Technische Universität Graz

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

R. Watschinger, M. Merta, G. Of, J. Zapletal

Berichte aus dem Institut für Angewandte Mathematik

Bericht 2021/11

Technische Universität Graz

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

R. Watschinger, M. Merta, G. Of, J. Zapletal

Berichte aus dem Institut für Angewandte Mathematik

Bericht 2021/11

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

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

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

Raphael Watschinger[a], Michal Merta[b,c,*], Günther Of[a], Jan Zapletal[b,c]

[a] *Institute of Applied Mathematics, Graz University of Technology. Steyrergasse 30, A-8010 Graz, Austria*
[b] *IT4Innovations, VSB – Technical University of Ostrava. 17. listopadu 2172/15, 708 00 Ostrava-Poruba, Czech Republic*
[c] *Department of Applied Mathematics, VSB – Technical University of Ostrava. 17. listopadu 2172/15, 708 00 Ostrava-Poruba, Czech Republic*

Abstract

We present a novel approach to the parallelization of the parabolic fast multipole method for a space-time boundary element method for the heat equation. We exploit the special temporal structure of the involved operators to provide an efficient distributed parallelization with respect to time and with a one-directional communication pattern. On top, we apply a task-based shared memory parallelization and SIMD vectorization. In the numerical tests we observe high efficiencies of our parallelization approach.

Keywords: boundary element method, space-time, heat equation, FMM, parallelization, HPC
2010 MSC: 65M38, 65Y05, 35K05

1. Introduction

Space-time methods have become a popular subject of research in recent years, e.g., see the proceedings [1]. Current advances in high performance computing (HPC) have facilitated this trend and both fields of research have benefited mutually. On the one hand, increasing computing power enables the solution of huge systems occurring in real-world problems. On the other hand, global space-time matrices allow to develop solvers with better parallel scalability. Classical approaches to a parallel solution of time-dependent partial differential equations use some decomposition in space together with time-stepping algorithms which are sequential with respect to time and thus limit the parallelization to the spatial components. Therefore, parallel-in-time algorithms, such as parareal [2] or space-time parallel multigrid [3] have gained popularity recently, as they enable parallelization in both spatial and temporal dimensions and restore scalability on large numbers of CPUs. Boundary element methods (BEM) have certain advantages over volume-based methods (such as the finite element method) with respect to parallelization. Because of its high computational intensity and the dense structure of its system matrices, the use of BEM can help to leverage the full potential of modern many-core CPUs equipped with wide SIMD (Single Instruction Multiple Data) registers or GPU accelerators.

*Corresponding author
Email address: michal.merta@vsb.cz (Michal Merta)

Space-time boundary element methods for the solution of the transient heat equation have been known for a relatively long time [4, 5, 6]. These methods describe solutions by boundary data only, which reduces the number of unknowns significantly compared to volume-based methods. However, the corresponding system matrices are dense which leads to high computational and memory complexities of a standard BEM. Therefore, several fast and data-sparse algorithms (such as Fourier series and FFT [7, 8], the parabolic FMM [9, 10], or a fast sparse grid method [11]) have been developed to provide efficient solvers with almost linear complexity. The parabolic fast multipole method (pFMM) has originally been described for Nyström discretizations [9, 10] and has been extended to Galerkin discretizations [12, 13] later on. It is based on a clustering of the computational domain in both space and time and an approximation of interactions in well-separated clusters by truncated series expansions. The resulting method can be seen as a combination of a one-dimensional FMM in time and fast Gauss transforms in space. Commonly, the related space-time system is solved by some sort of block forward elimination successively, i.e., like a sequential time-stepping scheme. While this is beneficial in terms of memory requirements, the parallelization is limited to the spatial components.

Many publications have been devoted to the efficient implementation and parallelization of the FMM (see, e.g., [14, 15, 16, 17, 18, 19]) mainly for particle simulations, but also for the solution of classical spatial boundary integral equations [20, 21], and, less frequently, space-time boundary integral equations [22]. A parallelization of a standard Galerkin space-time BEM for the heat equation in two spatial dimensions was considered in [23].

Here we present a novel parallel version of a space-time FMM for a Galerkin space-time BEM for the heat equation in three spatial dimensions based on the pFMM. We enable parallelization in time by solving the whole space-time system at once. Instead of simply transferring one of the parallelization approaches of spatial FMMs to our setting, we use a different concept to tailor our parallel algorithm to the special structure of the pFMM. The presented method employs a distribution of the space-time cluster tree with respect to time among MPI (Message Passing Interface) processes. In particular, we exploit the causality of the operators which leads to a one-directional communication in the temporal component. In addition, the computation on individual processes is parallelized in shared memory using OpenMP tasks with explicitly stated dependencies [24].

Our parallelization approach in shared and distributed memory is based on a data driven model, instead of a bulk-synchronous parallelization often used in scientific codes. We use a two-level task-based concept for the parallelization of the space-time FMM. We group FMM operations as tasks along the underlying temporal tree which reflects the communication pattern of our distribution. This provides a first coarse granularity to represent the dependencies of the FMM data/operations and to steer the load balancing and the MPI parallelization. By executing tasks (grouped FMM operations) based on individual dependencies, we overcome the strict separation of phases of classical FMM and can arrange the grouped operations more flexibly. This allows us to hide communication and to fill eventual idle times by independent tasks. When it comes to the execution of temporal tasks on a single MPI process we generate a fine granularity by creating multiple tasks for the related space-time operations by OpenMP. Here the OpenMP tasks scheduler can serve as a buffer for the created OpenMP tasks keeping the cores busy during the computation.

Our numerical experiments cover SIMD vectorization, shared memory performance using up to 36 cores, and scalability tests of the distributed memory parallelization with up to 256 MPI processes (6144 cores). In these tests we observe high efficiencies of our parallelization approach.

The structure of this paper is as follows. In Section 2 we briefly describe the boundary integral formulation of the Dirichlet initial boundary value problem of the heat equation and its discretization using space-time BEM. We provide a comprehensive description of the space-time FMM for a Galerkin BEM in Section 3. This serves as a basis for the detailed presentation of our parallelization concept in Section 4. Finally, numerical experiments and conclusions are provided in Section 5 and Section 6.

2. Boundary integral formulation of the heat equation and its discretization

We consider the Dirichlet boundary value problem of the transient heat equation with the heat capacity constant $\alpha > 0$ and zero initial condition for a bounded Lipschitz domain $\Omega \subset \mathbb{R}^3$ with boundary $\Gamma = \partial\Omega$ as a model problem:

$$\begin{aligned} \frac{\partial}{\partial t}u(\boldsymbol{x},t) - \alpha\Delta u(\boldsymbol{x},t) &= 0 && \text{for } (\boldsymbol{x},t) \in \Omega \times (0,T), \\ u(\boldsymbol{x},0) &= 0 && \text{for } \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x},t) &= g(\boldsymbol{x},t) && \text{for } (\boldsymbol{x},t) \in \Sigma := \Gamma \times (0,T). \end{aligned}$$

The solution of this problem can be described by the representation formula

$$u(\boldsymbol{x},t) = \widetilde{V}w(\boldsymbol{x},t) - Wu(\boldsymbol{x},t) \qquad \text{for } (\boldsymbol{x},t) \in \Omega \times (0,T)$$

with the conormal derivative $w := \alpha\frac{\partial u}{\partial \boldsymbol{n}}$, the single layer potential

$$\widetilde{V}w(\boldsymbol{x},t) := \int_0^t \int_\Gamma G_\alpha(\boldsymbol{x}-\boldsymbol{y}, t-\tau) w(\boldsymbol{y},\tau)\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}\,\mathrm{d}\tau,$$

the double layer potential

$$Wu(\boldsymbol{x},t) := \int_0^t \int_\Gamma \alpha\frac{\partial G_\alpha}{\partial \boldsymbol{n}_{\boldsymbol{y}}}(\boldsymbol{x}-\boldsymbol{y}, t-\tau) u(\boldsymbol{y},\tau)\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}\,\mathrm{d}\tau,$$

and the heat kernel

$$G_\alpha(\boldsymbol{x}-\boldsymbol{y}, t-\tau) = \begin{cases} (4\pi\alpha(t-\tau))^{-3/2} \exp\left(-\dfrac{|\boldsymbol{x}-\boldsymbol{y}|^2}{4\alpha(t-\tau)}\right) & \text{for } \tau < t, \\ 0 & \text{for } \tau > t. \end{cases} \tag{2.1}$$

While the Dirichlet datum $u = g$ is given on Σ, the unknown Neumann datum w can be determined from the boundary integral equation

$$Vw(\boldsymbol{x},t) = \left(\frac{1}{2}I + K\right) g(\boldsymbol{x},t) \qquad \text{for almost all } (\boldsymbol{x},t) \in \Sigma \tag{2.2}$$

with the single and double layer boundary integral operators

$$\begin{aligned} Vw(\boldsymbol{x},t) &= \int_0^t \int_\Gamma G_\alpha(\boldsymbol{x}-\boldsymbol{y}, t-\tau) w(\boldsymbol{y},\tau)\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}\,\mathrm{d}\tau, \\ Ku(\boldsymbol{x},t) &= \int_0^t \int_\Gamma \alpha\frac{\partial G_\alpha}{\partial \boldsymbol{n}_{\boldsymbol{y}}}(\boldsymbol{x}-\boldsymbol{y}, t-\tau) u(\boldsymbol{y},\tau)\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}\,\mathrm{d}\tau. \end{aligned}$$

As the Neumann datum w is unknown, we compute some numerical approximation. Typically a tensor product mesh is considered for uniform time-steps, $t_j = jh_t$, $j = 0, \dots, E_t$, and a fixed spatial surface mesh $\{\gamma_{j_{\boldsymbol{x}}}\}_{j_{\boldsymbol{x}}=1}^{E_{\boldsymbol{x}}}$ of triangles. For such a decomposition Σ_h of the lateral boundary $\Sigma = \Gamma \times (0,T)$ into space-time boundary elements $\sigma_{j_t,j_{\boldsymbol{x}}} = \gamma_{j_{\boldsymbol{x}}} \times (t_{j_t-1}, t_{j_t})$, a standard approximation of w is given by a piecewise constant approximation

$$w_h(\boldsymbol{y},\tau) = \sum_{j_t=1}^{E_t} \sum_{j_{\boldsymbol{x}}=1}^{E_{\boldsymbol{x}}} w_{j_t,j_{\boldsymbol{x}}} \varphi_{j_t,j_{\boldsymbol{x}}}^{0,0}(\boldsymbol{y},\tau)$$

with basis functions $\varphi_{j_t,j_{\boldsymbol{x}}}^{0,0}$ which are one on a space-time boundary element $\sigma_{j_t,j_{\boldsymbol{x}}}$ and zero otherwise.

