

$$\{(x,t)_i\}_{i=1}^{E} = \{(x_\ell, t_k) \colon \ell = 1, ..., E_\Omega; k = 1, ..., E_I\} \tag{2.12}$$

with $E = E_\Omega E_I$. We have to evaluate the integrals in (2.11) for each evaluation point, i.e. we have to compute

$$\boldsymbol{u_h} = \widetilde{\mathsf{M}}_h^0 \boldsymbol{u_0} + \widetilde{\mathsf{V}}_h \boldsymbol{w} - \widetilde{\mathsf{W}}_h \boldsymbol{g} \tag{2.13}$$

where

$$\begin{aligned} \widetilde{\mathsf{M}}_h^0[i,j] &:= (\widetilde{M}_0 \varphi_j^1)((x,t)_i), \\ \widetilde{\mathsf{V}}_h[i,\ell] &:= (\widetilde{V} \varphi_\ell^0)((x,t)_i), \\ \widetilde{\mathsf{W}}_h[i,j] &:= (W \varphi_j^{10})((x,t)_i). \end{aligned} \tag{2.14}$$

Note that we do not have to explicitly assemble the matrices (2.14) in order to compute $\boldsymbol{u_h}$ and the matrix representation (2.13) is only used to write the introduced evaluation of (2.11) in multiple evaluation points in a compact form.

2.3. Computation of matrix entries

In this section we present formulas for a stable computation of the matrix entries (2.6)–(2.8) and for the evaluation of the representation formula (2.11). Due to the singularity of the fundamental solution (2.3) at $(x,t) = (y,s)$ we have to deal with weakly singular integrands. For the assembly of the boundary element matrices V_h, K_h and M_h^0 we use an element-based strategy, i.e. we loop over all pairs of boundary elements for V_h and K_h, and over boundary elements and finite elements of the initial mesh Ω_h for M_h^0. Depending on the mutual position of the two elements we use different integration routines. Let us first consider the matrix V_h. In Fig. 2.2a the integration routines for the computation of the matrix entries $\mathsf{V}_h[\ell,\cdot]$ are shown. The grid represents a part of the space-time boundary element mesh Σ_h. The element σ_ℓ is fixed and depending on where the element σ_k is located, we distinguish between the following integration routines:

A – analytic integration,

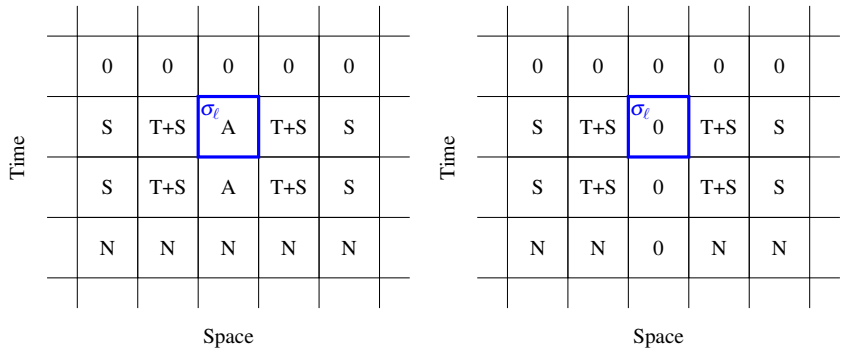
N – fully numerical integration,

S – semi-analytic integration, i.e. numerical in space and analytical in time,

T – transformation of the integral to get rid of the weak singularity.

We give a sketch of the overall situation in Fig. 2.2a. For the computation of the matrix entries corresponding to the elements marked with N, i.e. if two elements σ_ℓ and σ_k are well separated, we use numerical integration in space and time. The computation of these entries takes most of the computational time, but the evaluation of these integrals can be vectorized, see Section 3.3. The integrands corresponding to the elements marked with T have a singularity at the shared space-vertex. In these cases we transform the integrals with respect to the spatial dimensions to get rid of the weak singularity [25, 26] and then apply semi-analytic integration, i.e. numerical integration in space and analytical integration in time [27]. If the element σ_k is located above the element σ_ℓ, the value of the integral is zero due to the causality of the fundamental solution (2.3).

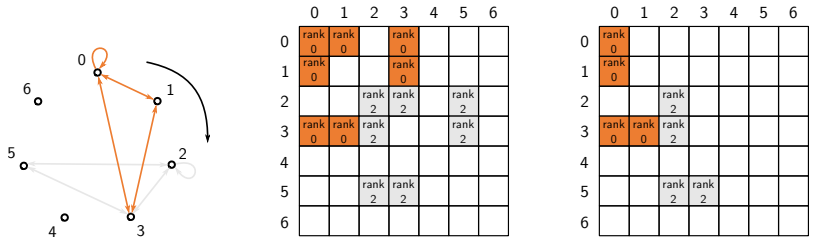
The situation is quite the same for the matrix K_h. The only difference is that the value of the integral is zero if the elements σ_ℓ and σ_k share the same spatial element γ, see Fig. 2.2b.



(a) Computation of V_h. (b) Computation of K_h.

Figure 2.2: Computation of the matrix entries $\mathsf{V}_h[\ell, \cdot]$ and $\mathsf{K}_h[\ell, \cdot]$ for a fixed boundary element σ_ℓ and varying element σ_k.

For the computation of the matrix entries of M_h^0, where we assemble a local matrix corresponding to a boundary element and a triangular element of the initial mesh, we proceed as follows. For the integral over the triangle we use the seven-point rule [28], and for the integral over the boundary element we apply semi-analytic integration, i.e. analytical in time and numerical in space. In this case we do not have to handle weakly singular integrands separately. The sparse mass matrix M_h can be assembled from local mass matrices in a standard way.



(a) Generating graph. (b) Spatial problem. (c) Space-time problem.

Figure 3.1: Distribution of the system matrix blocks among seven processes.

Similar integration techniques are used for the evaluation of the representation formula (2.11). However, since we evaluate (2.11) for $(x,t) \in Q$, we do not have to handle weakly singular integrands.

