Comput Visual Sci (2008) 11:237–249
DOI 10.1007/s00791-008-0095-z

REGULAR ARTICLE

Distributed $\mathcal{H}^2$-matrices for non-local operators

Steffen Börm · Joana Bendoraityte

Received: 10 October 2007 / Accepted: 12 January 2008 / Published online: 28 March 2008

Abstract $\mathcal{H}^2$-matrices can be used to approximate dense $n \times n$ matrices resulting from the discretization of certain non-local operators (e.g., Fredholm-type integral operators) in $\mathcal{O}(nk)$ units of storage, where k is a parameter controlling the accuracy of the approximation. Since typically $k \ll n$ holds, this representation is much more efficient than the conventional representation by a two-dimensional array. For very large problem dimensions, the amount of available storage becomes a limiting factor for practical algorithms. A popular way to provide sufficiently large amounts of storage at relatively low cost is to use a cluster of inexpensive computers that are connected by a network. This paper presents a method for managing an $\mathcal{H}^2$-matrix on a distributed-memory cluster that can be proven to be of almost optimal parallel efficiency.

1 Introduction

In order to explain the basic concepts of $\mathcal{H}^2$-matrix techniques we consider a simple model problem: let $\Gamma \subseteq \mathbb{R}^d$ be a subdomain or submanifold, and let $g : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a function. The Fredholm integral operator corresponding to g is given by

$$\mathcal{G}[u](x) := \int_\Gamma g(x, y)u(y)\,dy \quad \text{for all } x \in \Gamma.$$

In this context the function g is called the *integral kernel*. We discretize $\mathcal{G}$ by Galerkin's method using a finite element basis $(\varphi_i)_{i\in\mathcal{I}}$ and now have to handle the corresponding matrix $G \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ given by

$$G_{ij} = \int_\Gamma \varphi_i(x) \int_\Gamma g(x, y)\varphi_j(y)\,dy\,dx \quad \text{for all } i, j \in \mathcal{I}. \tag{1}$$

In typical applications the kernel g has global support, therefore the matrix G will be dense (unless special basis functions are used [1,6,13]). This means that the familiar sparse matrix techniques cannot be applied and we have to resort to *data-sparse* techniques, i.e., representations or approximations of G that require significantly less than the $\mathcal{O}(n^2)$ units of storage needed for the standard representation as a two-dimensional array.

A very successful approach are panel-clustering techniques [12,15] and the closely related multipole methods [10, 14]. These techniques pick suitable subdomains $\tau, \sigma \subseteq \Gamma$ and replace the kernel g by a degenerate expansion

$$g(x, y) \approx \tilde{g}_{\tau,\sigma}(x, y) = \sum_{\nu=1}^{k}\sum_{\mu=1}^{k} s_{\tau,\sigma,\nu,\mu}v_{\tau,\nu}(x)w_{\sigma,\mu}(y) \tag{2}$$

with coefficients $s_{\tau,\sigma,\nu,\mu}$ depending on τ and σ and expansion functions $v_{\tau,\nu}$ depending only on τ and $w_{\sigma,\mu}$ depending only σ.

The subsets τ and σ of Γ correspond to subsets

$$\hat{\tau} := \{i \in \mathcal{I} : \operatorname{supp}\varphi_i \subseteq \tau\},$$
$$\hat{\sigma} := \{j \in \mathcal{I} : \operatorname{supp}\varphi_j \subseteq \sigma\}$$

of the index set $\mathcal{I}$, and the degenerate approximation corresponds to the factorized approximation

$$G|_{\hat{\tau}\times\hat{\sigma}} \approx V_\tau S_{\tau\sigma} W_\sigma^\top \tag{3}$$

Dedicated to Wolfgang Hackbusch on the occasion of his 60th birthday.

Communicated by S. Sauter.

S. Börm (✉) · J. Bendoraityte
Max-Planck-Institut für Mathematik in den Naturwissenschaften,
Inselstraße 22–26, 04103 Leipzig, Germany
e-mail: sbo@mis.mpg.de

for the matrices given by

$$(V_\tau)_{iv} := \int_\Gamma \varphi_i(x) v_{\tau,v}(x)\,dx, \quad (S_{\tau\sigma})_{v\mu} := s_{\tau,\sigma,v,\mu}$$

$$(W_\sigma)_{j\mu} := \int_\Gamma \varphi_j(y) w_{\sigma,\mu}(y)\,dy$$

for all $i \in \hat{\tau}$, $j \in \hat{\sigma}$ and $v, \mu \in \{1, \ldots, k\}$. The factorized approximation requires only $(\#\hat{\tau} + k + \#\hat{\sigma})k$ units of storage, which can be far better than the $(\#\hat{\tau})(\#\hat{\sigma})$ units required by the standard array representation if k is small compared to $\#\hat{\tau}$ and $\#\hat{\sigma}$.

In most applications, we cannot find a degenerate approximation of the form (2) for the entire domains $\tau = \sigma = \Gamma$, but only for subdomains satisfying an *admissibility condition* that ensures that $g|_{\tau\times\sigma}$ is smooth enough to admit a usable degenerate approximation. The standard case is a kernel that has a singularity at $x = y$ and is locally analytic in the rest of the domain. In this situation, an admissibility condition of the type

$$\max\{\operatorname{diam}(\Omega_\tau), \operatorname{diam}(\Omega_\sigma)\} \leq \operatorname{dist}(\Omega_\tau, \Omega_\sigma) \tag{4}$$

is appropriate, where $\Omega_\tau \supseteq \tau$ and $\Omega_\sigma \supseteq \sigma$ are subsets of $\mathbb{R}^d$ that ensure that the degenerate approximation converges exponentially in k: if $\tilde{g}_{\tau,\sigma}$ is constructed by Taylor or multipole expansion, these subsets are usually spheres, if tensor interpolation is used, axis-parallel boxes are preferable.

In order to approximate the entire matrix G, we have to find a collection of subdomains $\tau \times \sigma$ of $\Gamma \times \Gamma$ such that the corresponding index sets $\hat{\tau} \times \hat{\sigma}$ form a disjoint block partition of G. This task is solved by a simple hierarchical approach: we construct a tree $\mathcal{T}_\mathcal{I}$ of subdomains with root $\varrho = \Gamma$ (and therefore $\hat{\varrho} = \mathcal{I}$) satisfying the following two conditions:

– for each $\tau \in \mathcal{T}_\mathcal{I}$ with $\operatorname{sons}(\tau) \neq \emptyset$, the index set $\hat{\tau}$ is the disjoint union of the index sets of its sons, i.e.,

$$\hat{\tau} = \dot{\bigcup}_{\tau' \in \operatorname{sons}(\tau)} \hat{\tau}' \quad \text{for all } \tau \in \mathcal{T}_\mathcal{I}, \operatorname{sons}(\tau) \neq \emptyset. \tag{5}$$

– For each $\tau \in \mathcal{T}_\mathcal{I}$ with $\operatorname{sons}(\tau) = \emptyset$, the index set $\hat{\tau}$ is "small", i.e., there is a constant λ satisfying

$$\#\hat{\tau} \leq \lambda \quad \text{for all } \tau \in \mathcal{T}_\mathcal{I}, \operatorname{sons}(\tau) = \emptyset. \tag{6}$$

A tree with these properties is called a *cluster tree*, its vertices are called *clusters*. Cluster trees for arbitrary geometries can be constructed by simple general algorithms [9].

We use the cluster tree to construct a partition of the matrix G into admissible blocks (and a small remainder): given a pair $\tau \times \sigma$ of clusters, we check whether the admissibility condition is satisfied. If this is the case, we call $\tau \times \sigma$ an *admissible block* and stop the procedure. Otherwise, we check whether we can split τ and σ into subdomains according to the structure of the cluster tree. If $\tau \times \sigma$ cannot be split any further, we call it an *inadmissible block* and stop the procedure. Otherwise, we split τ and σ and apply the procedure recursively to the resulting new pairs.

Starting with the root $\tau = \sigma = \varrho$ and collecting the admissible and inadmissible blocks in two sets P_{far} and P_{near} yields a partition $P_{\mathcal{I}\times\mathcal{I}} = P_{\text{far}} \dot{\cup} P_{\text{near}}$ with

$$\mathcal{I} \times \mathcal{I} = \dot{\bigcup}_{\tau\times\sigma \in P_{\mathcal{I}\times\mathcal{I}}} \hat{\tau} \times \hat{\sigma},$$

i.e., the necessary disjoint partition of $\mathcal{I} \times \mathcal{I}$ into a collection of admissible and inadmissible subblocks.