To find the yet unknown coefficients $w_{j_t,j_{\boldsymbol{x}}}$ we consider the Galerkin variational formulation

$$\int_{t_{k_t-1}}^{t_{k_t}} \int_{\gamma_{k_{\boldsymbol{x}}}} V w_h(\boldsymbol{x},t) ds_{\boldsymbol{x}} dt = \int_{t_{k_t-1}}^{t_{k_t}} \int_{\gamma_{k_{\boldsymbol{x}}}} \left(\frac{1}{2} I + K\right) g_h(\boldsymbol{x},t) \mathrm{d}\boldsymbol{s}_{\boldsymbol{x}} \, \mathrm{d}t$$

for all $k_t = 1, \dots, E_t$ and $k_{\boldsymbol{x}} = 1, \dots, E_{\boldsymbol{x}}$. Note that we have replaced the Dirichlet datum g by a $L_2(\Sigma)$ projection g_h, which is piecewise constant in time, but piecewise linear and globally continuous in space. The equivalent system of linear equations to find w_h or rather the vector $\boldsymbol{w}$ of coefficients $w_{j_t,j_{\boldsymbol{x}}}$ is

$$\mathsf{V}_h \boldsymbol{w} = \left(\frac{1}{2}\mathsf{M}_h + \mathsf{K}_h\right)\boldsymbol{g}, \tag{2.3}$$

where the matrix entries are given by

$$\mathsf{V}[(k_t-1)E_{\boldsymbol{x}} + k_{\boldsymbol{x}}, (j_t-1)E_{\boldsymbol{x}} + j_{\boldsymbol{x}}] = \int_{t_{k_t-1}}^{t_{k_t}} \int_{\gamma_{k_{\boldsymbol{x}}}} \int_{t_{j_t-1}}^{t_{j_t}} \int_{\gamma_{j_{\boldsymbol{x}}}} G_\alpha(\boldsymbol{x}-\boldsymbol{y}, t-\tau) \, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}} \, \mathrm{d}\tau \, \mathrm{d}\boldsymbol{s}_{\boldsymbol{x}} \, \mathrm{d}t$$

for $k_{\boldsymbol{x}}, j_{\boldsymbol{x}} = 1, \dots, E_{\boldsymbol{x}}$ and $k_t, j_t = 1, \dots, E_t$. Due to the causality of the heat kernel this matrix has lower triangular block structure. K_h is defined similarly but for trial functions which are piecewise constant in time, piecewise linear and globally continuous in space. M_h denotes the related mass matrix. Please check [25] for details on the discretization and implementation. Some detailed analysis of the integral equations and the presented boundary element method is provided in [5, 26].

Note that system (2.3) is huge and that the matrices V_h and K_h are dense except for their lower triangular block structure. Thus a standard BEM is limited to small examples. In general a data-sparse method such as the FMM is necessary to compute large-scale examples.

3. The sequential space-time FMM algorithm for the heat equation

Following [9, 10, 12] we first present a sequential space-time FMM for the multiplication of a vector $\boldsymbol{w}$ by the single layer operator matrix V_h before dealing with its parallelization. The considered FMM is based on a suitable expansion of the heat kernel G_α and a clustering of the computational domain, and can also be applied to other BEM matrices like K_h in (2.3) with slight modifications. In the parabolic fast multipole method in [9, 10, 12] the multiplication is executed in a forward-sweeping manner, based on the causality of the operators. However, this does not allow for a parallelization in time, which is our aim. Thus, we consider the method more like a standard FMM but highlight its special structure with respect to time, which forms the basis of our parallelization strategy.

3.1. A separable approximation of the heat kernel

Throughout the section we regard the heat kernel as a function of the differences $\boldsymbol{x}-\boldsymbol{y}$ and $t-\tau>0$ as in (2.1). We restrict the variables $(\boldsymbol{x},t)$ to a 4D target box $Z_1=X\times I$ and $(\boldsymbol{y},\tau)$ to a 4D source box $Z_2=Y\times J$, where $I=(c_4,c_4+2\tilde{h}_t]$ and $J=(d_4,d_4+2\tilde{h}_t]$ are two intervals of length $2\tilde{h}_t$ such that $c_4>d_4$ and $\operatorname{dist}(I,J)=c_4-d_4-2\tilde{h}_t>0$, and $X=(\boldsymbol{c},\boldsymbol{c}+2\tilde{h}_x\mathbf{1}]$ and $Y=(\boldsymbol{d},\boldsymbol{d}+2\tilde{h}_x\mathbf{1}]$ are cubes in $\mathbb{R}^3$ with edge length $2\tilde{h}_x$. Here we use the notation

$$(\boldsymbol{a},\boldsymbol{b}]:=(a_1,b_1]\times(a_2,b_2]\times(a_3,b_3].$$

For $(\boldsymbol{x},t)\in Z_1$ and $(\boldsymbol{y},\tau)\in Z_2$ there holds $t-\tau\geq\operatorname{dist}(I,J)>0$ and thus the heat kernel is smooth. As in [9, 10, 12] we interpolate it in the temporal intervals I and J and approximate it in the spatial boxes X and Y by means of a truncated Chebyshev expansion.

For this purpose, let $T_k(x)=\cos(k\arccos(x))$ be the Chebyshev polynomials of order k on $[-1,1]$ and let $\{\xi_k^{(m_t)}\}_{k=0}^{m_t}$ be the Chebyshev nodes of order m_t+1 in $[-1,1]$, i.e., the roots of T_{m_t+1}. On an interval $I=(a,b]$ we consider the transformed points $\xi_{I,k}^{(m_t)}:=\varphi_I(\xi_k^{(m_t)})$, where φ_I is the affine transformation from $(-1,1]$ to I, and the associated Lagrange polynomials

$$L_{I,b}(t):=\prod_{k\neq b}\frac{t-\xi_{I,k}^{(m_t)}}{\xi_{I,b}^{(m_t)}-\xi_{I,k}^{(m_t)}}\qquad\text{for all } b\in\{1,...,m_t+1\}.$$

For the expansion in space we use the transformed Chebyshev polynomials $T_{(a,b],k}:=T_k\circ\varphi_{(a,b]}^{-1}$ on intervals $(a,b]$ and their tensor products given by $T_{(\boldsymbol{a},\boldsymbol{b}],\boldsymbol{\kappa}}(\boldsymbol{x}):=\prod_j T_{(a_j,b_j],\kappa_j}(x_j)$, where $\boldsymbol{\kappa}$ is a multi-index in $\mathbb{N}_0^3$. By interpolating G_α in the temporal points $\{\xi_{I,k}^{(m_t)}\}_k$ and $\{\xi_{J,k}^{(m_t)}\}_k$ and approximating the resulting function via a truncated Chebyshev expansion in X and Y we get

$$G_\alpha(\boldsymbol{x}-\boldsymbol{y},t-\tau)\approx\sum_{a,b=0}^{m_t}\sum_{|\boldsymbol{\kappa}+\boldsymbol{\nu}|\leq m_{\boldsymbol{x}}}E_{a,\boldsymbol{\kappa},b,\boldsymbol{\nu}}T_{X,\boldsymbol{\nu}}(\boldsymbol{x})T_{Y,\boldsymbol{\kappa}}(\boldsymbol{y})L_{I,b}(t)L_{J,a}(\tau)\tag{3.1}$$

in $Z_1\times Z_2$. Here, $m_{\boldsymbol{x}}\geq 0$ is the expansion order in space, and

$$E_{a,\boldsymbol{\kappa},b,\boldsymbol{\nu}}=\frac{1}{(4\pi\alpha(\xi_{I,b}^{(m_t)}-\xi_{J,a}^{(m_t)}))^{3/2}}\prod_{j=1}^{3}E_{\kappa_j,\nu_j}(r_j,d_{a,b})$$

are the expansion coefficients, where

$$r_j:=(c_j-d_j)/\tilde{h}_x,\tag{3.2}$$

$$d_{a,b}:=4\alpha(\xi_{I,b}^{(m_t)}-\xi_{J,a}^{(m_t)})/\tilde{h}_x^2,\tag{3.3}$$

$$E_{k,\ell}(r,d_{a,b})=\frac{\lambda_k\lambda_l}{(m_{\boldsymbol{x}}+1)^2}\sum_{n,m=0}^{m_{\boldsymbol{x}}}\exp\left(-\frac{|r+\xi_n^{(m_{\boldsymbol{x}})}-\xi_m^{(m_{\boldsymbol{x}})}|^2}{d_{a,b}}\right)T_\ell(\xi_n^{(m_{\boldsymbol{x}})})T_k(\xi_m^{(m_{\boldsymbol{x}})})\tag{3.4}$$

and $\lambda_0=1$, $\lambda_k=2$ for all $k>0$, cf. [10, Section 5.3, page 209].

Let us comment on the approximation quality of (3.1). The temporal interpolation error converges exponentially to zero with respect to the interpolation degree m_t if the time intervals are well-separated, see e.g., [27, Lemma 4.1 and Equation (4.42)]. For the truncated Chebyshev

expansion one can show super-exponential convergence of the approximation error without requiring a separation of the spatial boxes X and Y, see [10, Section 5.3, page 209]. However, the effective approximation quality suffers for small values of $d_{a,b}$. Therefore, we bound $d_{a,b}$ from below in the later application by choosing the spatial box half-size $\tilde{h}_x$ for a given temporal interval half-size $\tilde{h}_t$ such that

$$\frac{\tilde{h}_x^2}{4\alpha\,\tilde{h}_t} \leq c_{\mathrm{st}} \tag{3.5}$$

for some constant $c_{\mathrm{st}} > 0$ [10, cf. ρ in (27) and Section 5.4]. Since $(\xi_{I,b}^{(m_t)} - \xi_{J,a}^{(m_t)}) \geq \operatorname{dist}(I, J)$, (3.5) implies that $d_{a,b} \gtrsim c_{\mathrm{st}}^{-1}$ if we guarantee that $\operatorname{dist}(I, J) \gtrsim \tilde{h}_t$, i.e., I and J are well-separated.

An important observation is that the heat kernel decays exponentially in space for fixed temporal variables. Therefore, we do not have to approximate the values of the kernel in boxes $Z_1 = X \times I$ and $Z_2 = Y \times J$ but can instead neglect them if the distance of the spatial boxes X and Y is large compared to the distance of the time intervals I and J.

3.2. A 4D space-time box cluster tree

For the FMM algorithm we establish a hierarchy of 4D boxes to partition the space-time tensor mesh Σ_h appropriately. The resulting structure is denoted as box cluster tree $\mathcal{T}_\Sigma$. Our approach is similar to the one in [9, 12]. However, instead of building separate trees in space and time first, we directly establish a 4D space-time tree, which is a more general approach applicable also to space-time meshes without a strict tensor product structure. For the construction we use a recursive refinement strategy which is sketched in Algorithm 1.