3. Parallel implementation

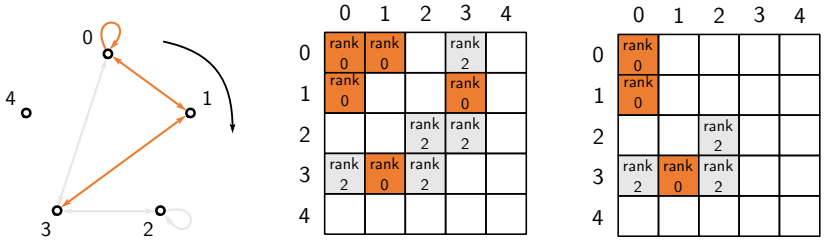
In the following sections we focus on several levels of parallelism. In Section 3.1 we start by modifying the method for the distribution of stationary BEM system matrices to support time-dependent problems. In Sections 3.2–3.3 we describe the shared-memory parallelization and vectorization of the code. Our aim is to fully utilize the capabilities of modern clusters equipped with multi- or many-core CPUs with wide SIMD registers in this way.

3.1. MPI distribution

The original method presented in [18] for spatial problems decomposes the input surface mesh into P submeshes which splits a system matrix A (the single- or double-layer operator matrix) into $P \times P$ blocks

$$\mathsf{A} = \begin{bmatrix} \mathsf{A}_{0,0} & \mathsf{A}_{0,1} & \cdots & \mathsf{A}_{0,P-1} \\ \mathsf{A}_{1,0} & \mathsf{A}_{1,1} & \cdots & \mathsf{A}_{1,P-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathsf{A}_{P-1,0} & \mathsf{A}_{P-1,1} & \cdots & \mathsf{A}_{P-1,P-1} \end{bmatrix}$$

and distributes these blocks among processes such that the number of shared mesh parts is minimal and each process owns a single diagonal block (since these usually include most of the singular entries). To find the optimal distribution, each matrix block $\mathsf{A}_{i,j}$ is regarded as an edge (i,j) of a directed complete graph K_P on P vertices. Next, a generator graph $G_0 \subset K_P$ is defined such that each oriented edge of G_0 corresponds to a block to be assembled by the process 0. The graphs $G_1, G_2, \ldots, G_{P-1}$ correspond to the remaining processes and are generated by a clock-wise rotation of G_0 along vertices of K_P placed on a circle (see Figures 3.1a–3.1b). The remaining task is to find the generating graph G_0. Optimal graph decompositions are theoretically known for special values



(a) Generating graph. (b) Spatial problem. (c) Space-time problem.

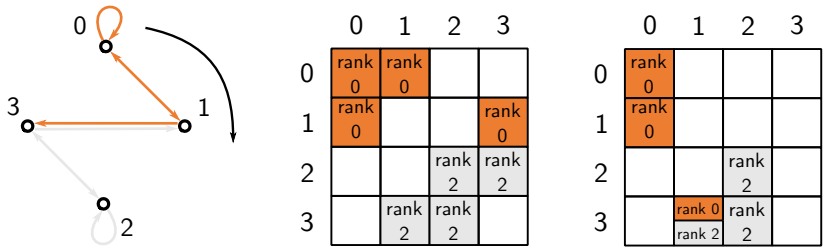
Figure 3.2: Distribution of the system matrix blocks among five processes.

of P ($P = 3, 7, 13, 21, ...$) only and are provided in [18]. Since these numbers of processes are rather unusual in high performance computing, a heuristic algorithm for finding nearly optimal decompositions for the remaining odd and even numbers of processes P is described in [19]. Notice that for odd numbers of processes the respective graph is decomposed into smaller undirected generating graphs, therefore the matrix blocks are distributed symmetrically, i.e. every process owns both blocks (i, j) and (j, i), see Figures 3.2a and 3.2b. However, when decomposing graphs for even number of processes, some edges have to be oriented and blocks are not distributed symmetrically (see Figures 3.3a–3.3b). A table with decompositions for $P = 2^k, k \in \{1, 2, \ldots, 10\}$ is presented in [19].

Adapting this method for the distribution of the matrices V_h and K_h from (2.6)–(2.7) for the time-dependent problem (2.5) is relatively straightforward. First, the space-time mesh is decomposed into slices in the temporal dimension (see Figure 2.1b). Due to the properties of the fundamental solution and the selected discrete spaces, the system matrices are block lower triangular with lower triangular blocks on the main diagonal, see (2.10). This justifies the original idea to assign a single diagonal block per process because of their different computational demands. In the case of an odd number of processes, the remaining blocks below the main diagonal are distributed according to the original scheme and the distribution of the blocks above the main diagonal is ignored (see Figures 3.1c and 3.2c). In the case of an even number of processes, the original decomposition is not symmetric, therefore some blocks have to be split between two processes (see Figure 3.3c). The construction of the generating graph ensures that each process owns exactly one shared block not influencing the load balancing. All shared blocks lie on the block subdiagonal starting with a block at the position $(P/2, 0)$.

Let us note that in [18, 19] the submatrices are approximated using the fast multipole or adaptive cross approximation methods. Here we restrict to the dense format and leave the data-sparse approximation as a topic of future work.

Next we define a distribution of the initial matrix M_h^0 from (2.8) which has a different structure from the matrices V_h and K_h. The number of its columns depends on the number of vertices of the initial mesh Ω_h, while the number of rows depends on the number of space-time elements σ_ℓ. We distribute



(a) Generating graph. (b) Spatial problem. (c) Space-time problem.

Figure 3.3: Distribution of the system matrix blocks among four processes.

whole block-rows of the matrix among processes, i.e. the initial mesh is not decomposed and the space-time mesh uses the same decomposition as for the matrices V_h and K_h. In particular, each process is responsible for the block-row corresponding to its first submesh.

The mass matrix M_h is block-diagonal, where each diagonal block represents the local mass matrix of one of the generated submeshes. These blocks are distributed among the processes. Hence each process assembles a single diagonal block corresponding to its first submesh.