Using this partition (cf. Fig. 1), the approximation $\widetilde{G}$ (cf. (3)) of G is defined by

$$\widetilde{G}|_{\hat{\tau}\times\hat{\sigma}} := \begin{cases} V_\tau S_{\tau\sigma} W_\sigma^\top & \text{if } \tau \times \sigma \in P_{\text{far}} \\ G|_{\hat{\tau}\times\hat{\sigma}} & \text{otherwise} \end{cases}$$

for all $\tau \times \sigma \in P_{\mathcal{I}\times\mathcal{I}}$.

Since we store only the $k \times k$ matrix $S_{\tau\sigma}$ for an admissible block and the small $\hat{\tau} \times \hat{\sigma}$ matrix $G|_{\hat{\tau}\times\hat{\sigma}}$ for an inadmissible block, it can be shown [9,3] that the storage requirements for all of these matrices are in $\mathcal{O}(nk)$.

Unfortunately, the same does not hold for the *cluster bases* $(V_\tau)_{\tau\in\mathcal{T}_\mathcal{I}}$ and $(W_\sigma)_{\sigma\in\mathcal{T}_\mathcal{I}}$: given an index $i \in \mathcal{I}$, there is a cluster τ with $i \in \hat{\tau}$ on almost all levels of the cluster tree $\mathcal{T}_\mathcal{I}$. For each of these clusters, the matrix V_τ contains a row of k entries corresponding to the index i. In standard situations, the number of levels is proportional to $\log n$, therefore we have to store $\mathcal{O}(k \log n)$ entries per index and a total of $\mathcal{O}(nk \log n)$ entries.

We can improve the efficiency by requiring the function systems $(v_{\tau,v})$ and $(w_{\sigma,\mu})$ to be *nested*: we assume that for each $\tau \in \mathcal{T}_\mathcal{I}$ and $\tau' \in \operatorname{sons}(\tau)$, there is a *transfer matrix*

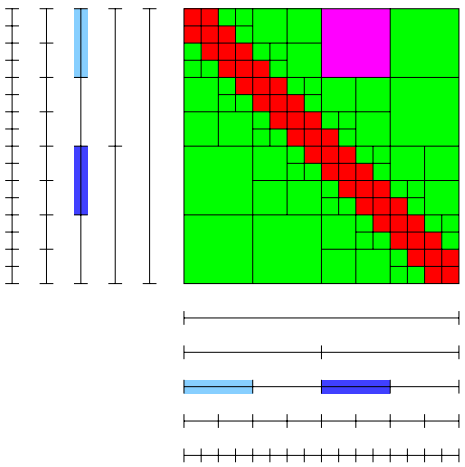

Fig. 1 Cluster tree and block partition for a simple one-dimensional problem. The clusters corresponding to one admissible block are marked

$E_{\tau'} \in \mathbb{R}^{k\times k}$ with

$$v_{\tau,\nu} = \sum_{\nu'=1}^{k} (E_{\tau'})_{\nu'\nu} v_{\tau',\nu'} \quad \text{for all } \nu \in \{1,\dots,k\}.$$

Typical polynomial and multipole bases fulfill this requirement, which allows us to observe

$$\begin{aligned}(V_\tau)_{i\nu} &= \int_\Gamma \varphi_i(x) v_{\tau,\nu}(x)\,dx \\ &= \sum_{\nu'=1}^{k} (E_{\tau'})_{\nu'\nu} \int_\Gamma \varphi_i(x) v_{\tau',\nu'}(x)\,dx \\ &= (V_{\tau'} E_{\tau'})_{i\nu} \quad \text{for all } i \in \hat{\tau}', \nu \in \{1,\dots,k\}. \end{aligned} \tag{7}$$

This means that we do not have to store V_τ, we only have to store the transfer matrices $E_{\tau'}$ for all $\tau' \in \operatorname{sons}(\tau)$ and take advantage of the nested structure.

Now we can extend this concept to the entire cluster basis (V_τ): we store $V_\tau \in \mathbb{R}^{\hat{\tau}\times k}$ only in leaf clusters and use the transfer matrices $(E_\tau)_{\tau\in\mathcal{T}_\mathcal{I}}$ for all other clusters. Since both types of matrix are small, it can be proven that an entire cluster basis in this representation requires only $\mathcal{O}(nk)$ units of storage [3].

We conclude that combining the factorized approximation of admissible blocks (3) with the factorized nested representation (7) of the cluster bases $(V_\tau)_{\tau\in\mathcal{T}_\mathcal{I}}$ and $(W_\sigma)_{\sigma\in\mathcal{T}_\mathcal{I}}$ leads to an approximation of G by an $\mathcal{H}^2$-*matrix* $\widetilde{G}$ [3,11].

Due to their good asymptotic complexity, $\mathcal{H}^2$-matrix techniques are attractive for large-scale computations dealing with many degrees of freedom and possibly complicated geometries. For this type of applications, distributed memory architectures are attractive, since they offer very good performance at relatively low costs, as long as the computations can be carried out in parallel and do not require too much data to be moved. In this paper, we present the first steps towards the implementation of $\mathcal{H}^2$-matrix algorithms on distributed memory architectures and prove that they can reach almost optimal parallel efficiency.

2 Original algorithms

Before we consider the parallel algorithms, we first recall the original sequential counterparts. In this paper, we will focus on the two most basic operations: the construction of the $\mathcal{H}^2$-matrix approximation and the efficient matrix-vector multiplication with a matrix in this representation.

A very simple and robust construction of an $\mathcal{H}^2$-matrix approximation of G is based on interpolation [4,8]: we fix interpolation points $(x_{\tau,\nu})_{\nu=1}^k$ and corresponding Lagrange polynomials $(\mathcal{L}_{\tau,\nu})_{\nu=1}^k$ for all clusters $\tau \in \mathcal{T}_\mathcal{I}$ and approximate g by its interpolant

$$\tilde{g}_{\tau,\sigma}(x,y) := \sum_{\nu=1}^{k}\sum_{\mu=1}^{k} g(x_{\tau,\nu}, x_{\sigma,\mu}) \mathcal{L}_{\tau,\nu}(x) \mathcal{L}_{\sigma,\mu}(y).$$

Finding the matrix $S_{\tau\sigma} \in \mathbb{R}^{k\times k}$ is now straightforward: each of its entries is just the evaluation of the kernel function in the k^2 tensor interpolation points, which can be assumed to require $\mathcal{O}(k^2)$ operations.

For the nearfield matrices and the matrices V_τ, we rely on quadrature techniques. We assume that one of the integrals can be approximated in not more than q operations. In simple situations, q can be a constant, in the general case q depends on n and k (cf. [5,16]).

Under this assumption, the construction of a nearfield matrix $G|_{\hat{\tau}\times\hat{\sigma}}$ for $\tau \times \sigma \in P_{\text{near}}$ requires the approximation of $(\#\hat{\tau})(\#\hat{\sigma})$ entries, each of which needs not more than q operations. Due to our construction, both τ and σ are leaf clusters, therefore (6) implies $\#\hat{\sigma} \leq \lambda$ and we get a bound of $q\lambda\#\hat{\tau}$ for the number of operations required for $G|_{\hat{\tau}\times\hat{\sigma}}$.

The matrices V_τ of the cluster basis are given by

$$(V_\tau)_{i\nu} = \int_\Gamma \varphi_i(x) \mathcal{L}_{\tau,\nu}(x)\,dx,$$

and for standard piecewise polynomial basis functions, the computation of one of these entries can be carried out by Gauss quadrature. Since V_τ has $(\#\hat{\tau})k$ entries, a total of $qk\#\hat{\tau}$ operations are required.

Since we are using interpolation to construct the approximation of the kernel, it is straightforward to use the same approach to construct the transfer matrices: if the same polynomial space is used for all clusters, interpolating $\mathcal{L}_{\tau,\nu}$ in the points corresponding to τ' yields

$$\mathcal{L}_{\tau,\nu} = \sum_{\nu'=1}^{k} \mathcal{L}_{\tau,\nu}(x_{\tau',\nu'}) \mathcal{L}_{\tau',\nu'}$$

and the transfer matrix $E_{\tau'}$ for $\tau \in \mathcal{T}_\mathcal{I}$ and $\tau' \in \operatorname{sons}(\tau)$ is therefore given by

$$(E_{\tau'})_{\nu'\nu} = \mathcal{L}_{\tau,\nu}(x_{\tau',\nu'}).$$

In general, the evaluation of the Lagrange polynomials may require more than $\mathcal{O}(1)$ operations, but in the special case of tensor interpolation, we can prepare d matrices corresponding to one-dimensional evaluations and use their tensor product to compute $E_{\tau'}$ in $\mathcal{O}(k^2)$ operations.