Algorithm 1 Construction of a 4D space-time box cluster tree $\mathcal{T}_\Sigma$

1: Let a space-time tensor mesh Σ_h inside a 4D space-time box $Z^{(0)} = (\boldsymbol{a}, \boldsymbol{a} + 2h_x^{(0)}\mathbf{1}] \times (0, T]$ be given such that $h_x^{(0)}$ and $h_t^{(0)} := T/2$ satisfy (3.5).
2: Let a bound $n_{\max}$ for the number of elements in a leaf box and $c_{\mathrm{st}} > 0$ for (3.5) be given.
3: Construct an empty tree $\mathcal{T}_\Sigma$ and add $Z^{(0)}$ as its root.
4: Call RefineCluster($Z^{(0)}$, $\mathcal{T}_\Sigma$)

5: **function** RefineCluster(Z, $\mathcal{T}_\Sigma$)
6: **if** $\#\{\sigma \subset \Sigma_h : \operatorname{center}(\sigma) \in Z\} \geq n_{\max}$
7: Let $\ell = \operatorname{level}(Z)$, $h_t^{(\ell+1)} = 2^{-\ell-1}h_t^{(0)}$ and $\tilde{h}_x$ be the spatial half-size of Z.
8: **if** $h_t^{(\ell+1)}$ and $\tilde{h}_x$ satisfy (3.5)
9: Subdivide Z into $n_{\mathrm{C}} = 2$ children $\{Z_j\}_{j=1}^{n_{\mathrm{C}}}$ by a temporal refinement.
10: **else**
11: Subdivide Z into $n_{\mathrm{C}} = 16$ children $\{Z_j\}_{j=1}^{n_{\mathrm{C}}}$ by a space-time refinement.
12: **for** $k = 1, \ldots, 16$
13: **if** $\#\{\sigma \subset \Sigma_h : \operatorname{center}(\sigma) \in Z_k\} \neq 0$
14: Add Z_k to $\mathcal{T}_\Sigma$ as child of T and call RefineCluster(Z_k, $\mathcal{T}_\Sigma$).

The refinement of a box $Z = (\boldsymbol{a}, \boldsymbol{b}] \times (c, d]$ in lines 9 and 11 of Algorithm 1 is done as follows. For the temporal center $(c + d)/2$ of Z there exists a unique interval (t_{k-1}, t_k) in the partition of $(0, T)$ such that $(c + d)/2 \in (t_{k-1}, t_k]$. We choose the temporal splitting point $\widetilde{c} = t_k$ if

$(t_{k-1} + t_k)/2 \leq (c+d)/2$ and $\widetilde{c} = t_{k-1}$ otherwise. In case of a purely temporal refinement, we split Z into the boxes $Z_1 = (\boldsymbol{a}, \boldsymbol{b}] \times (c, \widetilde{c}]$ and $Z_2 = (\boldsymbol{a}, \boldsymbol{b}] \times (\widetilde{c}, d]$. In case of a space-time refinement we additionally split the spatial part $(\boldsymbol{a}, \boldsymbol{b}]$ of Z uniformly into 8 boxes $(\boldsymbol{a}, \widetilde{\boldsymbol{a}}]$, ..., $(\widetilde{\boldsymbol{a}}, \boldsymbol{b}]$ where $\widetilde{\boldsymbol{a}} = 1/2\,(\boldsymbol{a} + \boldsymbol{b})$, and get 16 space-time boxes $\{Z_j\}_{j=1}^{16}$ as combinations of these refined spatial boxes and the temporal intervals $(c, \widetilde{c}]$ and $(\widetilde{c}, d]$. Note that due to (3.5) we alternate between purely temporal and space-time refinements in the construction of $\mathcal{T}_\Sigma$, at least after some initial temporal refinements.

With the described splitting in time it is guaranteed that the temporal part of a space-time element σ is always fully contained in a box Z if its center is in Z. To ensure that the spatial part of σ is fully contained in the respective box as well, we pad the boxes in $\mathcal{T}_\Sigma$ appropriately in a post-processing step, i.e., we extend the spatial size of a box Z if necessary such that for all σ with $\operatorname{center}(\sigma) \in Z$ there holds $\sigma \subset Z$. To retain the uniformity of the spatial parts of the boxes at a given level of $\mathcal{T}_\Sigma$, we pad all of them by the same amount in all directions. In addition, we pad boxes at level ℓ of $\mathcal{T}$ at least by the same amount as boxes at level $\ell + 1$. This ensures that the children of a box Z in $\mathcal{T}_\Sigma$ are fully contained in Z, which we also need later on.

A few aspects of Algorithm 1 require additional attention. The estimate (3.5) does not have to be satisfied in general for given $h_x^{(0)}$, $h_t^{(0)}$ and c_{st} as required in line 1. However, it can be established by additional refinements of the initial spatial box. Due to the non-uniform refinement in time and the padding in space, (3.5) might also be violated for other boxes in $\mathcal{T}_\Sigma$, but for suitably regular meshes Σ_h it will still hold for a slightly larger constant c_{st}. Finally, we want to point out that the refinement process should be stopped earlier for a box Z if it contains a space-time element $\sigma = \gamma \times (t_{j-1}, t_j)$ whose temporal or spatial size is considerably larger than the temporal or spatial half-size of Z. This is in particular the case if all of the elements in Z share the same temporal component.

We define the set of boxes/clusters at level ℓ of $\mathcal{T}_\Sigma$ by $\mathcal{T}_\Sigma^{(\ell)}$, its leaves by $\mathcal{L}_\Sigma$ and its depth, which is the largest level attained by any of its clusters, by $p(\mathcal{T}_\Sigma)$. For a cluster $Z \in \mathcal{T}_\Sigma$ we denote the set of all its children by $\operatorname{child}(Z)$ and its parent by $\operatorname{par}(Z)$. By $\hat{Z}$ we denote the set of all indices $(k_t, k_{\boldsymbol{x}})$ such that $\sigma_{k_t,k_{\boldsymbol{x}}} = \gamma_{k_{\boldsymbol{x}}} \times (t_{k_t-1}, t_{k_t}) \in Z$.

3.3. Nearfield and interaction lists of boxes in the space-time cluster tree

In Section 3.1 we have approximated the heat kernel $G_\alpha(\boldsymbol{x} - \boldsymbol{y}, t - \tau)$ for $(\boldsymbol{x}, t)$ in a target box Z_1 and $(\boldsymbol{y}, \tau)$ in a source box Z_2. We recall that for this approximation Z_1 and Z_2 have to be separated in time but not in space, and that the values of G_α are negligibly small if the spatial distance of Z_1 and Z_2 is large enough. Based on these observations we define the nearfield and interaction lists of target boxes in $\mathcal{T}_\Sigma$ which will determine the operations in the FMM algorithm.

We start by considering the temporal components of boxes in $\mathcal{T}_\Sigma$. By construction of $\mathcal{T}_\Sigma$ there exist at most 2^ℓ distinct time intervals that are components of boxes in $\mathcal{T}_\Sigma^{(\ell)}$. These intervals can be organized in a binary tree $\mathcal{T}_I$. In general, $\mathcal{T}_I$ is a full binary tree with depth $p(\mathcal{T}_I) = p(\mathcal{T}_\Sigma)$, but does not have to be a perfect binary tree. The intervals in $\mathcal{T}_I^{(\ell)}$, i.e., at level ℓ of $\mathcal{T}_I$, are numbered in ascending order from 0 to $2^\ell - 1$ skipping the numbers of potentially missing intervals. The leaves of $\mathcal{T}_I$ are denoted by $\mathcal{L}_I$.

For a target interval $I_k^{(\ell)}$ in $\mathcal{T}_I^{(\ell)}$ we define the nearfield $\mathcal{N}(I_k^{(\ell)})$ and interaction list $\mathcal{I}(I_k^{(\ell)})$ by

$$\mathcal{N}(I_k^{(\ell)}) := \begin{cases} \{I_k^{(\ell)}\}, & \text{if } k = 0, \\ \left(\{I_{k-1}^{(\ell)}, I_k^{(\ell)}\} \cap \mathcal{T}_I^{(\ell)}\right) \cup \left(\mathcal{N}(\mathrm{par}(I_k^{(\ell)})) \cap \mathcal{L}_I\right), & \text{otherwise,} \end{cases}$$

$$\mathcal{I}(I_k^{(\ell)}) := \begin{cases} \emptyset, & \text{if } k \in \{0,1\}, \\ \{I_{k-2}^{(\ell)}\} \cap \mathcal{T}_I^{(\ell)}, & \text{if } k \geq 2 \text{ and } k \text{ is even}, \\ \{I_{k-3}^{(\ell)}, I_{k-2}^{(\ell)}\} \cap \mathcal{T}_I^{(\ell)}, & \text{if } k \geq 2 \text{ and } k \text{ is odd}. \end{cases} \tag{3.6}$$

Due to the causality of the heat kernel, both sets include only intervals with indices $j \leq k$. The intervals in the interaction list of $I_k^{(\ell)}$ are well-separated from it and therefore suitable for the kernel approximation in Section 3.1. Earlier time intervals are not contained in $\mathcal{I}(I_k^{(\ell)})$ because they can be handled on a coarser level in the tree in the later FMM algorithm, see Section 3.4

The spatial component $X^{(\ell)}$ of a box $Z^{(\ell)} \in \mathcal{T}_\Sigma^{(\ell)}$ is contained in a regular grid $\mathcal{G}^\ell$ consisting of 8^{ℓ_x} possibly overlapping boxes, where ℓ_x is the number of spatial refinements of boxes in $\mathcal{T}_\Sigma^{(\ell)}$ and depends on ℓ. Boxes in the grid $\mathcal{G}^\ell$ can be labeled by using multi-indices in $\{0, ..., 2^{\ell_x} - 1\}^3$. We say that two boxes X and Y in $\mathcal{G}^\ell$ have *grid distance* n if the related multi-indices $\boldsymbol{\xi}$ and $\boldsymbol{\zeta}$ satisfy $n = \max_j\{|\xi_j - \zeta_j|\}$. For a fixed parameter n_{tr} we define the interaction area $\mathcal{I}_\mathcal{A}(X^{(\ell)})$ of a box $X^{(\ell)}$ in $\mathcal{G}^{(\ell)}$ by

$$\mathcal{I}_\mathcal{A}(X^{(\ell)}) := \{Y^{(\ell)} \in \mathcal{G}^{(\ell)} : \text{ the grid distance of } X^{(\ell)} \text{ and } Y^{(\ell)} \text{ is at most } n_{\mathrm{tr}}\}. \tag{3.7}$$

Finally, we define the nearfield and interaction list of a box $Z^{(\ell)} = X \times I \in \mathcal{T}_\Sigma^{(\ell)}$ by

$$\mathcal{N}(Z^{(\ell)}) := \left\{Z_{\mathrm{src}}^{(\ell)} = Y \times J \in \mathcal{T}_\Sigma^{(\ell)} : J \in \mathcal{N}(I) \text{ and } Y \in \mathcal{I}_\mathcal{A}(X)\right\} \cup \left(\mathcal{N}(\mathrm{par}(Z^{(\ell)})) \cap \mathcal{L}_\Sigma\right), \tag{3.8}$$

$$\mathcal{I}(Z^{(\ell)}) := \left\{Z_{\mathrm{src}}^{(\ell)} = Y \times J \in \mathcal{T}_\Sigma^{(\ell)} : J \in \mathcal{I}(I) \text{ and } Y \in \mathcal{I}_\mathcal{A}(X)\right\}. \tag{3.9}$$

Both sets exclude boxes $Z_{\mathrm{src}}^{(\ell)}$ whose spatial components are not in $\mathcal{I}_\mathcal{A}(X)$ which is motivated by the observations in the last paragraph of Section 3.1. Note that the same cutting parameter n_{tr} can be chosen for all boxes in $\mathcal{T}_\Sigma$, cf. [10, p.210].