It remains to establish an efficient scheme for a distributed evaluation of the discretized representation formula (2.11) in the given set of evaluation points (2.12). In order to reach a reasonable speedup we have to make the following assumption on the set of evaluations points. Recall that $\{t_k\}_{k=1}^{E_I}$ is an ordered set of time steps distributed in the interval $I = (0, T)$. We assume that each of the given time slices has the same amount of time steps E_I/P. This is necessary in order to balance the computation times between the processes.

In order to describe the parallel evaluation of (2.11) in the given set of evaluation points we consider the matrix representation (2.13). We have to distribute the matrix-vector products in an appropriate way. Therefore we split the set of evaluation points into P subsets according to the already given time slices and we obtain similar block structures for $\widetilde{\mathsf{V}}_h$, $\widetilde{\mathsf{W}}_h$ and $\widetilde{\mathsf{M}}_h^0$ as we have had for the BEM matrices V_h, K_h and M_h^0. To distribute the matrix-vector multiplication we can thus use exactly the same decomposition as for the system matrices. Note that, as already mentioned in Section 2.2, we do not have to explicitly assemble the matrices $\widetilde{\mathsf{V}}_h$, $\widetilde{\mathsf{W}}_h$ and $\widetilde{\mathsf{M}}_h^0$.

3.2. OpenMP threading

In this section we describe an efficient way of employing OpenMP threading in order to decrease the computation times of the assembly of the BEM matrices V_h, K_h, M_h^0 from (2.6)–(2.8), and the evaluation of the discretized representation formula (2.11). For better readability we consider the non-distributed system of linear equations (2.5), i.e. without the MPI distribution presented in Section 3.1. The developed scheme can be transferred to the distributed matrices created by the cyclic graph decomposition.

In order to assemble the boundary element matrices V_h and K_h we use an element-based strategy, where we loop over all pairs of space-time boundary elements, assemble a local matrix and map it to the global matrix, see Listing 3.1. OpenMP threading is employed for the outer loop over the elements. Recall that due to the given sorting of the elements of I_h both V_h and K_h are lower triangular block matrices, see (2.10). Hence the computational complexity is different for each iteration of the outer loop. Therefore, we apply dynamic scheduling and the outer loop starts with the elements located in the last time layer $N_I - 1$. The number of iterations of the inner loop, denoted with `N(l)` in Listing 3.1, depends on the current outer iteration variable `l` since we do not have to assemble the blocks in the upper triangular matrix. The function `N(l)` returns the number of boundary elements which are either located in the same time layer as the element σ_ℓ or in one of the time layers in the past. In this way we ensure that the length of the inner loop is decreasing. This is advantageous for the load balance.

```
int N(l) { return N_gamma * (1 + floor(l/N_gamma)); }

#pragma omp parallel for schedule(dynamic, 1)
for(int l = N-1; l >= 0; --l) {
  for(int k = 0; k < N(l); ++k) {
    getLocalMatrix(l, k, localMatrix);
    globalMatrix.add(l, k, localMatrix);
} }
```

Listing 3.1: Threaded element-based assembly of V_h and K_h.

The structure of the initial matrix M_h^0 is different, see Section 2.2. In order to assemble the matrix M_h^0 we again use the element-based strategy, where we loop over all boundary elements and elements of the initial mesh Ω_h, similarly as in Listing 3.1. Threading is employed for the outer loop over the boundary elements and dynamic scheduling is used again. The number of iterations of the inner loop does not depend on the index of the outer loop, since there are no vanishing entries in general compared to V_h and K_h.

A similar strategy is used for the evaluation of the discretized representation formula (2.11). We iterate over an array of evaluation points, which are sorted in the temporal direction, and, again, use dynamic scheduling, see Listing 3.2.

```
#pragma omp parallel for schedule(dynamic, 1)
for(int i = E-1; i >= 0; --i) {
  representationFormula(i, result);
}
```

Listing 3.2: Threaded evaluation of the representation formula.

All presented threading strategies can be carried over to the assembly of the blocks generated by the cyclic graph decomposition presented in Section 3.1. The main diagonal blocks of the matrices V_h and K_h have are structured as in (2.10), assuming that the time layers within the corresponding submesh are sorted appropriately. Thus, we apply the same threading strategy for the diagonal blocks as already discussed in this section. For the non-diagonal blocks of the matrices V_h and K_h we use dynamic scheduling as well, but the number

of iterations of the inner loop does not depend on the index of the outer loop anymore, since all the elements iterated over by the inner loop are located in the past of the element σ_ℓ, and therefore each pair of elements σ_k and σ_ℓ contributes to the block.

The threaded assembly of the block-rows of the initial matrix M_h^0 and the threaded evaluation of the distributed representation formula work exactly the same way as described before.

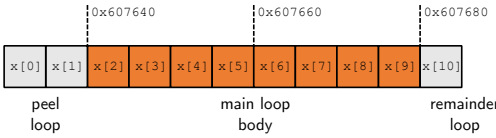
3.3. OpenMP vectorization

Let us describe the vectorization of the element matrix assembly for the single-layer matrix (2.6) which is based on numerical quadrature over pairs of space-time elements. We will limit ourselves to the case when the elements σ_ℓ and σ_k are well separated since their processing takes most of the computational time (the 'N' case from Figure 2.2a). The original scalar code consists of four nested `for` loops, two in spatial and two in temporal dimensions (see Listing 3.3). Inside each loop, coordinates of reference quadrature nodes are mapped to `x` and `y` located in the current space-time elements defined by the coordinates `xMin`, `xMax`, `yMin`, and `yMax` and the values `tMin` and `tMax`. Within the innermost loop the actual quadrature is performed using arrays of quadrature weights `w` and evaluations of the kernel function.