With a few simple assumptions (cf. Sect. 4), these estimates imply that the entire $\mathcal{H}^2$-matrix representation can be constructed in $\mathcal{O}((q+1)nk)$ operations.

It is not enough to store the matrix, we also have to work with it, and the simplest and most important operation is the matrix-vector multiplication, i.e., the computation of $y = \widetilde{G}x$ for $x, y \in \mathbb{R}^\mathcal{I}$.

Let us consider only the multiplication with one admissible block

$$\widetilde{G}|_{\hat{\tau}\times\hat{\sigma}} = V_\tau S_{\tau\sigma} W_\sigma^\top$$

for $\tau\times\sigma \in P_{\text{far}}$. Obviously, we can split the computation into three steps: first we evaluate $\hat{x}_\sigma := W_\sigma^\top x|_{\hat{\sigma}}$, then we compute $\hat{y}_\tau := S_{\tau\sigma}\hat{x}_\sigma$, and finally we get the result $y|_{\hat{\tau}} := V_\tau \hat{y}_\tau$.

Due to the nested representation, W_σ is only at our disposal in leaf clusters. For a cluster $\sigma \in \mathcal{T}_\mathcal{I}$ with $\operatorname{sons}(\sigma) \neq \emptyset$, we therefore have to use the transfer matrices:

$$\begin{aligned}(W_\sigma^\top x|_{\hat{\sigma}})_\mu &= \sum_{j\in\hat{\sigma}} (W_\sigma)_{j\mu} x_j = \sum_{\sigma'\in\operatorname{sons}(\sigma)} \sum_{j\in\hat{\sigma}'} (W_\sigma)_{j\mu} x_j \\ &= \sum_{\sigma'\in\operatorname{sons}(\sigma)} \sum_{j\in\hat{\sigma}'} (W_{\sigma'} E_{\sigma'})_{j\mu} x_j \\ &= \sum_{\sigma'\in\operatorname{sons}(\sigma)} \sum_{\mu'=1}^{k} (E_{\sigma'})_{\mu'\mu} (W_{\sigma'}^\top x|_{\hat{\sigma}'})_{\mu'} \\ &= \sum_{\sigma'\in\operatorname{sons}(\sigma)} (E_{\sigma'}^\top \hat{x}_{\sigma'}),\end{aligned}$$

therefore we can compute *all* vectors $(\hat{x}_\sigma)_{\sigma\in\mathcal{T}_\mathcal{I}}$ by the simple recursion

procedure forward(σ);
if $\operatorname{sons}(\sigma) = \emptyset$ **then** $\hat{x}_\sigma \leftarrow W_\sigma^\top x|_{\hat{\sigma}}$
else begin
  $\hat{x}_\sigma \leftarrow 0$;
  for $\sigma' \in \operatorname{sons}(\sigma)$ **do begin**
    forward(σ'); $\hat{x}_\sigma \leftarrow \hat{x}_\sigma + E_{\sigma'}^\top \hat{x}_{\sigma'}$
end end

called the *forward transformation algorithm*.

In the next step, we have to compute $\hat{y}_\tau$. For the sake of efficiency, we accumulate *all* contributions to it:

$$\hat{y}_\tau := \sum_{\substack{\sigma\in\mathcal{T}_\mathcal{I}\\ \tau\times\sigma\in P_{\text{far}}}} S_{\tau\sigma}\hat{x}_\sigma.$$

In the last step, we have to evaluate V_τ. We once more use the transfer matrices in order to deal with non-leaf clusters and get the recursion

procedure backward(τ);
if $\operatorname{sons}(\sigma) = \emptyset$ **then** $y|_{\hat{\tau}} \leftarrow y|_{\hat{\tau}} + V_\tau \hat{y}_\tau$
else begin
  for $\tau' \in \operatorname{sons}(\tau)$ **do begin**
    $\hat{y}_{\tau'} \leftarrow \hat{y}_{\tau'} + E_{\tau'}\hat{y}_\tau$; backward($\tau$)
end end

called the *backward transformation algorithm*. These three steps take care of the admissible blocks; the inadmissible blocks are small and can be handled directly. Since not more than two operations are performed for each entry of the $\mathcal{H}^2$-matrix representation, the total number of operations is in the same class as the total amount of storage, i.e., in $\mathcal{O}(nk)$.

3 Parallelization

Our goal is to find parallel variants of the algorithms introduced in the previous section that reduce the total runtime as far as possible.

We use a simple model of a distributed memory architecture: we assume that we have $p \in \mathbb{N}$ individual *processing nodes* denoted by indices from the set $\mathcal{N} := \{1, \ldots, p\}$, each of which contains a quantity of local storage and can manipulate this storage by running algorithms, unaffected by the operations of the other nodes.

In addition, each node can communicate with each other node by sending and receiving data. To keep the algorithms simple, we assume that send operations are *non-blocking*, i.e., that the sending node does not wait until the matching receive operation on the target node has been completed. Of course, the receive operation has to be *blocking*, i.e., the node executing it has to wait until the matching send operation has been completed and its data is available.

In practice, a node will typically be realized as a process on a networked computer that uses a suitable communications library (e.g., the MPI standard [7]) to exchange information between processes on the same or different computers in the network.

In order to use $\mathcal{H}^2$-matrices efficiently on a distributed memory architecture, we have to parallelize the algorithms in such a way that the part of the algorithm that executes on a node has most of the necessary information in local storage and that receive operations do not have to wait too long for the matching send operations. Since communication between the nodes is usually slower than accesses to local storage, we also would like to keep the number of communication steps low.

Our approach is motivated by the forward and backward transformation algorithms: the computation of $\hat{x}_\sigma$ requires only information connected to the sons of the cluster σ. If we ensure that most of the sons reside on the same node as σ itself, only a small number of communication steps are required.

We therefore follow a cluster-centric approach: we assign a *responsible* node $\mathcal{R}_\tau \in \mathcal{N}$ to each cluster τ. The responsible node for τ stores

- the matrices V_τ and W_τ if τ is a leaf,
- the matrices $E_{\tau'}$ for all $\tau' \in \operatorname{sons}(\tau)$ if τ is not a leaf,
- the matrices $S_{\tau\sigma}$ for all $\sigma \in \mathcal{T}_\mathcal{I}$ with $\tau\times\sigma \in P_{\text{far}}$ and
- the matrices $G|_{\hat{\tau}\times\hat{\sigma}}$ for all $\sigma \in \mathcal{T}_\mathcal{I}$ with $\tau\times\sigma \in P_{\text{near}}$.

All algorithms introduced so far work by recursion on subtrees of the cluster tree $\mathcal{T}_\mathcal{I}$. Even if a node α is not responsible for a cluster τ, and therefore does not have to perform any computations connected to this cluster, it may have to pass through τ in order to reach clusters on a deeper level of $\mathcal{T}_\mathcal{I}$ for which it is responsible.

To put this more precisely, we denote the full subtree of $\mathcal{T}_\mathcal{I}$ with root τ by $\mathcal{T}_\mathcal{I}^\tau$ and call a cluster $\tau^* \in \mathcal{T}_\mathcal{I}$ a *descendant* of τ if $\tau^* \in \mathcal{T}_\mathcal{I}^\tau$ holds. The node α has to pass through τ if there is a descendant τ^* of τ for which it is responsible. In this situation, we say that α is *active* in the cluster τ, and we denote the *set of active nodes* by

$$\mathcal{A}_\tau := \{\alpha \in \mathcal{N} \ : \ \text{there exists } \tau^* \in \mathcal{T}_\mathcal{I}^\tau \text{ with } \mathcal{R}_{\tau^*} = \alpha\}.$$

Using responsible and active nodes (cf. Fig. 2), we can now develop parallel variants of the algorithms introduced in the previous section.

The cluster bases $(V_\tau)_{\tau\in\mathcal{T}_\mathcal{I}}$ and $(W_\sigma)_{\sigma\in\mathcal{T}_\mathcal{I}}$ can be constructed by running the following algorithm on all nodes $\alpha \in \mathcal{N}$:

procedure par_basis(τ);
if $\alpha \in \mathcal{A}_\tau$ **then**
  for $\tau' \in$ sons(τ) **do** par_basis(τ');
if $\alpha = \mathcal{R}_\tau$ **then begin**
  if sons(τ) $= \emptyset$ **then**
    Prepare V_τ and W_τ
  else
    for $\tau' \in$ sons(τ) **do** Prepare $E_{\tau'}$
end

The algorithm is a straightforward recursion: if the current node α is active in τ, a recursive call takes care of clusters in the subtree $\mathcal{T}_\mathcal{I}^\tau$ for which α is responsible. If α is responsible for τ, the matrices of the cluster bases are constructed.