3.4. The main space-time FMM algorithm

With the box cluster tree $\mathcal{T}_\Sigma$ and the interaction and nearfield lists of its clusters we construct a partition of the matrix V_h into blocks. By $\mathsf{V}_h|_{\hat{Z}_{\mathrm{tar}} \times \hat{Z}_{\mathrm{src}}}$ we denote the block of V_h whose rows correspond to indices $(j_t, j_{\boldsymbol{x}}) \in \hat{Z}_{\mathrm{tar}}$ and columns to indices $(k_t, k_{\boldsymbol{x}}) \in \hat{Z}_{\mathrm{src}}$. We decompose V_h into *admissible blocks* corresponding to indices $\hat{Z}_{\mathrm{tar}} \times \hat{Z}_{\mathrm{src}}$ with $Z_{\mathrm{src}} \in \mathcal{I}(Z_{\mathrm{tar}})$, *inadmissible blocks* $\hat{Z}_{\mathrm{tar}} \times \hat{Z}_{\mathrm{src}}$ where $Z_{\mathrm{tar}} \in \mathcal{L}_\Sigma$ and $Z_{\mathrm{src}} \in \mathcal{N}(Z_{\mathrm{tar}})$, and remaining blocks, whose entries are zero due to the causality of V_h or negligibly small due to the exponential decay in space of the kernel G_α. The FMM algorithm is used to compute an efficient approximation of the product $\mathsf{V}_h \boldsymbol{w}$ using this partition.

Inadmissible blocks of V_h are small by construction, since a leaf box $Z_{\mathrm{tar}} \in \mathcal{L}_\Sigma$ contains only few space-time elements. Hence, we can afford to store and apply them directly, i.e., we compute the product $\widetilde{\boldsymbol{f}}|_{\hat{Z}_{\mathrm{tar}}} = (\mathsf{V}_h|_{\hat{Z}_{\mathrm{tar}} \times \hat{Z}_{\mathrm{src}}} \boldsymbol{w}|_{\hat{Z}_{\mathrm{src}}})$ as part of $\boldsymbol{f} = \mathsf{V}_h \boldsymbol{w}$ by

$$\widetilde{f}_{k_t, k_{\boldsymbol{x}}} = \sum_{(j_t, j_{\boldsymbol{x}}) \in \hat{Z}_{\mathrm{src}}} w_{j_t, j_{\boldsymbol{x}}} \int_{t_{k_t - 1}}^{t_{k_t}} \int_{\gamma_{k_{\boldsymbol{x}}}} \int_{t_{j_t - 1}}^{t_{j_t}} \int_{\gamma_{j_{\boldsymbol{x}}}} G_\alpha(\boldsymbol{x} - \boldsymbol{y}, t - \tau) \, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}} \, \mathrm{d}\tau \, \mathrm{d}\boldsymbol{s}_{\boldsymbol{x}} \, \mathrm{d}t \tag{3.10}$$

for all $(k_t, k_{\boldsymbol{x}}) \in \hat{Z}_{\mathrm{tar}}$, where $w_{j_t,j_{\boldsymbol{x}}}$ denotes a coefficient of the vector $\boldsymbol{w}$. The computation of the corresponding integrals is discussed in [25].

For an admissible block $\mathsf{V}_h|_{\hat{Z}_{\mathrm{tar}} \times \hat{Z}_{\mathrm{src}}}$ with boxes $Z_{\mathrm{tar}} = X \times I$ and $Z_{\mathrm{src}} = Y \times J$ we can replace the kernel G_α in (3.10) by its approximation in (3.1). The result can be computed in 3 steps:

S2M: For $a \in \{0, ..., m_t\}$ and $\boldsymbol{\kappa} \in \mathbb{N}_0^3$ with $|\boldsymbol{\kappa}| \leq m_{\boldsymbol{x}}$ compute the *moments* $\boldsymbol{\mu}(Z_{\mathrm{src}})$ by

$$\mu_{a,\boldsymbol{\kappa}}(Z_{\mathrm{src}}) := \sum_{(j_t,j_{\boldsymbol{x}}) \in \hat{Z}_{\mathrm{src}}} w_{j_t,j_{\boldsymbol{x}}} \int_{t_{j_t-1}}^{t_{j_t}} \int_{\gamma_{j_{\boldsymbol{x}}}} T_{Y,\boldsymbol{\kappa}}(\boldsymbol{y}) L_{J,a}(\tau) \, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}} \, \mathrm{d}\tau. \tag{3.11}$$

M2L: For $b \in \{0, ..., m_t\}$ and $\boldsymbol{\nu} \in \mathbb{N}_0^3$ with $|\boldsymbol{\nu}| \leq m_{\boldsymbol{x}}$ compute the *local contributions* $\boldsymbol{\lambda}(Z_{\mathrm{tar}})$ by

$$\lambda_{b,\boldsymbol{\nu}}(Z_{\mathrm{tar}}) := \sum_{a=0}^{m_t} \sum_{|\boldsymbol{\kappa}+\boldsymbol{\nu}| \leq m_{\boldsymbol{x}}} E_{a,\boldsymbol{\kappa},b,\boldsymbol{\nu}} \, \mu_{a,\boldsymbol{\kappa}}(Z_{\mathrm{src}}). \tag{3.12}$$

L2T: For all $(k_t, k_{\boldsymbol{x}})$ in $\hat{Z}_{\mathrm{tar}}$ evaluate

$$\widetilde{f}_{k_t,k_{\boldsymbol{x}}} = \sum_{b=0}^{m_t} \sum_{|\boldsymbol{\nu}| \leq m_{\boldsymbol{x}}} \lambda_{b,\boldsymbol{\nu}}(Z_{\mathrm{tar}}) \int_{t_{k_t-1}}^{t_{k_t}} \int_{\gamma_{k_{\boldsymbol{x}}}} T_{X,\boldsymbol{\nu}}(\boldsymbol{x}) L_{I,b}(t) \, \mathrm{d}\boldsymbol{s}_{\boldsymbol{x}} \, \mathrm{d}t. \tag{3.13}$$

To enhance the performance we additionally use a nested computation of the moments and local contributions, see e.g., [9, Sections 4.2 and 4.3]. Moments of a non-leaf cluster are computed from the moments of its children via M2M operations. Local contributions of a non-leaf cluster are passed down to its children with an L2L operation, and evaluated together with the children's local contributions. We distinguish temporal and space-time M2M and L2L operations. Note that the temporal L2L and space-time L2L operations are just the transposed operations of the corresponding M2M operations, which is why we discuss only the latter.

Temporal M2M: For a box $Z_{\mathrm{p}} = X \times I_{\mathrm{p}}$ whose children are refined only in time there holds

$$\mu_{a_{\mathrm{p}},\boldsymbol{\kappa}}(Z_{\mathrm{p}}) = \sum_{\substack{Z_{\mathrm{c}} \in \mathrm{child}(Z_{\mathrm{p}}) \\ Z_{\mathrm{c}} = X \times I_{\mathrm{c}}}} \sum_{a_{\mathrm{c}}=0}^{m_t} q^{(t)}_{a_{\mathrm{c}},a_{\mathrm{p}}}(I_{\mathrm{c}}, I_{\mathrm{p}}) \, \mu_{a_{\mathrm{c}},\boldsymbol{\kappa}}(Z_{\mathrm{c}}), \quad q^{(t)}_{a_{\mathrm{c}},a_{\mathrm{p}}}(I_{\mathrm{c}}, I_{\mathrm{p}}) := L_{I_{\mathrm{p}},a_{\mathrm{p}}}(\xi^{(m_t)}_{I_{\mathrm{c}},a_{\mathrm{c}}}). \tag{3.14}$$

Space-time M2M: If the children of a box $Z_{\mathrm{p}} = X_{\mathrm{p}} \times I_{\mathrm{p}}$ with $X_{\mathrm{p}} = X_{\mathrm{p},1} \times X_{\mathrm{p},2} \times X_{\mathrm{p},3}$ are refined in space and time, there holds

$$\mu_{a_{\mathrm{p}},\boldsymbol{\kappa}_{\mathrm{p}}}(Z_{\mathrm{p}}) = \sum_{\substack{Z_{\mathrm{c}} \in \mathrm{child}(Z_{\mathrm{p}}) \\ Z_{\mathrm{c}} = X_{\mathrm{c}} \times I_{\mathrm{c}}}} \sum_{a_{\mathrm{c}}=0}^{m_t} \sum_{\boldsymbol{\kappa}_{\mathrm{c}} \leq \boldsymbol{\kappa}_{\mathrm{p}}} q^{(t)}_{a_{\mathrm{c}},a_{\mathrm{p}}}(I_{\mathrm{c}}, I_{\mathrm{p}}) \, q^{(\boldsymbol{x})}_{\boldsymbol{\kappa}_{\mathrm{c}},\boldsymbol{\kappa}_{\mathrm{p}}}(X_{\mathrm{c}}, X_{\mathrm{p}}) \, \mu_{a_{\mathrm{c}},\boldsymbol{\kappa}_{\mathrm{c}}}(Z_{\mathrm{c}}). \tag{3.15}$$

Here $\boldsymbol{\kappa}_{\mathrm{c}} \leq \boldsymbol{\kappa}_{\mathrm{p}}$ is understood componentwise, $q^{(t)}_{a_{\mathrm{c}},a_{\mathrm{p}}}(I_{\mathrm{c}}, I_{\mathrm{p}})$ are the coefficients in (3.14) and $q^{(\boldsymbol{x})}_{\boldsymbol{\kappa}_{\mathrm{c}},\boldsymbol{\kappa}_{\mathrm{p}}}(X_{\mathrm{c}}, X_{\mathrm{p}}) := \prod_j q^{(\boldsymbol{x})}_{k_{\mathrm{c}},k_{\mathrm{p}}}(X_{\mathrm{c},j}, X_{\mathrm{p},j})$ with

$$q^{(\boldsymbol{x})}_{k_{\mathrm{c}},k_{\mathrm{p}}}(X_{\mathrm{c},j}, X_{\mathrm{p},j}) := \frac{\lambda_{k_{\mathrm{c}}}}{m_{\boldsymbol{x}}+1} \sum_{n=0}^{m_{\boldsymbol{x}}} T_{X_{\mathrm{p},j},k_{\mathrm{p}}}(\xi^{(m_{\boldsymbol{x}})}_{X_{\mathrm{c},j},n}) T_{X_{\mathrm{c},j},k_{\mathrm{c}}}(\xi^{(m_{\boldsymbol{x}})}_{X_{\mathrm{c},j},n}),$$

where $\lambda_0 = 1$ and $\lambda_k = 2$ for all $k \geq 1$.

The resulting space-time FMM algorithm for the computation of the product $\mathsf{V}_h \boldsymbol{w}$ is presented in Algorithm 2. If the temporal and spatial mesh sizes h_t and h_x of Σ_h satisfy $h_t \sim h_x^2$, the algorithm reduces the runtime complexity of the matrix-vector multiplication $\mathsf{V}_h \boldsymbol{w}$ from $\mathcal{O}((E_t\, E_{\boldsymbol{x}})^2)$ to $\mathcal{O}(m_t^2 m_{\boldsymbol{x}}^4 E_t E_{\boldsymbol{x}})$, see e.g., [9, Section 5.4]. In [12] an additional nearfield compression is provided for meshes whose temporal mesh sizes are too large. Such a nearfield compression is not considered in this work.