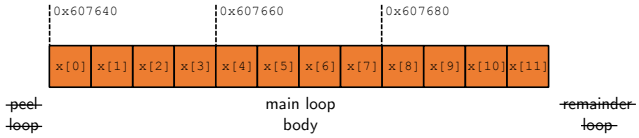
```
for ( int i = 0; i < N_GAUSS; ++i ) {
  getQuadraturePoints( x, xMin, xMax );
  for ( int j = 0; j < N_GAUSS; ++j ) {
    getQuadraturePoints( y, yMin, yMax );
    aux = innerProd( x, y );
    for ( int k = 0; k < N_GAUSS; ++k ) {
      getQuadraturePoints( t, tMin, tMax );
      for ( int l = 0; l < N_GAUSS; ++l ) {
        getQuadraturePoints( s, sMin, sMax );
        result += w[i] * w[j] * w[k] * w[l] *
          exp( -0.25 * alpha * aux / (t - s)) / (t - s);
} } } }
return result * (xMax - xMin) * (tMax - tMin)
              * (yMax - yMin) * (sMax - sMin) / ( 4.0 * M_PI );
```

Listing 3.3: Original scalar numerical quadrature over pair of space-time elements.

Since individual loops are too short to be efficiently vectorized (in our case, `N_GAUSS=4`), one cannot employ the usual and most straightforward approach of vectorizing the innermost loop. Therefore, the first step is to manually collapse the four loops into a single one with the length $\texttt{N\_GAUSS}^4$. To ensure unit-strided accesses to data within the loop the original array of quadrature weights `w` with the size `N_GAUSS` is replaced by an array `w_unrl` of length $\texttt{N\_GAUSS}^4$ containing precomputed products of four quadrature weights for each loop iteration. Similar optimization is applied to the arrays containing coordinates of quadrature points in the reference and the actual element. Moreover, to ensure unit-strided memory access patterns, we split the spatial coordinates into two separate arrays, such that `unrl_x1` and `unrl_x2` contain respectively the first and the



(a) Unaligned array – compiler creates peel and remainder loops.



(b) Aligned array – all work is performed in the main loop body. Element `x[11]` contains a dummy value.

Figure 3.4: Comparison of SIMD processing of unaligned and aligned arrays.

second spatial coordinates of the quadrature points in the current element (thus converting the data from the array of structures to structure of arrays). Memory buffers such as `unrl_x1` or `unrl_x2` are allocated on a per-thread basis (using the `threadprivate` pragma) only once during the initialization of the program. When assembling the local contribution, the arrays of actual quadrature points are filled with values in a separate vectorized loop (see Listing 3.4).

Especially when dealing with relatively short loops, it is necessary to allocate data on memory addresses which are multiples of the cache line length. When vectorizing the loop, this prevents the compiler from creating the so-called peel loop for elements stored in front of the first occurrence of such an address. For the current Intel Xeon and Xeon Phi processors the cache line size is 64 bytes and a proper alignment can be achieved using the `__attribute__((aligned(64)))` clause in the case of static allocation, or by the `_mm_malloc` method instead of `malloc` or `new` for dynamic allocation. To prevent creation of the so-called remainder loop for elements at the end of an array not filling the whole vector register, data padding can be used (see Figure 3.4). In our case, the collapsed quadrature points are padded by dummy values to fill the whole multiple of the cache line size while the quadrature weights are padded by zeroes in order not to modify the result of the numerical integration.

The actual vectorized numerical quadrature is depicted in Listing 3.5. We use the OpenMP pragma `simd` in combination with suitable clauses to assist compilers with vectorization. We inform the compiler about the memory alignment of arrays by the `aligned` clause. The `private` and `reduction` clauses have similar meaning as in OpenMP threading, and the `simdlen` clause specifies the length of a vector.

Similar optimization and vectorization techniques can be applied to the evaluation of the representation formula (2.11). In this case the local contribution consists of integration over a single space-time or spatial element and the quadra-

```
int unrl_size = N_GAUSS * N_GAUSS * N_GAUSS * N_GAUSS;

#pragma omp simd \
aligned( unrl_x_ref, unrl_y_ref, unrl_t_ref, unrl_s_ref : 64 ) \
aligned( unrl_x1, unrl_x2, unrl_y1, unrl_y2, unrl_t, unrl_s : 64 ) \
simdlen( 8 )
for ( int i = 0; i < unrl_size; ++i ) {
  unrl_x1[i] = xMin[0] + unrl_x_ref[i] * (xMax[0] - xMin[0]);
  unrl_x2[i] = xMin[1] + unrl_x_ref[i] * (xMax[1] - xMin[1]);
  ... // same for unrl_y1, unrl_y2, unrl_t, unrl_s
}
```

Listing 3.4: Vectorized mapping of quadrature nodes to a pair of space-time elements.

```
#pragma omp simd \
aligned( unrl_weights, unrl_x1, unrl_x2, unrl_y1 : 64 ) \
aligned( unrl_y2, unrl_t, unrl_s : 64 ) \
private( abs_xy_squared, ts_inv ) \
reduction( + : result ) \
simdlen( 8 )
for ( int i = 0; i < unrl_size; ++i ) {
  aux = ( unrl_x1[i] - unrl_y1[i] ) * ( unrl_x1[i] - unrl_y1[i] )
      + ( unrl_x2[i] - unrl_y2[i] ) * ( unrl_x2[i] - unrl_y2[i] );
  inv = 1.0 / ( unrl_t[i] - unrl_s[i] );
  result += unrl_w[i] * inv * exp( -0.25 * alpha * aux * inv );
}
return result * (xMax - xMin) * (tMax - tMin)
              * (yMax - yMin) * (sMax - sMin) / ( 4.0 * M_PI );
```

Listing 3.5: Vectorized numerical quadrature over a pair of space-time elements.

ture is therefore performed in two nested loops. These may be collapsed and optimized similarly as described for the system matrix assembly.