For the admissible and inadmissible blocks (cf. Fig. 3), we use a similar recursion:

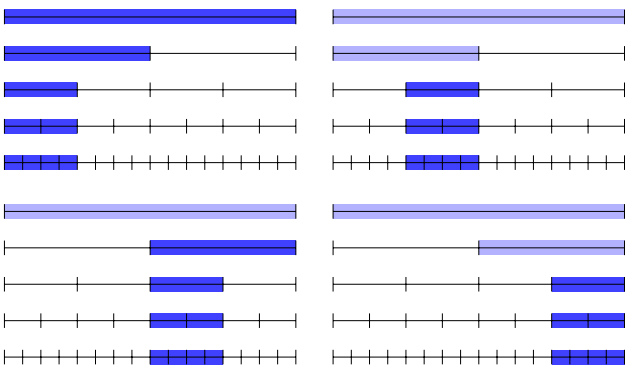

Fig. 2 Responsible (*dark blue*) and active (*light blue*) nodes for a simple one-dimensional cluster tree with $p = 4$ processing nodes

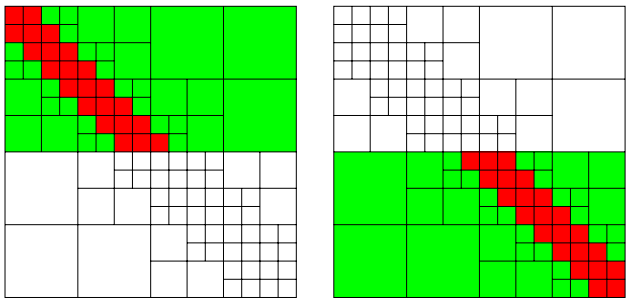

Fig. 3 Distributed representation of blocks for $p = 2$ nodes

procedure par_matrix($\tau \times \sigma$);
if $\tau \times \sigma \in P_{\mathcal{I}\times\mathcal{I}}$ **then begin**
  if $\alpha = \mathcal{R}_\tau$ **then begin**
    if $\tau \times \sigma \in P_{\text{far}}$ **then**
      Prepare $S_{\tau\sigma}$
    else
      Prepare $G|_{\hat{\tau}\times\hat{\sigma}}$
  end
end else
  if $\alpha \in \mathcal{A}_\tau$ **then**
    for $\tau' \in$ sons(τ), $\sigma' \in$ sons(σ) **do** par_matrix($\tau' \times \sigma'$)

This procedure first checks if it has reached an element of $P_{\mathcal{I}\times\mathcal{I}}$. In this case, the relevant matrices are prepared if the current node α is responsible for the row cluster τ. If $\tau \times \sigma$ is not an element of $P_{\mathcal{I}\times\mathcal{I}}$ and if α is active, we proceed recursively to the sons of τ and σ.

We can see that a node $\alpha \in \mathcal{N}$ performs no arithmetic operations in the algorithms "par_basis" and "par_matrix" if it is not responsible for the cluster τ. If we can ensure that each of the nodes is responsible for roughly the same number of clusters, we can expect very good parallel efficiency.

If we assume that the cluster tree $\mathcal{T}_\mathcal{I}$ and the partition $P_{\mathcal{I}\times\mathcal{I}}$ are available on all nodes, the construction of the $\mathcal{H}^2$-matrix $\widetilde{G}$ is perfectly parallelizable since it requires no communication and all arithmetic operations can be distributed almost perfectly among the processing nodes.

The forward transformation, on the other hand, requires communication: even if a node α is responsible for a cluster σ, it is not necessarily responsible for all sons of σ, therefore the vectors $\hat{x}_{\sigma'}$ for $\sigma' \in$ sons(σ) with $\mathcal{R}_{\sigma'} \neq \alpha$ have to be transferred from the node $\mathcal{R}_{\sigma'}$ to the node α.

We modify the forward transformation algorithm to take care of the special case:

procedure par_forward(σ);
if $\alpha \in \mathcal{A}_\sigma$ **then begin**
  for $\sigma' \in$ sons(σ) **do begin**
    par_forward(σ');
    if $\alpha \neq \mathcal{R}_\sigma$ and $\alpha = \mathcal{R}_{\sigma'}$ **then**
      Send $\hat{x}_{\sigma'}$ to node $\mathcal{R}_\sigma$ without blocking
  end;
  for $\sigma' \in$ sons(σ) **do** (*)

```
    if α = R_σ and α ≠ R_σ' then
      Receive x̂_σ' from node R_σ'
end;
if α = R_σ then begin
  if sons(σ) = ∅ then x̂_σ ← W_σ^⊤ x|_σ̂
  else begin
    x̂_σ ← 0;
    for σ' ∈ sons(σ) do x̂_σ ← x̂_σ + E_σ'^⊤ x̂_σ'
end end
```

If the current node α is active in σ, recursive calls for the sons of σ' are necessary, since they may contain clusters for which α is responsible. Communication is required if different nodes are responsible for σ and a son σ': if $\mathcal{R}_\sigma \neq \mathcal{R}_{\sigma'}$ holds, the node $\mathcal{R}_{\sigma'}$ has to send $\hat{x}_{\sigma'}$ to the node $\mathcal{R}_\sigma$ (cf. Fig. 5).

The backward transformation algorithm can be treated in a similar way: if $\mathcal{R}_\tau \neq \mathcal{R}_{\tau'}$ holds, the node $\mathcal{R}_\tau$ has to send the vector $\hat{y}_{\tau'}$ to the node $\mathcal{R}_{\tau'}$. The parallel backward transformation algorithm takes the following form:

```
procedure par_backward(τ);
if α = R_τ then begin
  if sons(τ) = ∅ then y|_τ̂ ← y|_τ̂ + V_τ ŷ_τ
  else
    for τ' ∈ sons(τ) do begin
      if α = R_τ' then ŷ_τ' ← ŷ_τ' + E_τ' ŷ_τ
                   else y̌_τ' ← E_τ' ŷ_τ
    end
end;
if α ∈ A_τ then begin
  for τ' ∈ sons(τ) do
    if α = R_τ and α ≠ R_τ' then
      Send y̌_τ' to node R_τ' without blocking;
  for τ' ∈ sons(τ) do begin                    (*)
    if α ≠ R_τ and α = R_τ' then begin
      Receive y̌_τ' from node R_τ;
      ŷ_τ' ← ŷ_τ' + y̌_τ'
    end;
    par_backward(τ')
end end
```

Here we have to handle a special case: the backward transformation adds the transformed values of the father to the sons, therefore we need an auxiliary vector $\check{y}_{\tau'}$ to receive the contribution of a father cluster to a son cluster in case the node is responsible for τ' but not for τ.

The parallel forward and backward transformations require a communication step if different processing nodes are responsible for father and son clusters, and since communication can be very time-consuming, we should try to reduce the number of these situations as far as possible. Therefore, we are interested in ensuring that nodes are responsible for entire subtrees, since this means that no communication is required to perform operations within the subtrees.

The most communication-intensive parts of the matrix-vector multiplication are the multiplication by the matrices $S_{\tau\sigma}$ for admissible and $G|_{\hat{\tau}\times\hat{\sigma}}$ for inadmissible blocks $\tau\times\sigma$: if the processing node α is responsible for τ, but not for σ, the vectors $\hat{x}_\sigma$ or $x|_{\hat{\sigma}}$, respectively, have to transferred from the node $\mathcal{R}_\sigma$ to the node α.

In typical situations, the vector $\hat{x}_\sigma$ may be required for more than 50 admissible blocks, therefore it is more efficient to broadcast it only once to all "interested parties" in a separate step.

A node α needs the vector $\hat{x}_\sigma$ if it has to compute the product $S_{\tau\sigma}\hat{x}_\sigma$ for a cluster τ, and it has to compute this product if it is responsible for τ and if $\tau\times\sigma \in P_{\mathrm{far}}$. If the node α is not responsible for σ, the vector $\hat{x}_\sigma$ is not locally available, and it has to be transmitted from the responsible node $\mathcal{R}_\sigma$.