Algorithm 2 Space-time FMM for the approximate evaluation of $\boldsymbol{f} = \mathsf{V}_h \boldsymbol{w}$

Choose the parameters $n_{\max}$, c_{st}, n_{tr} and the expansion degrees m_t and $m_{\boldsymbol{x}}$.
Construct the box cluster tree $\mathcal{T}_\Sigma$ and determine the sets $\mathcal{N}(Z)$ and $\mathcal{I}(Z)$ for all $Z \in \mathcal{T}_\Sigma$ according to (3.8) and (3.9).
Initialize $\boldsymbol{f} = \boldsymbol{0}$.
▷ *Forward transformation*
for all leaves $Z \in \mathcal{L}_\Sigma$
    S2M: Compute $\boldsymbol{\mu}(Z)$ by (3.11).
for all levels $\ell = p(\mathcal{T}_\Sigma) - 1, \ldots, 2$
    for all non-leaf boxes $Z_{\mathrm{p}} \in \mathcal{T}_\Sigma^{(\ell)}$
        if children of Z_{p} are refined only in time
            Temporal M2M: Compute $\boldsymbol{\mu}(Z_{\mathrm{p}})$ by (3.14).
        else
            Space-time M2M: Compute $\boldsymbol{\mu}(Z_{\mathrm{p}})$ by (3.15).
▷ *Multiplication phase*
for all boxes $Z_{\mathrm{tar}} \in \mathcal{T}_\Sigma$
    Initialize $\boldsymbol{\lambda}(Z_{\mathrm{tar}}) = \boldsymbol{0}$.
    for all boxes $Z_{\mathrm{src}} \in \mathcal{I}(Z_{\mathrm{tar}})$
        M2L: Update $\boldsymbol{\lambda}(Z_{\mathrm{tar}})$ by adding the result from (3.12).
▷ *Backward transformation*
for all levels $\ell = 3, \ldots, p(\mathcal{T}_\Sigma)$
    for all boxes $Z_{\mathrm{c}} \in \mathcal{T}_\Sigma^{(\ell)}$
        if Z_{c} results from its parent Z_{p} by a purely temporal refinement
            Temporal L2L: Update $\boldsymbol{\lambda}(Z_{\mathrm{c}})$ using $\boldsymbol{\lambda}(Z_{\mathrm{p}})$.
        else
            Space-time L2L: Update $\boldsymbol{\lambda}(Z_{\mathrm{c}})$ using $\boldsymbol{\lambda}(Z_{\mathrm{p}})$.
for all leaves $Z \in \mathcal{L}_\Sigma$
    for all k_t and $k_{\boldsymbol{x}}$ such that $\sigma_{k_t,k_{\boldsymbol{x}}} \in Z$
        L2T: Update $f_{k_t,k_{\boldsymbol{x}}}$ by adding the result from (3.13).
▷ *Nearfield evaluation*
for all leaves $Z_{\mathrm{tar}} \in \mathcal{L}_\Sigma$
    for all Z_{src} in the nearfield $\mathcal{N}(Z_{\mathrm{tar}})$
        Update $f|_{\hat{Z}_{\mathrm{tar}}} \mathrel{+}= \mathsf{V}_h|_{\hat{Z}_{\mathrm{tar}} \times \hat{Z}_{\mathrm{src}}} \boldsymbol{w}|_{\hat{Z}_{\mathrm{src}}}$.

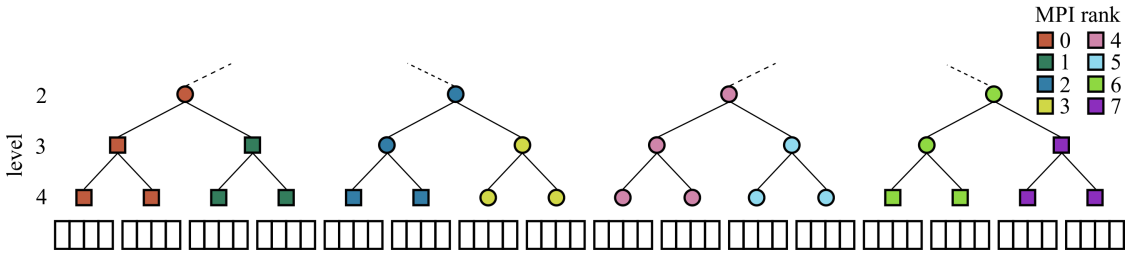

Figure 4.1: 16 initial time-slices each containing 4 time-steps. The distribution tree is built above these time-slices and split among available MPI processes. The clusters of the LET on process 4 are marked with circles.

4. Parallelization of the space-time FMM in shared and distributed memory

We present a novel parallel implementation of the space-time FMM presented in Section 3. Our distributed parallelization strategy relies on a decomposition of the one-dimensional temporal tree $\mathcal{T}_I$, see Section 3.3, into locally essential trees (LET) [28], which are distributed among the available MPI processes. A LET is a local part of a tree extended by fractions of the global tree which are required in the collaboration of the processes. The FMM operations are assigned to the processes according to these temporal subtrees. The necessary inter-process communication is also handled clusterwise in the temporal tree in a one-directional way mainly between successive processes due to the causality of the boundary integral operators. This clusterwise communication leads to a small number of communication events of reasonable size.

A distributed space-time cluster tree is assembled collaboratively based on the temporal tree. Certain parts of the FMM can be computed locally on the space-time subtrees, while other parts of the computations depend on the results from other subtrees, i.e., processes. We employ a small scheduling system to keep track of the dependencies. Relations between the temporal clusters within the global distributed tree define dependencies which are used to decompose the computation in Algorithm 2 into tasks which can be executed asynchronously. During the matrix vector multiplication a scheduler goes through a list of available tasks and executes those with fulfilled dependencies. This enables an asynchronous parallelization instead of bulk-synchronous approach which is often used to parallelize the FMM. Let us describe the individual phases of the computation in more details.

4.1. Preprocessing

The preprocessing phase consists of the assembly of the temporal cluster tree and its decomposition, the assembly of the distributed space-time mesh, and the creation of the distributed space-time cluster tree. We start by splitting the global time interval $(0, T)$ and the related time-steps into time-slices. Each of the slices contains multiple time-steps (see Figure 4.1). Next, a binary tree is built by a recursive bisection of the global interval and the time-slices are assigned to the nodes of the tree accordingly. We distribute the individual nodes of the temporal cluster tree among the available N_{proc} MPI processes. This temporal decomposition will drive the distribution of the space-time cluster tree later on and will determine the required communication between processes. Multiple distribution approaches can be employed; in our experiments we try to reduce inter-process communication by determining the distribution starting from the finest tree level and taking the following steps:

- On level $\lceil \log_2 N_{\text{proc}} \rceil$ and finer levels of the temporal cluster tree, distribute the clusters and the related time-slices among the N_{proc} processes in ascending order and as uniformly as possible.

- On level $\lceil \log_2 N_{\text{proc}} \rceil - 1$, a cluster is assigned to the process that handles its left child. In this way we improve the load balancing because the temporal interaction list of the left child is smaller than the one of the right child, see (3.6).

- On each level ℓ with $\ell < \lceil \log_2 N_{\text{proc}} \rceil - 1$, we split the processes into 2^ℓ groups of ascending order to assign the 2^ℓ cluster of the level. From each group we pick a process, which has been responsible for the smallest number of clusters so far, and assign it to the related cluster. Again we aim at improving the load balancing.

Figure 4.1 gives an example of a distribution based on this strategy. Please check the coloring of the nodes for the assignment of processes.

The computation of the matrix-vector product on a process requires data from remote processes. On each process the *locally essential tree* is constructed. The LET on process p contains the assigned clusters as well as clusters that are needed for nearfield computations and clusters which are related to FMM operations requiring communication from or to process p (see MPI rank 4 in Figure 4.1). The clusters in the LET for which the process is not directly responsible contain the ranks of the responsible remote processes. Since the temporal tree defines the distribution of work among processes we will call it the *scheduling tree*.

Next, we combine the time-steps in the individual time-slices with the spatial mesh to create a distributed space-time tensor product mesh. In addition to time-steps for which a process is directly responsible we also include time-steps in the temporal nearfield of its clusters to reduce communication during the assembly of the nearfield blocks of the matrix.

A distributed 4D space-time box cluster tree is created level-wise top down as described in Section 3.2 and in Algorithm 1. The nodes of the tree are assigned to the processes based on the assignment of their temporal components defined in the scheduling tree. Each node of the scheduling tree stores the information about the corresponding space-time clusters. The depth of the space-time tree may be larger than the depth of the temporal tree. In such a case the temporal tree is locally extended accordingly. When building the upper part of the space-time cluster tree, where clusters contain elements from local meshes assigned to multiple processes, communication is required but can be limited to a reasonable amount. However, the construction of the lower parts of the tree is done independently on each process and just some synchronization with direct neighbors is carried out to set up the communication.

4.2. Matrix-vector multiplication

Our parallelization approach in shared and distributed memory is based on a data driven model, instead of a bulk-synchronous parallelization often used in scientific codes. In [14] such an approach is used for the parallelization of an FMM for particle simulations using `Charm++`. We use a different strategy for work distribution and execution and a custom scheduler in combination with OpenMP `tasks` in order to avoid dependencies on external software. To distinguish our own tasks from OpenMP `tasks`, the latter ones will be denoted in monospaced font from now on.

Our top-level scheduler is based on the FMM operations and dependencies with respect to the temporal scheduling tree. At this stage just consider Algorithm 2 reduced to the scheduling tree $\mathcal{T}_I$ and the related temporal operations. If such a temporal FMM operation is called, OpenMP

`tasks` of all related space-time operations of the attached space-time clusters are generated to implement shared memory parallelization and to control the granularity of the `tasks`.

The strategy of the temporal scheduler is as follows: Most tasks are related to pairs of clusters. Certain temporal tasks do not have dependencies and can be executed at any time, while some temporal tasks depend on the results of other temporal tasks. We distinguish local and remote dependencies. If a computation is done on the same process, the dependency is resolved locally. If a tasks requires data from a computation of another process, the calculation can start at earliest after the communication has taken place. The temporal scheduler prioritizes those tasks which other tasks depend on. In case none of these tasks can be executed at some time, we avoid idle times by scheduling independent tasks.

In more details, we start by decomposing the temporal variant of Algorithm 2 into tasks and defining their mutual dependencies with respect to the temporal scheduling tree. We distinguish the following lists of tasks:

- *M-list* – S2M and M2M operations including send operations of the computed moments to the parent and the clusters in the interaction list,
- *M2L-list* – transformations of moments into local contributions (M2L) including possible downward send operations or evaluations of the local contributions in case of a leaf (L2T),
- *L-list* – translations of local contributions from parent (L2L) including possible downward send operations or evaluations of the local contributions in case of a leaf (L2T),
- *N-list* – execution of the nearfield operations.