4. Numerical experiments

In this section we evaluate the efficiency of the proposed parallelization techniques. The numerical experiments for testing the shared- and distributed-memory scalability were executed on the Salomon cluster at IT4Innovations National Supercomputing Center in Ostrava, Czech Republic. The cluster is equipped with 1008 nodes with two 12-core Intel Xeon E5-2680v3 Haswell processors and 128 GB of RAM. Nodes of the cluster are interconnected by the InfiniBand 7D enhanced hypercube network. Vectorization experiments were in addition carried out on the Marconi A2 and A3 systems in Cineca, Italy, equipped respectively with one 68-core Intel Xeon Phi 7250 Knights Landing and two 24-core Intel Xeon 8160 Skylake CPUs per node supporting the AVX512 instruction set extension. The code was compiled by the Intel Compiler 2018 with the `-O3` optimization level and either `-xcore-avx2`, `-xcore-avx512 -qopt-zmm-usage=high`, or `-xmic-avx512` compiler flags respectively for the Haswell, Skylake or Knights Landing architectures.

nodes ↓	V_h assembly [s]			V_h speedup			V_h efficiency [%]		
mesh →	65k	262k	1M	65k	262k	1M	65k	262k	1M
1	138.0	—	—	1.0	—	—	100.0	—	—
2	68.4	—	—	2.0	—	—	100.9	—	—
4	33.9	—	—	4.1	—	—	101.8	—	—
8	17.7	272.0	—	7.8	1.0	—	97.5	100.0	—
16	8.6	141.1	—	16.0	1.9	—	100.3	96.4	—
32	4.5	70.0	—	30.7	3.9	—	95.8	97.1	—
64	2.3	35.0	593.1	60.8	7.8	1.0	95.0	97.1	100.0
128	—	17.7	281.7	—	15.4	2.1	—	96.0	105.3
256	—	—	145.9	—	—	4.1	—	—	101.6

Table 4.1: Assembly of V_h on 65 536, 262 144, and 1 048 576 space-time elements.

nodes ↓	K_h assembly [s]			K_h speedup			K_h efficiency [%]		
mesh →	65k	262k	1M	65k	262k	1M	65k	262k	1M
1	162.5	—	—	1.0	—	—	100.0	—	—
2	80.8	—	—	2.0	—	—	100.5	—	—
4	40.3	—	—	4.0	—	—	100.8	—	—
8	21.8	317.4	—	7.5	1.0	—	93.2	100.0	—
16	10.2	163.4	—	15.9	1.9	—	99.6	97.1	—
32	5.2	81.2	—	31.2	3.9	—	97.6	97.7	—
64	2.6	40.9	673.4	62.5	7.8	1.0	97.6	97.0	100.0
128	—	20.7	325.6	—	15.3	2.1	—	95.8	103.4
256	—	—	172.5	—	—	3.9	—	—	97.6

Table 4.2: Assembly of K_h on 65 536, 262 144, and 1 048 576 space-time elements.

All presented examples refer to the initial Dirichlet boundary value problem (2.1) on the space-time domain $Q := (0,1)^3$. We consider the exact solution

$$u(x,t) := \exp\left(-\frac{t}{\alpha}\right)\sin\left(x\cos\frac{\pi}{8} + y\sin\frac{\pi}{8}\right) \quad \text{for } (x,t) \in Q$$

and determine the Dirichlet datum g and the initial datum u_0 accordingly. The heat capacity constant is set to $\alpha = 10$. The system of linear equations (2.5) is solved by using the GMRES method with a relative precision of 10^{-8} without a preconditioner. In order to obtain the boundary and finite element meshes, we decompose the space-time boundary Σ and the domain $\Omega = (0,1)^2$ into 4 space-time-rectangles and 4 triangles, respectively, and then apply uniform refinement.

4.1. Scalability in distributed memory

In the first part of the performance experiments we focus on the parallel scalability of the proposed solver presented in Section 3.1. We tested the assem-

nodes ↓	M_h^0 assembly [s]			M_h^0 speedup			M_h^0 efficiency [%]		
mesh →	65k	262k	1M	65k	262k	1M	65k	262k	1M
1	163.7	—	—	1.0	—	—	100.0	—	—
2	82.8	—	—	2.0	—	—	98.9	—	—
4	41.0	—	—	4.0	—	—	99.8	—	—
8	20.8	332.0	—	7.9	1.0	—	98.4	100.0	—
16	10.4	167.3	—	15.7	2.0	—	98.4	99.2	—
32	5.3	83.4	—	30.9	4.0	—	96.5	99.5	—
64	2.7	42.5	687.3	60.6	7.8	1.0	94.7	97.6	100.0
128	—	21.5	343.8	—	15.4	2.0	—	96.5	100.0
256	—	—	181.4	—	—	3.8	—	—	94.7

Table 4.3: Assembly of M_h^0 on 65 536, 262 144, and 1 048 576 space-time elements and the same number of triangles in Ω_h.

bly of the BEM matrices V_h, K_h and M_h^0 from (2.5), the related matrix-vector multiplication, and the evaluation of the discrete representation formula (2.11). Strong scaling of the parallel solver was tested using a tensor product decomposition of the space-time boundary Σ into 65 536, 262 144, and 1 048 576 space-time surface elements and the same number of finite elements for the triangulation of the domain Ω. This corresponds to 512, 1 024, 2 048 spatial boundary elements and 128, 256, 512 time layers. In order to test the performance of the representation formula we chose 558 080 evaluation points for all three problem sizes. More precisely, we used a finite element mesh of the domain Ω with 545 nodes and computed the solution in these nodes in 1 024 different time steps, uniformly distributed in the interval $[0, 1]$. We used up to 256 nodes (6 144 cores) of the Salomon cluster for our computations and executed two MPI processes per node. Each MPI process used 12 cores for the assembly of the matrix blocks, for the matrix-vector multiplication, and for the evaluation of the representation formula. Note that the number of nodes we can use for our computations is restricted by the number of time layers of our boundary element mesh, i.e. starting with one element of our temporal decomposition I_h at the level $L = 0$ and using a uniform refinement strategy we end up with 2^L time layers at the level L. Thus, due to the structure of the parallel solver presented in Section 3.1 we can use 2^L MPI processes and therefore 2^{L-1} nodes at most. Conversely, for fine meshes we need a certain number of nodes to store the matrices.