We collect all clusters that have to be sent from a node $\beta \in \mathcal{N}$ to a node $\alpha \in \mathcal{N}$ in the set

$$\mathcal{T}_{\alpha\beta} := \big\{\sigma \in \mathcal{T}_\mathcal{I} \;:\; \text{there exists } \tau \in \mathcal{T}_\mathcal{I} \text{ with } \tau\times\sigma \in P_{\mathcal{I}\times\mathcal{I}}, \mathcal{R}_\tau = \alpha, \mathcal{R}_\sigma = \beta\big\}$$

(cf. Fig. 4) and see that now the broadcast step can be easily handled by the following procedure:

```
procedure broadcast;
for β ∈ N\{α} do begin
  for σ ∈ T_βα do begin
    Send x̂_σ to node β without blocking;
    if sons(σ) = ∅ then
      Send x|_σ̂ to node β without blocking
end end;
```

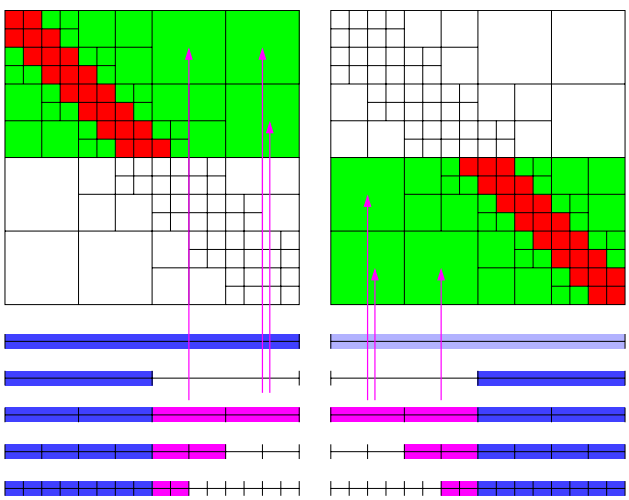

Fig. 4 Transfer sets $\mathcal{T}_{\alpha\beta}$ for $p = 2$ nodes. Clusters that have to be obtained from another node are shown in magenta. Magenta arrows point to the blocks that requires these clusters

for $\beta \in \mathcal{N}\backslash\{\alpha\}$ **do begin** (*)
  for $\sigma \in \mathcal{T}_{\alpha\beta}$ **do begin**
    Receive $\hat{x}_\sigma$ from node β;
    if sons$(\sigma) = \emptyset$ **then**
      Receive $x|_{\hat{\sigma}}$ from node β
end end

For the sake of simplicity, we use *non-blocking* send operations to avoid having to worry about possible deadlocks: since all send operations in all nodes are performed first and without waiting for completion, no subsequent receive operations will be starved.

The sets $\mathcal{T}_{\alpha\beta}$ describing the necessary transmissions between nodes can be constructed by a simple recursive pass through the block partition:

procedure discover($\tau \times \sigma$);
if $\tau \times \sigma \in P_{\mathcal{I}\times\mathcal{I}}$ **then begin**
  if $\alpha = \mathcal{R}_\tau$ **then begin**
    $\beta \leftarrow \mathcal{R}_\sigma$; $\mathcal{T}_{\alpha\beta} \leftarrow \mathcal{T}_{\alpha\beta} \cup \{\sigma\}$
  end;
  if $\alpha = \mathcal{R}_\sigma$ **then begin**
    $\beta \leftarrow \mathcal{R}_\tau$; $\mathcal{T}_{\beta\alpha} \leftarrow \mathcal{T}_{\beta\alpha} \cup \{\sigma\}$
  end
end else
  if $\alpha \in \mathcal{A}_\tau$ or $\alpha \in \mathcal{A}_\sigma$ **then**
    for $\tau' \in$ sons(τ), $\sigma' \in$ sons(σ) **do** discover($\tau' \times \sigma'$)

In a practical implementation, the sets $\mathcal{T}_{\alpha\beta}$ can be realized as lists, and if the lists for $\mathcal{T}_{\alpha\beta}$ and $\mathcal{T}_{\beta\alpha}$ are constructed in the same order by the "discover" algorithm and processed in the same order by the "broadcast" algorithm, the send and receive operations match and no additional synchronization is required.

Once all necessary vectors $\hat{x}_\sigma$ are available in a node α, the actual multiplication can take place:

procedure par_multiply($\tau \times \sigma$);
if $\alpha = \mathcal{R}_\tau$ **then begin**
  if $\tau \times \sigma \in P_{\text{far}}$ **then** $\hat{y}_\tau \leftarrow \hat{y}_\tau + S_{\tau\sigma}\hat{x}_\sigma$;
  if $\tau \times \sigma \in P_{\text{near}}$ **then** $y|_{\hat{\tau}} \leftarrow y|_{\hat{\tau}} + G|_{\hat{\tau}\times\hat{\sigma}}x|_{\hat{\sigma}}$
if $\alpha \in \mathcal{A}_\tau$ **then**
  for $\tau' \in$ sons(τ), $\sigma' \in$ sons(σ) **do** par_multiply($\tau' \times \sigma'$)

The parallel matrix-vector multiplication consists of four steps: the forward transformation "par_forward", the communication step "broadcast", the actual multiplication step "par_multiply", and finally the backward transformation "par_backward". In preparation, we need to use "discover" once to initialize the sets $\mathcal{T}_{\alpha\beta}$ on all nodes, but this procedure is not part of the actual matrix-vector multiplication.

4 Complexity

We have seen that the $\mathcal{H}^2$-matrix requires $\mathcal{O}(nk)$ units of storage and that one matrix-vector multiplication can be performed in $\mathcal{O}(nk)$ operations.

The distributed $\mathcal{H}^2$-matrix approach can be considered successful if the storage requirements *per processing node* and the time required to complete the algorithms is reduced significantly. We will now prove that our techniques comes very close to the optimal parallel efficiency if the problem dimension n is significantly larger than the number p of processing nodes.

4.1 Assignment of responsible nodes

In order to keep the presentation simple, we restrict our attention to the analysis of a relatively simple model.

Assumption 1 (Nodes) We assume that p is a power of two, i.e., that there is an integer $\ell \in \mathbb{N}_0$ with $p = 2^\ell$.

A look at the parallel forward and backward transformation reveals that communication steps are only required if different nodes are responsible for father and son clusters. An ideal situation is a node that is responsible for an entire subtree, since then all computations within the subtree can be carried out without any communication.

Therefore, we assume that the cluster tree $\mathcal{T}_\mathcal{I}$ can be split into p subtrees for the p available processing nodes and a small prefix that handles the communication between the nodes. The prefix can be defined easily by splitting $\mathcal{T}_\mathcal{I}$ into levels: for all $\tau \in \mathcal{T}_\mathcal{I}$ the *level* of τ is defined by

$$\text{level}(\tau) := \begin{cases} \text{level}(\tau^+) + 1 & \text{if there exists } \tau^+ \in \mathcal{T}_\mathcal{I} \\ & \text{with } \tau \in \text{sons}(\tau^+), \\ 0 & \text{otherwise.} \end{cases}$$

Now we can describe our assumptions with respect to the cluster tree: we require the prefix to be a binary tree and the subtrees to be of approximately equal size:

Assumption 2 (Tree) We assume that

- the first ℓ levels of $\mathcal{T}_\mathcal{I}$ form a binary tree, i.e.,

$$\#\,\text{sons}(\tau) = 2 \quad \text{for all } \tau \in \mathcal{T}_\mathcal{I} \text{ with } \text{level}(\tau) < \ell,$$

- that the sizes of the subtrees starting on level ℓ are comparable, i.e., that there is a constant C_{st} independent of n, k and p such that we have

$$\#\mathcal{T}_\mathcal{I}^\tau \leq C_{\text{st}}\frac{n}{kp} \quad \text{and}$$
$$\#\hat{\tau} \leq C_{\text{st}}\frac{n}{p} \quad \text{for all } \tau \in \mathcal{T}_\mathcal{I} \text{ with } \text{level}(\tau) = \ell.$$

Given a tree satisfying these assumptions, it is an easy task to assign each cluster to a node while guaranteeing that the amount of communication steps is minimized: our construction is based on the set $\mathcal{A}_\tau$ of active nodes. For the root cluster ϱ, the obvious choice is $\mathcal{A}_\varrho = \mathcal{N}$, since a node missing from $\mathcal{A}_\varrho$ would not be responsible for any cluster and therefore perform no computations at all.