The tasks in a list are ordered in a way which is advantageous for our parallel execution. Within each list a task can be identified from the related cluster. Thus we just store temporal clusters in these lists. Each cluster is assigned to one or more of the lists depending on its position in the tree. This enables us to specify dependencies for individual clusters (tasks):

- Tasks of non-leaf clusters in the M-list depend on the completion of the M-list operations of their children.
- Tasks of clusters in the M2L-list depend on the completion of M-list operations of clusters in their interaction lists (see (3.6)).
- Tasks of clusters in the L-list depend on the completion of their parents' M2L- and L-list operations.
- Nearfield tasks of clusters in the N-list have no dependencies.

Note that each process creates only the parts of these lists which are relevant to its local part of the scheduling tree and to its LET, respectively. We do not describe this restriction explicitly but it results from the local scheduling tree naturally.

A simplified parallel matrix-vector multiplication algorithm is described in Algorithm 3. We start by filling the above mentioned lists by clusters of the scheduling tree according to the presented rules. In addition to the distributed parallelization with MPI, we make use of OpenMP thread parallelization within each process. First, we enter the OpenMP `parallel` region and create a `single` section to ensure that only one thread will execute the main scheduling loop. The routine StartMPIReceiveOperations() creates a non-blocking receive operation using

Algorithm 3 Parallel space-time FMM for the approximate evaluation of $\boldsymbol{f} = \mathsf{V}_h \boldsymbol{w}$

```
1: Fill the M_list, M2L_list, L_list, and N_list; Initialize f = 0.
2: Start OpenMP parallel region
3:     Start OpenMP single section
4:         StartMPIReceiveOperations( )
5:         while the lists are not empty
6:             CheckMPIForReceivedData( )
7:             [cluster, list] = FindNextCluster(M_list, L_list, M2L_list, N_list)
8:             if n_generated_tasks > treshold
9:                 Suspend the execution of scheduling task using the taskyield construct
10:            if list == 0
11:                CreateOpenMPMListTask(cluster)
12:                RemoveClusterFromList(cluster, M_list)
13:            else if list == 1
14:                CreateOpenMPLListTask(cluster)
15:                RemoveClusterFromList(cluster, L_list)
16:            else if list == 2
17:                CreateOpenMPM2LListTask(cluster)
18:                RemoveClusterFromList(cluster, M2L_list)
19:            else if list == 3
20:                CreateOpenMPNListTask(cluster)
21:                RemoveClusterFromList(cluster, N_list)
```

the `MPI_Irecv()` function for every temporal cluster in the lists requiring data from remote processes. Individual clusters and operations are distinguished using the `tag` argument of the MPI function.

At the beginning of each iteration in the while loop of Algorithm 3 the scheduling thread calls CheckMPIForReceivedData() which uses `MPI_Testsome()` to check for new data received from remote processes and updates the dependencies of the respective tasks and temporal clusters. Then the scheduling thread iterates through the lists calling FindNextCluster to find the next cluster or rather task ready to be executed, i.e., one with all dependencies fulfilled. Due to the succession of dependencies, first the M-list tasks are checked, then the L-list tasks, followed by the M2L-list tasks. Since the nearfield tasks are independent of all other tasks, the N-list is traversed last. If the scheduling thread finds a task, it creates an OpenMP `task` for executing the corresponding operations on the cluster and removes the cluster from the list using RemoveClusterFromList. Here the OpenMP task scheduler serves as a buffer for the initialized `tasks` and allows for a finer granularity of the tasks and a more efficient parallelization.

In order to avoid collisions of the OpenMP `tasks` during the memory access, the generated OpenMP `task` is subject to additional dependencies specified using the OpenMP `depend` clause. E.g., a task generated from the M-list depends on all previously generated `tasks` where the cluster has the same parent as the current cluster (to avoid collisions during the M2M operations) and the `tasks` where the cluster is in the interaction lists of the same cluster as the current cluster (to avoid collisions during the M2L operations).

When executing a task from one of the four lists, the related operations in the space-time cluster tree are processed. To improve the granularity of the shared memory parallelization,

additional finer level OpenMP `tasks` are created by iterating through the associated space-time clusters using the `taskloop` construct. After these associated finer level OpenMP `tasks` are completed, the dependencies in the temporal scheduler are updated. If the depending cluster is owned by a remote process, the required data are sent using the non-blocking `MPI_Isend()` operation.

After all lists are empty and all OpenMP `tasks` are completed, every process has computed its local part of the matrix-vector product $\boldsymbol{f} = \mathsf{V}_h \boldsymbol{w}$.

Notice, that the scheduling thread may suspend the execution of the `while` loop if the number of already generated tasks is greater than a certain threshold and join other threads in executing the generated `tasks` (see line 9 in Algorithm 3). However, the `taskyield` construct is a non-binding request and may result in a `no operation`. In this context, a suspension of the scheduling task is not observed, e.g., in the GCC compiler (v9.3). Therefore, we mainly focus on the Intel compiler in Section 5 on our numerical experiments.

Note. One could choose another task-based approach to hybrid OpenMP-MPI parallelization. Instead of providing a custom scheduler run by a dedicated thread, all tasks could be created at once including tasks responsible for data receiving and sending. The correct execution order would be ensured using the OpenMP `depend` clause. Unfortunately, due to the above-described characteristics of the `taskyield` construct and a limited number of OpenMP tasks we cannot prevent deadlocks when only receive or send operations are posted on the processes. Alternatively, one could rely on special compilers supporting task suspending such as OmpSS [29]. However, we decided to implement our own simple scheduler to reduce the number of external dependencies. This also enables us to better control the granularity of computation.

Note. Since we based our MPI parallelization on a decomposition of the temporal scheduling tree, the distributed memory parallelization is limited by the total number of time-steps. Extending the distributed parallelization into a spatial dimension would thus improve the scalability on large machines. The simplest approach that does not significantly disrupt the existing implementation would be to start from an existing decomposition and to replace each MPI process by a group of processes. Within this group, the processes would collaborate on the local space-time operations and a certain process from the group would be responsible for the communication with other groups using the same communication structure as the current algorithm. This hierarchical nature of the communication could exploit the topology of the computer cluster and thus further reduce the communication time.

5. Numerical experiments

The parallel space-time FMM algorithm described in Section 4 has been implemented in the publicly available C++ library BESTHEA [30]. To evaluate its performance we carried out numerical experiments using the Salomon and Barbora clusters at IT4Innovations National Supercomputing Center in Ostrava, Czech Republic. The Salomon cluster consists of 1009 compute nodes equipped with two 12-core Intel Xeon E5-2680v3 processors and 128 GB of RAM. The theoretical peak performance of the cluster is 2 PFLOP/s. The Barbora cluster consists of 201 computational nodes equipped with two 18-core Intel Cascade Lake 6240 CPUs and 192 GB of RAM. Nodes within both clusters are interconnected using the InfiniBand network. On Salomon we used the Intel Compiler v19.1.1 and the Intel Math Kernel Library (MKL) v2020.1 unless stated otherwise, while on Barbora we employed the Intel Compiler v19.1.3 and MKL v2020.4. The affinity

	# threads	1	2	4	8	18	36
V_h	time $[s]$	505.7	254.6	127.4	63.8	28.4	14.4
	efficiency [%]	100.0	99.3	99.3	99.2	98.8	97.6
K_h	time $[s]$	488.2	258.9	129.9	64.9	29.4	18.3
	efficiency [%]	100	94.3	94.0	94.0	92.3	74.1
iteration	time $[s]$	220.9	111.5	55.5	27.9	12.5	6.4
	efficiency [%]	100	99.1	99.5	98.8	98.3	95.8

Table 5.1: Assembly times and time per GMRES iteration for different numbers of OpenMP threads and a problem with 98 304 space-time surface elements (64 time-steps, 1 536 spatial elements) using a single node of the Barbora cluster.

of the threads to cores was set using the variable `KMP_AFFINITY=granularity=core,compact`, which guarantees that the threads will stay within a single socket of the two-socket system when possible. The details of how to reproduce the numerical results are provided in the text below and in the software repository [30].

5.1. Shared memory performance

The performance of our OpenMP parallelization of the space-time FMM was tested on up to 36 cores on a node of the Barbora cluster. The scalability was tested for the Dirichlet problem and the boundary element method described in Section 2. As the Dirichlet datum we use

$$u(\boldsymbol{x}, t) = G_\alpha(\boldsymbol{x} - \boldsymbol{y}^*, t) \quad \text{for } (\boldsymbol{x}, t) \in \Sigma, \tag{5.1}$$

where $\boldsymbol{y}* = (1.5, 1.5, 1.5)$ and $\alpha = 1$. The lateral boundary of the space-time domain $(-0.5, 0.5)^3 \times (0, 0.25)$ was discretized into 1 536 spatial and 64 temporal elements, resulting in 98 304 space-time boundary elements in total. The temporal elements were distributed among 16 time-slices. The space-time leaf cluster size was limited using the value $n_{\max} = 80$ (see Algorithm 1) and the orders of the Lagrange and Chebyshev polynomials were both set to $m_t = m_{\boldsymbol{x}} = 6$. Finally, we used $c_{\text{st}} = 0.9$ in the relation (3.5) controlling the spatial and temporal box sizes and the cutting parameter $n_{\text{tr}} = 5$ in (3.7). Note that here and in all other experiments we have chosen the parameters for the FMM such that they do not affect the approximation quality of the BEM (see, e.g., [5, 26] for results regarding the approximation quality).

The assembly times of the single and double layer matrices, respectively, and the time per GMRES (generalized minimal residual method) iteration are given in Table 5.1 with respect to the number of threads. Up to 18 threads on a single socket the efficiency of the matrix assembly is above 90%, while for 36 threads spanning over two sockets the efficiency drops mainly for the assembly of the double layer matrix due to the less efficient memory access. The iterative solution, which is mostly composed of the matrix-vector multiplication, scales almost optimally even for both sockets fully occupied.

On modern CPUs, the performance of a code highly depends on its ability to exploit SIMD vectorization. In our code we employ vectorization using OpenMP `simd` pragmas. This is demonstrated in Figure 5.1 where we present the scalability of the assembly of the system matrices (mainly involving the assembly of nearfield parts) and the iterative solution using GMRES with

respect to the SIMD registers vector width. We used the same settings as in the previous experiment. Compared to the matrix assembly, the GMRES vector scalability is limited due to the complexity of the space-time FMM code for the matrix vector multiplication. While further optimization would probably be possible, the current code is a trade-off between the performance and readability. Nevertheless, the achieved speedup still reduces the solution time significantly.

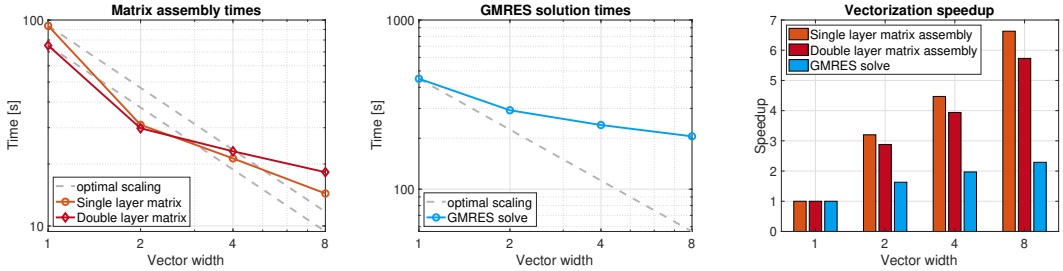

Figure 5.1: Efficiency of the vectorization with respect to the width of the vector registers. The non-vectorized version (vector width equals one) was compiled using the compiler flags `-no-vec -no-simd -qno-openmp-simd`. The remaining versions were compiled using the flags `-xcore-avx512 -qopt-zmm-usage=high` with the vector width set by the `simdlen` OpenMP clause.