In Tables 4.1–4.4 the assembly and evaluation times including the speedup and efficiency are listed. We obtain almost optimal parallel scalability of the assembly of the BEM matrices and the evaluation of the representation formula. Scalability of the matrix-vector multiplication is evaluated in Table 4.5. Since the matrix blocks are distributed, each process only multiplies with blocks it is responsible for and exchanges the result with the remaining processes. For sufficiently large problems the scalability is optimal. In the case of smaller problems, the efficiency decreases with the increasing number of compute nodes

as the communication starts to dominate over the computation. Nevertheless the efficiency is still good.

nodes ↓	$\tilde{u}$ evaluation [s]			$\tilde{u}$ speedup			$\tilde{u}$ efficiency [%]		
mesh →	65k	262k	1M	65k	262k	1M	65k	262k	1M
1	420.3	—	—	1.0	—	—	100.0	—	—
2	211.2	—	—	2.0	—	—	99.5	—	—
4	110.7	—	—	3.8	—	—	94.9	—	—
8	55.6	219.0	—	7.6	1.0	—	94.5	100.0	—
16	27.6	110.2	—	15.2	2.0	—	95.2	99.4	—
32	13.6	55.1	—	30.9	4.0	—	96.6	99.4	—
64	7.0	28.5	112.9	60.0	7.7	1.0	93.8	96.1	100.0
128	—	14.0	56.0	—	15.6	2.0	—	97.6	100.8
256	—	—	30.0	—	—	4.0	—	—	100.4

Table 4.4: Evaluation of the representation formula $\tilde{u}$ on 65 536, 262 144, and 1 048 576 space-time elements in 558 080 evaluation points.

nodes ↓	$\mathsf{V}_h\boldsymbol{f}$ time [s]			$\mathsf{V}_h\boldsymbol{f}$ speedup			$\mathsf{V}_h\boldsymbol{f}$ efficiency [%]		
mesh →	65k	262k	1M	65k	262k	1M	65k	262k	1M
1	41.9	—	—	1.0	—	—	100.0	—	—
2	22.4	—	—	1.9	—	—	93.5	—	—
4	11.3	—	—	3.7	—	—	92.7	—	—
8	5.6	89.8	—	7.5	1.0	—	93.5	100.0	—
16	2.8	45.8	—	15.0	2.0	—	93.5	98.0	—
32	1.5	22.5	—	28.1	4.0	—	87.9	99.9	—
64	0.9	11.5	182.2	46.6	7.8	1.0	72.7	97.6	100.0
128	—	6.5	96.8	—	13.8	1.9	—	86.0	94.1
256	—	—	46.0	—	—	4.0	—	—	99.0

Table 4.5: 250 matrix-vector products $\mathsf{V}_h\boldsymbol{f}$ on 65 536, 262 144, and 1 048 576 space-time elements.

4.2. Scalability in shared memory

In the second part we examine the parallel scalability in shared memory, i.e. we test the performance of the OpenMP threading introduced in Section 3.2. As before, we consider both the assembly of the BEM matrices V_h, K_h and M_h^0 as well as the evaluation of the representation formula $\tilde{u}$. The presented computation times refer to a space-time boundary element mesh Σ_h with 16 384 elements and a triangulation Ω_h consisting of 16 384 finite elements. For testing the efficiency of the parallel evaluation of $\tilde{u}$ we used a finite element mesh of Ω with 545 nodes and computed the solution in these nodes at 30 different times, i.e. in 16 350 points in total. In Table 4.6 we provide the assembly and

	# threads	1	2	4	6	8	10	12
V_h	time $[s]$	190.9	94.8	51.6	33.2	25.6	20.0	16.9
	speedup	1.0	2.0	3.7	5.8	7.5	9.5	11.3
K_h	time $[s]$	222.2	116.6	56.1	30.0	30.7	23.2	20.4
	speedup	1.0	1.9	4.0	7.4	7.2	9.6	10.9
M_h^0	time $[s]$	236.5	121.0	59.4	39.9	30.2	24.1	20.3
	speedup	1	2.0	4.0	5.9	7.8	9.8	11.7
$\tilde{u}$	time $[s]$	81.1	44.5	20.4	14.6	10.2	8.2	7.5
	speedup	1.0	1.8	4.0	5.6	8.0	9.9	10.8

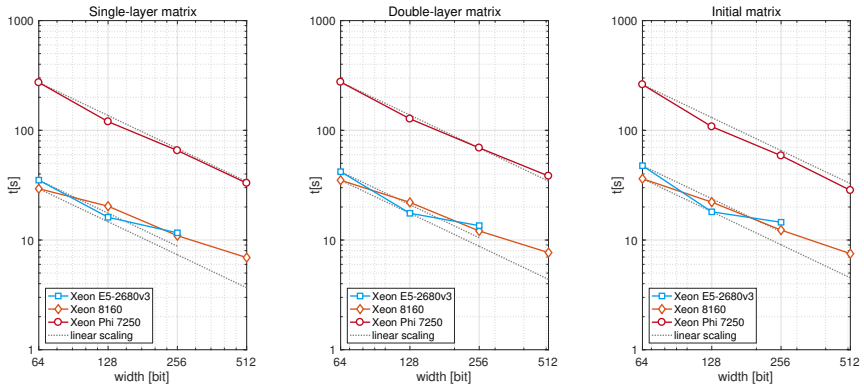
Table 4.6: Assembly and representation formula evaluation times for different numbers of OpenMP threads and a problem with 16 384 space-time surface elements, 16 384 triangles in Ω_h, and 16 350 evaluation points.

evaluation times for different numbers of OpenMP threads. Since each MPI process uses 12 cores for its computations, we limited the maximum number of threads in the experiments to 12. On the multi-core Xeon processors of the Salomon cluster we obtain the almost optimal speedup of 11.3 (10.9, 11.7) for the assembly of the BEM matrices and the speedup of 10.8 for the evaluation of the representation formula.

4.3. Vectorization efficiency

Efficiency of the vectorized system matrix assembly on several architectures was tested on a mesh consisting of 4 096 surface space-time elements. In Figure 4.1, the scalability of the vectorization is depicted with respect to the width of the SIMD vector. The scalar version (64-bits vector width) was compiled with `-no-vec -no-simd -qno-openmp-simd` in addition to the vectorization flag, the remaining widths of the vectors were set during the runtime by the `simdlen` OpenMP clause, see Listings 3.4–3.5. The tests presented in Figure 4.1 were carried out using a single thread in order to minimize the effects of frequency scaling when running an AVX512 code on multiple cores. We obtain almost optimal scaling on the Xeon Phi 7250 processor; the code running on Xeon CPUs is less efficient, however, still scales reasonably well.