Once $\mathcal{A}_\tau$ has been found for a cluster τ, we can construct the sets for the sons of τ: if $\mathcal{A}_\tau$ contains only one node, the same node will be responsible for the sons of τ. If $\mathcal{A}_\tau$ contains more than one node, we split the set into subsets for the sons of τ. In order to keep the presentation simple, we restrict our attention to contiguous sets

$$\mathcal{A}_\tau = \{q_\tau, \ldots, q_\tau + \max\{0, p_\tau - 1\}\}$$

for integers q_τ and p_τ and get the following algorithm:

procedure setup_responsibility(τ, q_τ, p_τ);
$\mathcal{R}_\tau \leftarrow q_\tau$;
$\mathcal{A}_\tau \leftarrow \{q_\tau, \ldots, q_\tau + \max\{0, p_\tau - 1\}\}$;
if sons(τ) $\neq \emptyset$ **then begin**
  $\{\tau_1, \tau_2\} \leftarrow$ sons(τ);
  $p' \leftarrow \lfloor p_\tau/2 \rfloor$; $q_1 \leftarrow q_\tau$; $q_2 \leftarrow q_\tau + p'$;
  setup_responsibility(τ_1, q_1, p');
  setup_responsibility($\tau_2, q_2, p_\tau - p'$)
end

This algorithm ensures that subtrees below level ℓ are handled by exactly one node and that each node is only responsible for not more than one cluster above level ℓ.

Lemma 1 (Responsibilities) *Let $\tau \in \mathcal{T}_\mathcal{I}$.*

- *If* level(τ) $\leq \ell$, *we have*

$$\mathcal{A}_\tau \cap \mathcal{A}_\sigma = \emptyset \quad \text{for all } \sigma \in \mathcal{T}_\mathcal{I}, \operatorname{level}(\sigma) = \operatorname{level}(\tau).$$

- *If* level(τ) $\geq \ell$, *we have*

$$\mathcal{R}_\tau = \mathcal{R}_{\tau^*} \quad \text{for all } \tau^* \in \mathcal{T}_\mathcal{I}^\tau.$$

- *For each processing node $\alpha \in \mathcal{N}$, there is a unique cluster τ_α with*

$$\operatorname{level}(\tau_\alpha) = \ell \quad \text{and} \quad \mathcal{R}_{\tau_\alpha} = \alpha. \tag{8}$$

- *For each processing node $\alpha \in \mathcal{N}$, we have*

$$\#\{\tau \in \mathcal{T}_\mathcal{I} : \mathcal{R}_\tau = \alpha\} \leq C_{\mathrm{st}} \frac{n}{kp} + \log_2 p. \tag{9}$$

Proof Algorithm "setup_responsibility" starts the recursion with $p_\varrho = p$, and due to Assumption 1, we get

$$p_\tau = \lfloor p 2^{-\operatorname{level}(\tau)} \rfloor = \lfloor 2^{\ell - \operatorname{level}(\tau)} \rfloor \quad \text{for all } \tau \in \mathcal{T}_\mathcal{I}.$$

We consider the results of the algorithm for a cluster $\tau \in \mathcal{T}_\mathcal{I}$ with level(τ) $< \ell$. Since τ belongs to a low level, we have $p_\tau \geq 2$ and therefore $p' = \lfloor p_\tau/2 \rfloor \geq 1$. We conclude that the sets $\mathcal{A}_{\tau_1}$ and $\mathcal{A}_{\tau_2}$ are disjoint. By a simple induction, this implies the first claim.

Let us now consider a cluster $\tau \in \mathcal{T}_\mathcal{I}$ with level(τ) $\geq \ell$. Due to level(τ) $\geq \ell$, we have $p_\tau \leq 1$ and therefore $\#\mathcal{A}_\tau = 1$. Due to the definition of $\mathcal{A}_\tau$, this means that the processing node $\mathcal{R}_\tau$ is responsible for all clusters in $\mathcal{T}_\mathcal{I}^\tau$, and we have proven the second claim.

According to Assumption 2, there are exactly $2^\ell = p$ clusters on level ℓ, and we have proven that none of them are the responsibility of the same node. Since there are not more than p nodes, this implies the third claim.

Let $\alpha \in \mathcal{N}$. The first claim implies

$$\#\{\tau \in \mathcal{T}_\mathcal{I} : \alpha \in \mathcal{A}_\tau, \operatorname{level}(\tau) = i\} = 1$$

for all $i \in \{0, \ldots, \ell\}$. Combining this with the second and third claim and Assumption 2 yields

$$\begin{aligned}
&\#\{\tau \in \mathcal{T}_\mathcal{I} : \alpha \in \mathcal{A}_\tau\} \\
&\quad = \sum_{i=0}^{\infty} \#\{\tau \in \mathcal{T}_\mathcal{I} \,:\, \alpha \in \mathcal{A}_\tau, \operatorname{level}(\tau) = i\} \\
&\quad = \sum_{i=0}^{\ell-1} \#\{\tau \in \mathcal{T}_\mathcal{I} \,:\, \alpha \in \mathcal{A}_\tau, \operatorname{level}(\tau) = i\} + \#\mathcal{T}_\mathcal{I}^{\tau_\alpha} \\
&\quad \leq \ell + C_{\mathrm{st}} \frac{n}{kp} = C_{\mathrm{st}} \frac{n}{kp} + \log_2 p.
\end{aligned}$$

This is the estimate we need. □

In order to find bounds for the storage complexity for the matrices $S_{\tau\sigma}$ and $G|_{\hat{\tau}\times\hat{\sigma}}$ corresponding to the blocks $\tau \times \sigma \in P_{\mathcal{I}\times\mathcal{I}}$, we rely on the concept of *sparse partitions* [9] to bound the number of blocks connected to one cluster. By definition, the matrices are only stored in the processing node $\alpha \in \mathcal{N}$ if α is responsible for τ, so the results of Lemma 1 allow us to derive the necessary estimates.

Assumption 3 (Sparsity) For all clusters $\tau \in \mathcal{T}_\mathcal{I}$, we let

$$\operatorname{row}(\tau) := \{\sigma \in \mathcal{T}_\mathcal{I} \,:\, \tau \times \sigma \in P_{\mathcal{I}\times\mathcal{I}}\},$$
$$\operatorname{col}(\tau) := \{\sigma \in \mathcal{T}_\mathcal{I} \,:\, \sigma \times \tau \in P_{\mathcal{I}\times\mathcal{I}}\}.$$

We assume that there is a constant $C_{\mathrm{sp}} \in \mathbb{N}$ independent of n, p and k such that

$$\#\operatorname{row}(\tau) \leq C_{\mathrm{sp}}, \ \#\operatorname{col}(\tau) \leq C_{\mathrm{sp}} \quad \text{for all } \tau \in \mathcal{T}_\mathcal{I}.$$

Estimates for C_{sp} in general situations can be found in [9].

4.2 Storage complexity

In the distributed algorithm, the matrices corresponding to a cluster τ or only stored in a processing node α if α is responsible for τ. This allows us to combine Assumption 2 with Lemma 1 to prove estimates for the storage requirements per node.

Lemma 2 (Cluster basis) *Let $\alpha \in \mathcal{N}$ be a processing node. The distributed representation of the cluster basis $(V_\tau)_{\tau\in\mathcal{T}_\mathcal{I}}$ requires $\mathcal{O}(nk/p + k^2 \log_2 p)$ units of storage in the node α.*

Proof Let $\tau \in \mathcal{T}_\mathcal{I}$ with $\mathcal{R}_\tau = \alpha$. If τ is a leaf, the processing node α stores the matrix $V_\tau \in \mathbb{R}^{\hat{\tau}\times k}$ and requires $(\#\hat{\tau})k$ units of storage. Due to Assumption 2, leaf clusters cannot appear on levels less than ℓ, and Lemma 1 yields that τ has to be a descendant of the cluster τ_α defined in (8). The leaves of $\mathcal{T}_\mathcal{I}^{\tau_\alpha}$ correspond to a disjoint partition of $\hat{\tau}_\alpha$, and we conclude that all matrices V_τ stored in the node α require

$$\sum_{\substack{\#\hat{\tau}\in\mathcal{T}_\mathcal{I}^{\tau_\alpha}\\ \operatorname{sons}(\tau)=\emptyset}} (\#\hat{\tau})k = (\#\hat{\tau}_\alpha)k \leq C_{\mathrm{st}}\frac{nk}{p} \quad \text{units of storage.}$$

If τ is not a leaf, we have to store the transfer matrices $E_{\tau'}$ for all sons τ' of τ. If $\operatorname{level}(\tau) < \ell$, Assumption 2 yields $\#\operatorname{sons}(\tau) = 2$ and the two transfer matrices require $2k^2$ units of storage, which leads to a total of

$$\sum_{\substack{\tau\in\mathcal{T}_\mathcal{I}\\ \mathcal{R}_\tau=\alpha,\operatorname{level}(\tau)<\ell}} 2k^2 = 2\ell k^2 = 2k^2\log_2 p \quad \text{units of storage.}$$