Next, we solve a Dirichlet problem with 1.5 million space-time boundary elements using eight computational nodes of the Salomon cluster to demonstrate the performance of the task scheduling algorithm presented in Section 4. We use the same space-time domain as in the previous example discretized into 6144 spatial and 256 temporal elements (equally distributed among 16 time-slices) and the Dirichlet datum given by (5.1). We set $n_{\max} = 800, m_t = m_x = 6, c_{\text{st}} = 0.9$, and $n_{\text{st}} = 5$. In Figure 5.2 we visualize the execution of OpenMP `tasks` on 24 cores of the node with MPI rank 5. As expected, the computational time is dominated by the `tasks` dedicated to the M2L computation (denoted by the rectangles in the shades of red) and by the nearfield operations (blue rectangles). L2L and L2T operations are displayed in green and require a negligible amount of the computational time. The S2M and M2M operations at the beginning of computation are depicted in orange but are hardly visible in the graph, therefore we provide a zoom into the first 8000 μs of the computation in Figure 5.3. The moments when data are sent or received via MPI operations are marked with green and yellow triangles, respectively. Note, that the scheduling thread 0 does not only take care of data reception and creating `tasks` when dependencies are fulfilled, but also participates in the execution of `tasks`. We observe that the scheduling algorithm (in combination with the Intel OpenMP runtime scheduler) is able to efficiently utilize all available threads. Since the MPI communication is non-blocking, it is hidden by the computation and the scheduling thread can participate in computations and only check for received data whenever it is scheduling new tasks. A very small amount of idle time is still visible in Figure 5.2. We could probably overcome these idle times by using a more flexible task scheduler instead of the OpenMP tasks, but since the amount of idle time is negligibly small we kept the latter.

Compare the results in Figures 5.2 and 5.3 with those in Figure 5.4. Here we solved a smaller problem on a single node using GCC v9.3 as a compiler. Notice that the first thread does not participate in the `task` execution up until the very end of the computation since it is busy creating the `tasks` (and possibly receiving data from other nodes) and the `taskyield` construct has no effect on it. Of course, this imbalance leads to longer execution times.

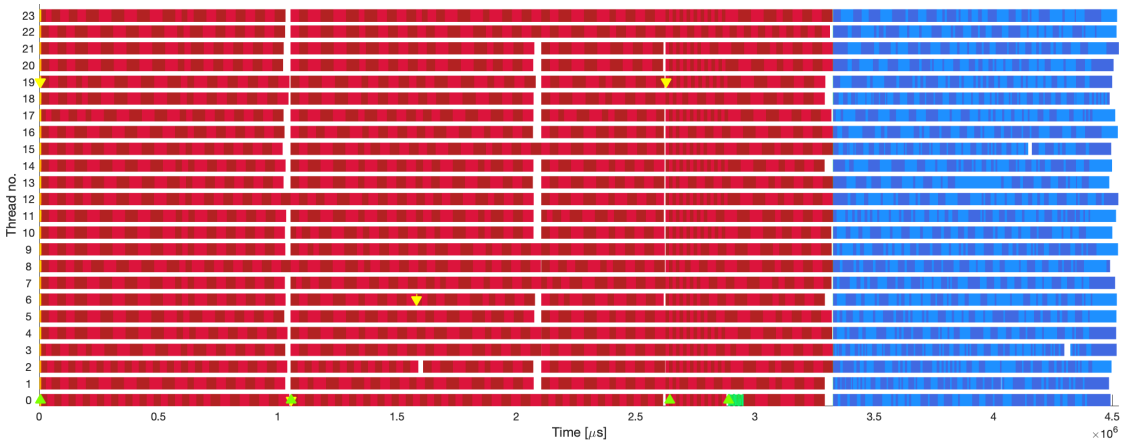

Figure 5.2: Execution of the OpenMP tasks on one of Salomon's nodes during a parallel matrix-vector multiplication using 8 nodes. The computation is dominated by the tasks dedicated to the M2L operations (light and dark red rectangles) and nearfield operations (light and dark blue rectangles). The S2M and M2M operations at the beginning of the computation are displayed in orange but are hardly visible (see Figure 5.3 for details) and L2L and L2T operations are depicted in green. Finally, moments of MPI communication are marked with yellow (send) and green (receive) triangles. The code was compiled using the Intel Compiler v19.1.1.

5.2. Distributed memory performance

Distributed memory scalability of the code was tested on up to 256 nodes (6144 cores) of the Salomon cluster. A hybrid MPI-OpenMP parallelization with one MPI process per node and 24 OpenMP threads per process was employed.

	# nodes	16	32	64	128	256
V_h	time $[s]$	769.8	385.5	194.4	97.1	50.0
	efficiency [%]	100.0	99.8	99.0	99.1	96.2
K_h	time $[s]$	502.4	252.5	128.6	63.0	31.9
	efficiency [%]	100.0	99.5	97.7	99.7	98.4
iteration	time $[s]$	14.7	7.3	3.7	2.1	1.5
	efficiency [%]	100.0	101.5	99.4	89.4	62.6

Table 5.2: Scalability of the code on up to 256 nodes of the Salomon cluster for a problem with 12 582 912 space-time surface elements (1024 time-steps, 12 288 spatial elements).

We again solve the model problem from Section 2 with the Dirichlet datum (5.1) and the lateral boundary of the computational domain $(-0.5, 0.5)^3 \times (0, 0.25)$ discretized into 12 288 spatial and 1024 temporal boundary elements (resulting in 12 582 912 space-time boundary elements in total). We equally distribute the temporal elements among 256 time-slices. The variable $n_{\max}$ is set to 800. The orders of the Chebyshev and Lagrange polynomials are set to 12 and 4, respectively, while the parameters $c_{\text{st}} = 4.1$ and $n_{\text{tr}} = 2$ are used.

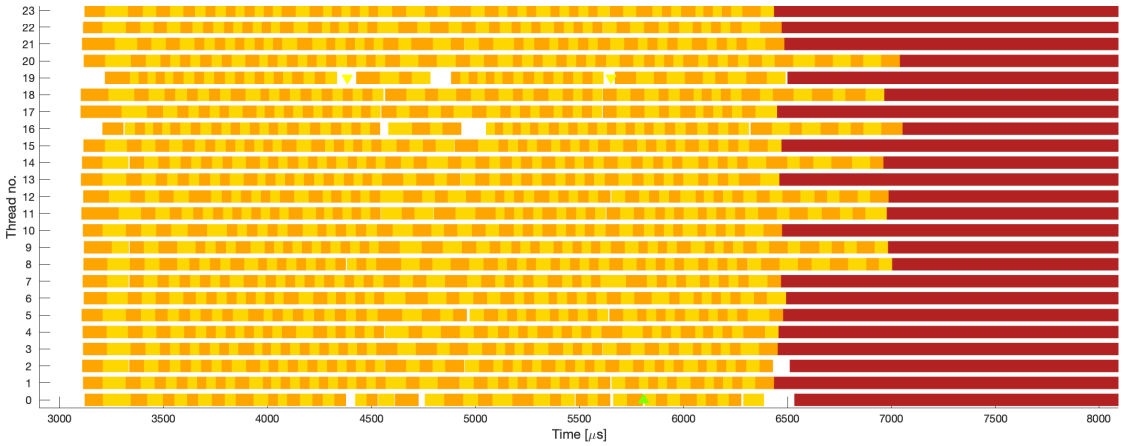

Figure 5.3: Details of the S2M and M2M tasks (in orange) from Figure 5.2. Moments of MPI communication are marked with yellow (send) and green (receive) triangles.

Results of the tests are presented in Table 5.2. The assembly of the system matrices V_h and K_h scales almost optimally up to 256 compute nodes. The same holds for the iteration times but there is a slight drop in efficiency for 256 nodes. This is probably due to the relatively small number of considered time-slices and time-steps. In fact, we decomposed the time interval into 256 time-slices, so when using 256 nodes we assign only one slice to each node and reach the limit of our parallelization scheme for this example. Nonetheless, we achieve a high efficiency for the distributed GMRES solver, reducing the time per iteration from 14.7 s on 16 nodes to 1.5 s on 256 nodes. The iterative solver requires 60 iterations to reach a relative accuracy of 10^{-8}, thus the total computation time is reduced from more than 2100 s on 16 nodes to approximately 200 s on 256 nodes.

Finally, to demonstrate the performance of our code on more realistic examples, we solve the Dirichlet problem for a crankshaft discretized by 42 888 spatial surface elements and the time interval $(0, 0.25)$ divided into 1024 time-steps (leading to a space-time surface mesh with approximately 44 million boundary elements). We use the same Dirichlet datum as in the previous examples, uniformly distribute the temporal elements among 256 time-slices, and set $n_{\max} = 800$,

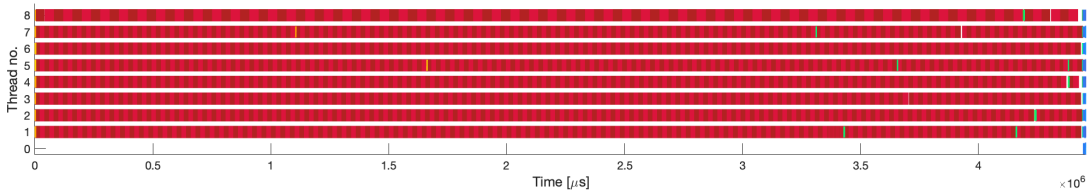

Figure 5.4: Exemplary visualization of the OpenMP task execution on one cluster node during a distributed matrix-vector multiplication using GCC v9.3. The `taskyield` construct shows no effect, and thus the first thread only schedules tasks and does not participate in their execution up until the very end.

$m_t = 3$, $m_x = 12$, $c_{\mathrm{st}} = 4.5$, and $n_{\mathrm{tr}} = 2$. Using 128 nodes of the Salomon cluster we are able to assemble the system matrices and solve the problem in less than two hours with a relative GMRES accuracy of 10^{-6}, see Table 5.3. The solution at the end of the time interval is depicted in Figure 5.5.

# compute nodes	V_h assembly	K_h assembly	solution	# iterations
128	1161.58 s	1223.97 s	2927.70 s	399

Table 5.3: Distributed solution of the "crankshaft problem" with 43 917 312 space-time surface elements (1024 time-steps, 42 888 spatial elements).

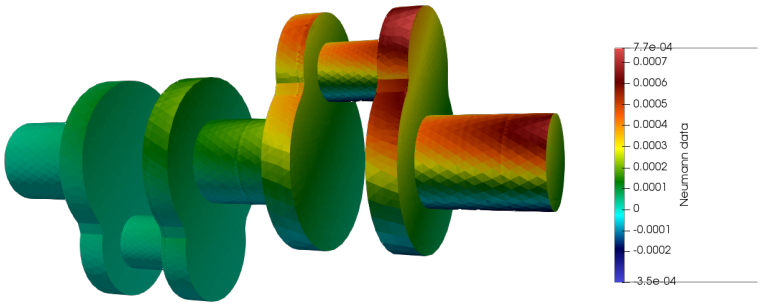

Figure 5.5: Computed Neumann datum for the crankshaft discretized by 43 917 312 space-time elements at time $t = 0.25$.