In Tables 4.7–4.8 we compare the speedup of vectorization using either a single core or the maximum number of physical cores per socket. Here, AVX512$(\cdot)$ or AVX2$(\cdot)$ refers to the length of the vector set by the `simdlen(·)` clause. One can observe a drop in the speedup when using multiple cores per socket which is especially prominent in the case of AVX512 on Xeon Phi 7250 and Xeon 8160. In this case, the core frequency may drop significantly since the AVX512 instructions consume a large amount of energy [29]. Moreover, a simultaneous access of several threads to the main memory may influence the performance as well.



(a) V_h. (b) K_h. (c) M_h^0.

Figure 4.1: Scalability of the matrix assembly with respect to the SIMD vector width.

5. Conclusion

In the paper, we have presented a parallel space-time boundary element solver for the heat equation. The solver is parallelized using MPI in the distributed memory, OpenMP is used for the shared memory parallelization and vectorization. The distribution of the system matrices among computational nodes is based on the method presented in [18, 19] for spatial problems. We have successfully adapted the method to support the time-dependent problem for the heat equation. A space-time computational mesh is decomposed into slices which inherently define blocks in the system matrices. These blocks are distributed among MPI processes using the graph-decomposition-based scheme. The numerical experiments show optimal scalability of the global system matrix assembly in distributed memory and almost optimal scalability of the individual blocks assembly in shared memory. An additional performance gain is obtained using SIMD vectorization. We have also demonstrated distributed-memory scalability of the matrix-vector multiplication and the evaluation of the representation formula.

The presented method provides opportunities for further research and development of numerical methods. While in [18, 19] the individual matrix blocks are approximated using either the adaptive cross approximation or the fast multipole method, we limited ourselves to classical BEM leading to dense system matrices. Their data-sparse approximation is a topic of future work. Together with data-sparse methods the developed technology will serve as a base for the development of a parallel fast three-dimensional solver.

architecture	threads	matrix	AVX512(2)	AVX512(4)	AVX512(8)
Xeon Phi 7250	1	V_h	2.27	4.17	7.79
		K_h	2.16	3.98	7.18
		M_h^0	2.42	4.45	9.20
	68	V_h	2.18	3.84	6.52
		K_h	2.07	3.61	6.02
		M_h^0	2.35	4.18	7.86
Xeon 8160	1	V_h	1.44	2.68	4.24
		K_h	1.59	2.89	4.57
		M_h^0	1.63	2.94	4.82
	24	V_h	1.40	2.46	3.75
		K_h	1.54	2.57	3.92
		M_h^0	1.60	2.73	4.28

Table 4.7: Speedup of the AVX512 code with respect to the scalar baseline.

architecture	threads	matrix	AVX2(2)	AVX2(4)
Xeon E5-2680v3	1	V_h	2.18	3.02
		K_h	2.39	3.10
		M_h^0	2.63	3.27
	12	V_h	2.09	2.74
		K_h	2.34	2.95
		M_h^0	2.66	3.25

Table 4.8: Speedup of the AVX2 code with respect to the scalar baseline.

Acknowledgements

The research was supported by the project 'Efficient parallel implementation of boundary element methods' provided jointly by the Ministry of Education, Youth and Sports (7AMB17AT028) and OeAD (CZ 16/2017). SD acknowledges the support provided by the International Research Training Group 1754, funded by the German Research Foundation (DFG) and the Austrian Science Fund (FWF). JZ an MM further acknowledge the support provided by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPS II) project 'IT4Innovations excellence in science – LQ1602' and by the IT4Innovations infrastructure which is supported from the Large Infrastructures for Research, Experimental Development and Innovations project 'IT4Innovations National Supercomputing Center – LM2015070'.

References

[1] P. Noon, The Single Layer Heat Potential and Galerkin Boundary Element Methods for the Heat Equation., Thesis, University of Maryland (1988).

[2] D. N. Arnold, P. J. Noon, Coercivity of the single layer heat potential, J. Comput. Math. 7 (2) (1989) 100–104.
URL www.jstor.org/stable/43692419

[3] M. Costabel, Boundary integral operators for the heat equation., Integral Equations and Operator Theory 13 (1990) 498–552. doi:10.1007/BF01210400.

[4] G. C. Hsiao, J. Saranen, Boundary integral solution of the two-dimensional heat equation, Math. Methods Appl. Sci. 16 (2) (1993) 87–114. doi:10.1002/mma.1670160203.

[5] M. Costabel, Time–dependent problems with the boundary integral equation method, in: E. Stein, R. de Borst, T. J. R. Hughes (Eds.), Encyclopedia of Computational Mechanics, John Wiley & Sons, 2004, pp. 703–721. doi:10.1002/0470091355.ecm022.

[6] C. Lubich, R. Schneider, Time discretization of parabolic boundary integral equations, Numer. Math. 63 (4) (1992) 455–481. doi:10.1007/BF01385870.

[7] R. Chapko, R. Kress, Rothe's method for the heat equation and boundary integral equations, J. Integral Equations Appl. 9 (1) (1997) 47–69. doi:10.1216/jiea/1181075987.

[8] J. Tausch, Nyström discretization of parabolic boundary integral equations, Appl. Numer. Math. 59 (11) (2009) 2843–2856. doi:10.1016/j.apnum.2008.12.032.

[9] M. Costabel, J. Saranen, The spline collocation method for parabolic boundary integral equations on smooth curves, Numer. Math. 93 (3) (2003) 549–562. doi:10.1007/s002110200405.

[10] M. Messner, M. Schanz, J. Tausch, A fast Galerkin method for parabolic space–time boundary integral equations, J. Comput. Phys. 258 (2014) 15–30. doi:10.1016/j.jcp.2013.10.029.

[11] M. Messner, A Fast Multipole Galerkin Boundary Element Method for the Transient Heat Equation., Monographic Series TU Graz: Computation in Engineering and Science 23. doi:10.3217/978-3-85125-350-4.