If $\operatorname{level}(\tau) \geq \ell$, α is also responsible for the sons of τ and we get

$$\sum_{\substack{\tau\in\mathcal{T}_\mathcal{I}\\ \mathcal{R}_\tau=\alpha,\operatorname{level}(\tau)\geq\ell}} \sum_{\tau'\in\operatorname{sons}(\tau)} k^2 = \sum_{\substack{\tau\in\mathcal{T}_\mathcal{I}\\ \mathcal{R}_\tau=\alpha,\operatorname{level}(\tau)>\ell}} k^2 < (\#\mathcal{T}_\mathcal{I}^{\tau_\alpha})k^2 \leq C_{\mathrm{st}}\frac{nk}{p} \quad \text{units of storage.}$$

Adding the three estimates yields a bound of

$$2C_{\mathrm{st}}\frac{nk}{p} + 2k^2\log_2 p \quad \text{units of storage,}$$

and this is the upper bound we need. □

In order to find a bound for the storage requirements of the complete $\mathcal{H}^2$-matrix, we need Assumption 3:

Lemma 3 (Block matrices) *Let $\alpha \in \mathcal{N}$ be a processing node. The distributed representation of the farfield matrices $(S_{\tau\sigma})_{\tau\times\sigma\in P_{\mathrm{far}}}$ and the nearfield matrices $(G|_{\hat{\tau}\times\hat{\sigma}})_{\tau\times\sigma\in P_{\mathrm{near}}}$ requires $\mathcal{O}(n(k+\lambda)/p + k^2\log_2 p)$ units of storage in the node α.*

Proof Let $\tau \in \mathcal{T}_\mathcal{I}$ with $\mathcal{R}_\tau = \alpha$. According to Assumption 3, there are not more than C_{sp} blocks of the form $\tau\times\sigma$ for $\sigma \in \operatorname{row}(\tau)$. For one $\sigma \in \operatorname{row}(\tau)$, the block $\tau\times\sigma$ can be either admissible or inadmissible. If it is admissible, the node α has to store the matrix $S_{\tau\sigma}$, and this requires k^2 units of storage. If it is not admissible, the node stores $G|_{\hat{\tau}\times\hat{\sigma}}$, and since inadmissible blocks only appear if τ and σ are leaves, this requires not more than $\lambda\#\hat{\tau}$ units of storage.

If $\operatorname{level}(\tau) \geq \ell$, Lemma 1 implies $\tau \in \mathcal{T}_\mathcal{I}^{\tau_\alpha}$, and we conclude that all matrices stored in node α require not more than

$$\begin{aligned}
&\sum_{\substack{\tau\times\sigma\in P_{\mathrm{far}}\\ \mathcal{R}_\tau=\alpha}} k^2 + \sum_{\substack{\tau\times\sigma\in P_{\mathrm{near}}\\ \mathcal{R}_\tau=\alpha}} \lambda\#\hat{\tau}\\
&\quad\leq \sum_{\substack{\tau\in\mathcal{T}_\mathcal{I}\\ \mathcal{R}_\tau=\alpha}} \sum_{\substack{\sigma\in\operatorname{row}(\tau)\\ \mathcal{R}_\tau=\alpha}} k^2 + \sum_{\substack{\tau\in\mathcal{T}_\mathcal{I}\\ \mathcal{R}_\tau=\alpha,\operatorname{sons}(\tau)=\emptyset}} \sum_{\substack{\sigma\in\operatorname{row}(\tau)\\ \mathcal{R}_\tau=\alpha}} \lambda\#\hat{\tau}\\
&\quad\leq C_{\mathrm{sp}} \sum_{\substack{\tau\in\mathcal{T}_\mathcal{I}\\ \mathcal{R}_\tau=\alpha}} k^2 + C_{\mathrm{sp}}\lambda \sum_{\substack{\tau\in\mathcal{T}_\mathcal{I}\\ \mathcal{R}_\tau=\alpha,\operatorname{sons}(\tau)=\emptyset}} \#\hat{\tau}.
\end{aligned}$$

Due to Assumption 2, $\tau \in \mathcal{T}_\mathcal{I}$ can only be a leaf cluster if $\operatorname{level}(\tau) \geq \ell$ holds, and for these levels $\mathcal{R}_\tau = \alpha$ implies $\tau \in \mathcal{T}_\mathcal{I}^{\tau_\alpha}$, so we get

$$\lambda \sum_{\substack{\tau\in\mathcal{T}_\mathcal{I}\\ \mathcal{R}_\tau=\alpha,\operatorname{sons}(\tau)=\emptyset}} \#\hat{\tau} \leq \lambda \sum_{\substack{\tau\in\mathcal{T}_\mathcal{I}^{\tau_\alpha}\\ \operatorname{sons}(\tau)=\emptyset}} \#\hat{\tau} = \lambda\#\hat{\tau}_\alpha \leq C_{\mathrm{st}}\frac{n\lambda}{p}.$$

Lemma 1 yields

$$\sum_{\substack{\tau\in\mathcal{T}_\mathcal{I}\\ \mathcal{R}_\tau=\alpha}} k^2 \leq C_{\mathrm{st}}k^2\frac{n}{kp} + k^2\log_2 p = C_{\mathrm{st}}\frac{nk}{p} + k^2\log_2 p.$$

We add both estimates to complete the proof. □

Theorem 1 (Distributed storage) *The distributed representation of the $\mathcal{H}^2$-matrix $\widetilde{G}$ requires $\mathcal{O}(n(k+\lambda)/p + k^2\log_2 p)$ units of storage in each processing node.*

Proof Combine Lemmas 2 and 3. □

In typical situations, we have $\lambda \sim k$ and conclude that the distributed representation of an $\mathcal{H}^2$-matrix requires $\mathcal{O}(nk/p + k^2\log_2 p)$ units of storage per node. For sufficiently large problems, e.g., for $n \geq kp\log_2 p$, we reach the optimal order nk/p of complexity.

4.3 Time complexity

In order to analyze the performance of our method, it is not sufficient to count the operations per processing node: if a node has to wait for data from another node, the *runtime* of the algorithm may no longer be proportional to the number of operations.

We therefore have to use a more sophisticated approach. The basis of our analysis is similar to the scheme used in the BSP model [17]: the computation is split into a sequence of $s \in \mathbb{N}$ *supersteps* that are performed independently on all processing nodes. The $(i+1)$th superstep starts on all nodes simultaneously as soon as all nodes have completed the previous ith superstep. We assume that all data sent in the ith superstep is available for corresponding receive operations in the $(i+1)$th superstep.

Since all operations during one superstep are performed independently on all nodes, the time t_i required for the ith superstep is the maximum of the times $t_{i,\alpha}$ required by the nodes $\alpha \in \mathcal{N}$ to complete their part of the superstep. The time required for the entire parallel algorithm is the sum of the times t_i required for all supersteps, i.e.,

$$t_{\text{total}} = \sum_{i=1}^{s} t_i = \sum_{i=1}^{s} \max\{t_{i,\alpha} \ : \ \alpha \in \mathcal{N}\}.$$

We measure the time in *cycles* and assume that one cycle is enough time to perform one arithmetic operation or send or receive one floating point number, including memory access, index computations etc.

Lemma 4 (Construction) *The construction of the $\mathcal{H}^2$-matrix representation, i.e., of the cluster bases $(V_\tau)_{\tau\in\mathcal{T}_\mathcal{I}}$ and $(W_\sigma)_{\sigma\in\mathcal{T}_\mathcal{I}}$, the matrices $(S_{\tau\sigma})_{\tau\times\sigma\in P_{\text{far}}}$ and $(G|_{\hat\tau\times\hat\sigma})_{\tau\times\sigma\in P_{\text{near}}}$ requires $\mathcal{O}((q+1)nk/p + (q+1)k^2\log_2 p)$ cycles.*

Proof The construction of the matrix requires no communication, therefore it can be completed in one superstep. Due to the assumptions of Sect. 2, the computation of one entry of the matrices requires not more than $q+1$ operations, and the number of entries per processing node is bounded due to the Lemmas 2 and 3. □

Lemma 5 (Forward transformation) *The distributed forward transformation requires $\mathcal{O}(nk/p + k^2\log_2 p)$ cycles.*

Proof We consider the algorithm "par_forward". It proceeds from the leaves of the tree $\mathcal{T}_\mathcal{I}$ towards the root to compute the vectors $\hat{x}_\sigma$, therefore it is straightforward to use the levels of $\mathcal{T}_\mathcal{I}$ to define supersteps.