6. Conclusion and Outlook

In this paper we developed a parallel space-time FMM for the heat equation. We started from an existing space-time FMM and noticed that its temporal structure can be exploited for parallelization. In fact, the original space-time FMM can be associated with a 1D temporal tree that can be distributed among computing processes and allows to group the actual FMM operations in time. In our algorithm we used a simple task scheduler to execute these groups of operations in parallel based on individual dependencies and to realize inter-process communication in an asynchronous manner. This allowed us to overcome the strict distinction between FMM phases (forward transformation, multiplication phase, backward transformation) and global synchronization between processes. The resulting algorithm was implemented in the publicly available C++ library BESTHEA [30] as a hybrid MPI-OpenMP code. In several numerical experiments we investigated its efficiency. In particular, we showed close to optimal scalability on a large number of computing nodes.

The parallelism in time of the proposed method is a big advantage over other methods for the solution of boundary value problems of the heat equation like time-stepping schemes. As we have mentioned at the end of Section 4, one could refine the method by extending the distributed parallelization to an additional spatial dimension. We are optimistic that such an extension is compatible with the presented approach and would allow to increase its scalability even further.

Acknowledgements

The authors acknowledge the support provided by the Czech Science Foundation under the project 17-22615S, the Austrian Science Fund (FWF) under the project I 4033-N32, and by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90140).

References

[1] U. Langer, O. Steinbach (Eds.), Space-Time Methods. Applications to Partial Differential Equations, Vol. 25 of Radon Series on Computational and Applied Mathematics, De Gruyter, Berlin/Boston, 2019. doi:10.1515/9783110548488.

[2] J.-L. Lions, Y. Maday, G. Turinici, Résolution d'EDP par un schéma en temps «pararéel», C. R. Acad. Sci. - Series I - Mathematics 332 (7) (2001) 661–668. doi:10.1016/S0764-4442(00)01793-6.

[3] M. Gander, M. Neumüller, Analysis of a new space-time parallel multigrid algorithm for parabolic problems, SIAM J. Sci. Comput. 38 (4) (2016) A2173–A2208. doi:10.1137/15M1046605.

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

[5] M. Costabel, Boundary integral operators for the heat equation, Integral Equ. Oper. Theory 13 (1990) 498–552. doi:10.1007/BF01210400.

[6] 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.

[7] L. Greengard, J. Strain, A fast algorithm for the evaluation of heat potentials, Comm. Pure Appl. Math. 43 (8) (1990) 949–963. doi:10.1002/cpa.3160430802.

[8] L. Greengard, P. Lin, Spectral approximation of the free-space heat kernel, Appl. Comput. Harmon. Anal. 9 (1) (2000) 83–97. doi:10.1006/acha.2000.0310.

[9] J. Tausch, A fast method for solving the heat equation by layer potentials, J. Comput. Phys. 224 (2) (2007) 956–969. doi:10.1016/j.jcp.2006.11.001.

[10] J. Tausch, Fast Nyström methods for parabolic boundary integral equations, in: U. Langer, M. Schanz, O. Steinbach, W. Wendland (Eds.), Fast Boundary Element Methods in Engineering and Industrial Applications, Vol. 63 of Lecture Notes in Applied and Computational Mechanics, Springer-Verlag, Berlin Heidelberg, 2012, pp. 185–219. doi:10.1007/978-3-642-25670-7_6.

[11] H. Harbrecht, J. Tausch, A fast sparse grid based space-time boundary element method for the nonstationary heat equation, Numer. Math. 140 (1) (2018) 239–264. doi:10.1007/s00211-018-0963-5.

[12] 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.

[13] 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.

[14] M. AbdulJabbar, R. Yokota, D. Keyes, Asynchronous execution of the fast multipole method using Charm++ (2014). arXiv:1405.7487.

[15] M. AbdulJabbar, G. Markomanolis, H. Ibeid, R. Yokota, D. Keyes, Communication reducing algorithms for distributed hierarchical N-body problems with boundary distributions, Lecture Notes in Computer Science 10266 (2017) 79–96. doi:10.1007/978-3-319-58667-0_5.

[16] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, G. Biros, A massively parallel adaptive fast-multipole method on heterogeneous architectures, in: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09, Association for Computing Machinery, New York, 2009, pp. 1–12. doi:10.1145/1654059.1654118.

[17] F. Cruz, M. Knepley, L. Barba, PetFMM—A dynamically load-balancing parallel fast multipole library, Int. J. Numer. Methods. Eng. 85 (4) (2011) 403–428. doi:10.1002/nme.2972.

[18] R. Yokota, L. A. Barba, A tuned and scalable fast multipole method as a preeminent algorithm for exascale systems, J. High Perform. Comput. Appl. 26 (4) (2012) 337–346. doi:10.1177/1094342011429952.

[19] E. Agullo, B. Bramas, O. Coulaud, E. Darve, M. Messner, T. Takahashi, Task-based FMM for multicore architectures, SIAM J. Sci. Comput. 36 (1) (2014) C66–C93. doi:10.1137/130915662.

[20] M. Abduljabbar, M. Al Farhan, N. Al-Harthi, R. Chen, R. Yokota, H. Bagci, D. Keyes, Extreme scale FMM-accelerated boundary integral equation solver for wave scattering, SIAM J. Sci. Comput. 41 (3) (2019) C245–C268. doi:10.1137/18M1173599.

[21] T. Wang, C. D. Cooper, T. Betcke, L. A. Barba, High-productivity, high-performance workflow for virus-scale electrostatic simulations with Bempp-Exafmm (2021). arXiv:2103.01048.

[22] T. Takahashi, An interpolation-based fast-multipole accelerated boundary integral equation method for the three-dimensional wave equation, J. Comput. Phys. 258 (2014) 809–832. doi:10.1016/j.jcp.2013.11.008.

[23] S. Dohr, J. Zapletal, G. Of, M. Merta, M. Kravčenko, A parallel space-time boundary element method for the heat equation, Comput. Math. with Appl. 78 (9) (2019) 2852–2866. doi:10.1016/j.camwa.2018.12.031.

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

[25] J. Zapletal, R. Watschinger, G. Of, M. Merta, Semi-analytic integration for a parallel space-time boundary element method modeling the heat equation (2021). arXiv:2102.09811.

[26] S. Dohr, K. Niino, O. Steinbach, Space-time boundary element methods for the heat equation, in: Space-time methods – applications to partial differential equations, De Gruyter, 2019, pp. 1–60. doi:10.1515/9783110548488-001.

[27] M. Messner, A Fast Multipole Galerkin Boundary Element Method for the Transient Heat Equation, 1st Edition, Vol. 23 of Monographic Series TU Graz: Computation in Engineering and Science, Verlag der Technischen Universität Graz, 2014. doi:10.3217/978-3-85125-350-4.

[28] M. S. Warren, J. K. Salmon, A parallel hashed oct-tree N-body algorithm, in: Proceedings of the 1993 ACM/IEEE Conference on Supercomputing, Supercomputing '93, Association for Computing Machinery, New York, 1993, pp. 12–21. doi:10.1145/169627.169640.

[29] J. Schuchart, K. Tsugane, J. Gracia, M. Sato, The impact of taskyield on the design of tasks communicating through MPI, in: B. R. de Supinski, P. Valero-Lara, X. Martorell, S. Mateo Bellido, J. Labarta (Eds.), Evolving OpenMP for Evolving Architectures, Springer International Publishing, Cham, 2018, pp. 3–17. doi:10.1007/978-3-319-98521-3_1.

[30] M. Merta, G. Of, R. Watschinger, J. Zapletal, BESTHEA, https://github.com/zap150/besthea (2020).

Erschienene Preprints ab Nummer 2020/1

2020/1	D. Pacheco, T. Müller, O. Steinbach, G. Brenn: A mixed finite element formulation for generalised Newtonian fluid flows with appropriate natural outflow boundary conditions
2020/2	U. Langer, O. Steinbach, F. Tröltzsch, H. Yang: Unstructured space-time finite element methods for optimal sparse control of parabolic equations
2020/3	D.R.Q. Pacheco, R. Schussnig, O. Steinbach, T.-P. Fries: A fully consistent equal-order finite element method for incompressible flow problems
2020/4	U. Langer, O. Steinbach, F. Tröltzsch, H. Yang: Unstructured space-time finite element methods for optimal control of parabolic equations
2020/5	P. Gangl, K. Sturm, M. Neunteufel, J. Schöberl: Fully and Semi-Automated Shape Differentiation in NGSolve
2020/6	U. Langer, O. Steinbach, F. Tröltzsch, H. Yang: Space-time finite element discretization of parabolic optimal control problems with energy regularization
2020/7	G. Of, R. Watschinger: Complexity analysis of a fast directional matrix-vector multiplication
2020/8	P. Gangl, K. Sturm: Topological derivative for PDEs on surfaces
2020/9	D. R. Q. Pacheco, O. Steinbach: A continuous finite element framework for the pressure Poisson equation allowing non-Newtonian and compressible flow behaviour
2020/10	S. Kurz, S. Schöps, G. Unger, F. Wolf: Solving Maxwell's Eigenvalue Problem via Isogeometric Boundary Elements and a Contour Integral Method
2020/11	U. Langer, M. Schanz, O. Steinbach, W. L. Wendland (eds.): 18th Workshop on Fast Boundary Element Methods in Industrial Applications , Book of Abstracts
2020/12	P. Gangl, S. Köthe, C. Mellak, A. Cesarano, A. Mütze: Multi-objective free-form shape optimization of a synchronous reluctance machine
2021/1	O. Steinbach, M. Zank: A generalized inf-sup stable variational formulation for the wave equation
2021/2	U. Langer, O. Steinbach, H. Yang: Robust discretization and solvers for elliptic optimal control problems with energy regularization
2021/3	R. Löscher, O. Steinbach, M. Zank: Numerical results for an unconditionally stable space-time finite element method for the wave equation
2021/4	O. Steinbach, P. Gaulhofer: On space-time finite element domain decomposition methods for the heat equation
2021/5	D. R. Q. Pacheco, O. Steinbach: Space-time finite element tearing and interconnecting domain decomposition methods
2021/6	U. Langer, O. Steinbach, F. Tröltzsch, H. Yang: Space-time finite element methods for the initial temperature reconstruction
2021/7	J. Zapletal, R. Watschinger, G. Of, M. Merta: Semi-analytic integration for a parallel space-time boundary element method modeling the heat equation
2021/8	R. Watschinger, G. Of: An integration by parts formula for the bilinear form of the hypersingular boundary integral operator for the transient heat equation in three spatial dimensions
2021/9	O. Steinbach, C. Urzua-Torres: A new approach to space-time boundary integral equations for the wave equation
2021/10	O. Steinbach, C. Urzua-Torres, M. Zank: Towards coercive boundary element methods for the wave equation