[12] M. Messner, M. Schanz, J. Tausch, An efficient Galerkin boundary element method for the transient heat equation, SIAM J. Sci. Comput. 37 (3) (2015) A1554–A1576. doi:10.1137/151004422.

[13] J. Dongarra, et al., The international exascale software project roadmap, The International Journal of High Performance Computing Applications 25 (1) (2011) 3–60. doi:10.1177/1094342010391989.

[14] R. Speck, D. Ruprecht, M. Emmett, M. Minion, M. Bolten, R. Krause, A space-time parallel solver for the three-dimensional heat equation, in: Parallel Computing: Accelerating Computational Science and Engineering (CSE), Vol. 25 of Advances in Parallel Computing, IOS Press, 2014, pp. 263 – 272. doi:10.3233/978-1-61499-381-0-263.

[15] J. Dongarra, J. Hittinger, J. Bell, L. Chacon, R. Falgout, M. Heroux, P. Hovland, E. Ng, C. Webster, S. Wild, Applied mathematics research for exascale computing, Tech. rep., U.S. Department of Energy (2014). doi:10.2172/1149042.

[16] J.-L. Lions, Y. Maday, G. Turinici, Résolution d'edp par un schéma en temps « pararéel », Comptes Rendus de l'Académie des Sciences - Series I - Mathematics 332 (7) (2001) 661 – 668. doi:10.1016/S0764-4442(00)01793-6.

[17] M. Gander, M. Neumüller, Analysis of a new space-time parallel multigrid algorithm for parabolic problems, SIAM Journal on Scientific Computing 38 (4) (2016) A2173–A2208. doi:10.1137/15M1046605.

[18] D. Lukas, P. Kovar, T. Kovarova, M. Merta, A parallel fast boundary element method using cyclic graph decompositions, Numerical Algorithms 70 (4) (2015) 807–824. doi:10.1007/s11075-015-9974-9.

[19] M. Kravcenko, M. Merta, J. Zapletal, Distributed fast boundary element methods for Helmholtz problems, submitted (2018).

[20] M. Kretz, V. Lindenstruth, Vc: A C++ library for explicit vectorization, Software: Practice and Experience 42 (11) (2012) 1409–1430. doi:10.1002/spe.1149.

[21] Intel Corporation, Vector Mathematical Functions, [Online; accessed 29-August-2018] (2018).
URL https://software.intel.com/en-us/mkl-developer-reference-c-vector-mathematical-functions

[22] OpenMP Application Programming Interface, [Online; accessed 29-August-2018] (2015).
URL https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf

[23] D. N. Arnold, P. J. Noon, Boundary integral equations of the first kind for the heat equation, in: Boundary elements IX, Vol. 3 (Stuttgart, 1987), Comput. Mech., Southampton, 1987, pp. 213–229.

[24] J. L. Lions, E. Magenes, Non-Homogeneous Boundary Value Problems and Applications. Volume II, Springer, Berlin-Heidelberg-New York, 1972.

[25] S. Sauter, C. Schwab, Boundary Element Methods, Springer, Berlin-Heidelberg, 2011.

[26] G. C. Hsiao, P. Kopp, W. L. Wendland, A Galerkin collocation method for some integral equations of the first kind, Computing 25 (2) (1980) 89–130. doi:10.1007/BF02259638.

[27] F. Sgallari, A weak formulation of boundary integral equations for time dependent parabolic problems, Applied Mathematical Modelling 9 (4) (1985) 295–301. doi:10.1016/0307-904X(85)90068-X.

[28] J. Radon, Zur mechanischen Kubatur, Monatsh. Math. 52 (4) (1948) 286–300. doi:10.1007/BF01525334.

[29] Xeon Platinum 8160 – Intel, [Online; accessed 26-September-2018] (2017). URL https://en.wikichip.org/wiki/intel/xeon_platinum/8160

Erschienene Preprints ab Nummer 2015/1

2015/1 O. Steinbach: Space-time finite element methods for parabolic problems

2015/2 O. Steinbach, G. Unger: Combined boundary integral equations for acoustic scattering-resonance problems problems.

2015/3 C. Erath, G. Of, F.–J. Sayas: A non-symmetric coupling of the finite volume method and the boundary element method

2015/4 U. Langer, M. Schanz, O. Steinbach, W.L. Wendland (eds.): 13th Workshop on Fast Boundary Element Methods in Industrial Applications, Book of Abstracts

2016/1 U. Langer, M. Schanz, O. Steinbach, W.L. Wendland (eds.): 14th Workshop on Fast Boundary Element Methods in Industrial Applications, Book of Abstracts

2016/2 O. Steinbach: Stability of the Laplace single layer boundary integral operator in Sobolev spaces

2017/1 O. Steinbach, H. Yang: An algebraic multigrid method for an adaptive space-time finite element discretization

2017/2 G. Unger: Convergence analysis of a Galerkin boundary element method for electromagnetic eigenvalue problems

2017/3 J. Zapletal, G. Of, M. Merta: Parallel and vectorized implementation of analytic evaluation of boundary integral operators

2017/4 S. Dohr, O. Steinbach: Preconditioned space-time boundary element methods for the one-dimensional heat equation

2017/5 O. Steinbach, H. Yang: Comparison of algebraic multigrid methods for an adaptive space–time finite element discretization of the heat equation in 3D and 4D

2017/6 S. Dohr, K. Niino, O. Steinbach: Preconditioned space-time boundary element methods for the heat equation

2017/7 O. Steinbach, M. Zank: Coercive space-time finite element methods for initial boundary value problems

2017/8 U. Langer, M. Schanz, O. Steinbach, W.L. Wendland (eds.): 15th Workshop on Fast Boundary Element Methods in Industrial Applications, Book of Abstracts

2018/1 U. Langer, M. Schanz, O. Steinbach, W.L. Wendland (eds.): 16th Workshop on Fast Boundary Element Methods in Industrial Applications, Book of Abstracts

2018/2 S. Dohr, J. Zapletal, G. Of, M. Merta, M. Kravcenko: A parallel space-time boundary element method for the heat equation