We use a "breakpoint" concept: the first superstep in a node $\alpha \in \mathcal{N}$ contains all operations carried out by this node until the first time it reaches the line marked by (*) with $\text{level}(\sigma) = \ell - 1$. At this point, no receive operations have taken place, the vectors $\hat{x}_\sigma$ have been computed for all $\sigma \in \mathcal{T}_\mathcal{I}$ with $\text{level}(\sigma) \geq \ell$, and the vectors required for level $\ell - 1$ have been sent (the breakpoint is marked by a dashed line in Fig. 5).

The second superstep contains all operations carried out until the node reaches the line marked by (*) with $\text{level}(\sigma) = \ell - 2$. The node receives the vectors $\hat{x}_{\sigma'}$ sent in the previous superstep, computes $\hat{x}_\sigma$, and sends it if necessary.

The following supersteps are defined in a similar fashion: for $i \in \{2, \ldots, \ell\}$, the ith superstep contains the operations carried out from the point when the node reaches the line (*) with $\text{level}(\sigma) = \ell - i + 1$ until it reaches this line with $\text{level}(\sigma) = \ell - i$. The node receives the vectors from level $\ell - i + 1$ and computes and sends the vectors for level $\ell - i$.

The final superstep $i = \ell + 1$ takes care of the remaining operations in the root cluster with $\text{level}(\sigma) = 0$.

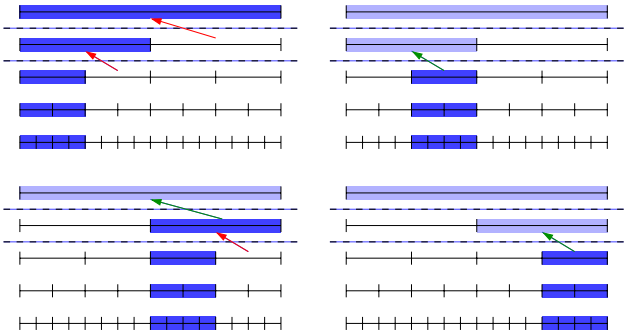

Fig. 5 Distributed forward transformation for $p = 4$ nodes. Send operations are shown as green arrows, receive operations by red arrows. The dashed lines indicate the boundaries of the supersteps

In the first superstep, node α computes the vectors $\hat{x}_\sigma$ for all $\sigma \in \mathcal{T}_\mathcal{I}$ with $\text{level}(\sigma) \geq \ell$ and $\mathcal{R}_\sigma = \alpha$. According to Lemma 1, this means that the vectors $\hat{x}_\sigma$ are computed for all $\sigma \in \mathcal{T}_\mathcal{I}^{\tau_\alpha}$.

If $\sigma \in \mathcal{T}_\mathcal{I}^{\tau_\alpha}$ is a leaf, the computation of $\hat{x}_\sigma$ requires $2k(\#\hat\sigma)$ operations for the multiplication with $W_\sigma^\top$, and the node α has to perform a total of

$$\sum_{\sigma\in\mathcal{T}_\mathcal{I}^{\tau_\alpha}} 2k(\#\hat\sigma) = 2k \sum_{\sigma\in\mathcal{T}_\mathcal{I}^{\tau_\alpha}} \#\hat\sigma = 2k\#\hat\tau_\alpha \leq 2C_{\text{st}}\frac{nk}{p} \quad \text{operations}$$

for all such leaf clusters.

If $\sigma \in \mathcal{T}_\mathcal{I}^{\tau_\alpha}$ is not a leaf, the computation of $\hat{x}_\sigma$ requires $2k^2$ operations for the multiplication by the transfer matrix $E_{\sigma'}^\top$ for each son $\sigma' \in \text{sons}(\sigma)$, therefore the node α requires a total of

$$\sum_{\sigma\in\mathcal{T}_\mathcal{I}^{\tau_\alpha}} \sum_{\sigma'\in\text{sons}(\sigma)} 2k^2 \leq \sum_{\sigma'\in\mathcal{T}_\mathcal{I}^{\tau_\alpha}} 2k^2 = 2k^2\#\mathcal{T}_\mathcal{I}^{\tau_\alpha} \leq 2k^2 C_{\text{st}}\frac{n}{kp} = 2C_{\text{st}}\frac{nk}{p} \quad \text{operations}$$

for all non-leaf clusters.

At the end of the first superstep, node α may send the vector $\hat{x}_\sigma$ to another node, which takes k operations.

We have assumed that one cycle provides enough time for one arithmetic or communication operation, so we get a bound of

$$t_1 := 4C_{\text{st}}\frac{nk}{p} + k$$

for the maximal number of cycles required in the first superstep.

Now we consider the other supersteps $i \in \{2, \ldots, \ell + 1\}$. These supersteps involve only clusters $\sigma \in \mathcal{T}_\mathcal{I}$ with $\text{level}(\sigma) \leq \ell$, therefore Assumption 2 implies that these clusters have exactly two sons. If α is responsible for one of these clusters σ, Algorithm "setup_responsibility" ensures

that only one of the sons is the responsibility of another node, therefore exactly one receive operation for a vector $\hat{x}_{\sigma'}$ with $\mathcal{R}_{\sigma'} \neq \alpha$ has to be performed. This takes k operations.

Then the vector $\hat{x}_\sigma$ is computed by multiplying by the transfer matrices $E_{\sigma'}^\top$ for all $\sigma' \in \operatorname{sons}(\sigma)$, and since there are exactly two sons, this requires $4k^2$ operations.

If we are not in the last superstep, the result may have to be sent to a different node, which takes k operations, leading to an upper bound of

$$t_i := \begin{cases} 4k^2 + 2k & \text{if } i < \ell + 1, \\ 4k^2 + k & \text{otherwise, i.e., if } i = \ell + 1 \end{cases}$$

for the maximal number of cycles required in the ith superstep.

The total run time is bounded by the sum over all supersteps, i.e., by

$$\begin{aligned} \sum_{i=1}^{\ell+1} t_i &= 4C_{\mathrm{st}} \frac{nk}{p} + k + 4k^2\ell + 2k(\ell - 1) + k \\ &= 4C_{\mathrm{st}} \frac{nk}{p} + 4k^2\ell + 2k\ell \leq 4C_{\mathrm{st}} \frac{nk}{p} + 6k^2\ell \quad \text{cycles.} \end{aligned}$$

Due to $\ell = \log_2 p$, this is the estimate we had to prove. □

Lemma 6 (Backward transformation) *The parallel backward transformation requires $\mathcal{O}(nk/p + k^2 \log_2 p)$ cycles.*

Proof As in Lemma 5. Since the backward transformation works from the root clusters towards the leaves, we only have to reverse the order of the supersteps. □

Lemma 7 (Broadcast) *The exchange of the vectors $\hat{x}_\sigma$ by the algorithm "broadcast" requires $\mathcal{O}(n/p + k \log_2 p)$ cycles.*

Proof We base the proof on bounds for the cardinalities of the sets $T_{\alpha\beta}$. Let $\alpha, \beta, \gamma \in \mathcal{N}$ with $\beta \neq \gamma$, and let $\sigma \in \mathcal{T}_{\alpha\beta}$. By definition, this means that we can find a cluster $\tau \in \mathcal{T}_\mathcal{I}$ with $\tau \times \sigma \in P_{\mathcal{I}\times\mathcal{I}}$, $\mathcal{R}_\tau = \alpha$ and $\mathcal{R}_\sigma = \beta \neq \gamma$, therefore we conclude $\sigma \notin \mathcal{T}_{\alpha\gamma}$ and

$$\mathcal{T}_{\alpha\beta} \cap \mathcal{T}_{\alpha\gamma} = \emptyset \quad \text{for all } \alpha, \beta, \gamma \in \mathcal{N}, \beta \neq \gamma.$$

By a similar argument, we prove

$$\mathcal{T}_{\alpha\beta} \cap \mathcal{T}_{\gamma\beta} = \emptyset \quad \text{for all } \alpha, \beta\gamma \in \mathcal{N}, \alpha \neq \gamma.$$

Let now $\alpha, \beta \in \mathcal{N}$. By definition, we have

$$\begin{aligned} \mathcal{T}_{\alpha\beta} &= \{\sigma \in \mathcal{T}_\mathcal{I} : \text{ there exists } \tau \in \mathcal{T}_\mathcal{I} \text{ with} \\ &\qquad \sigma \in \operatorname{row}(\tau), \mathcal{R}_\tau = \alpha, \mathcal{R}_\sigma = \beta\} \\ &\subseteq \bigcup_{\substack{\tau \in \mathcal{T}_\mathcal{I} \\ \mathcal{R}_\tau = \alpha}} \operatorname{row}(\tau). \end{aligned}$$

Since the sets $\mathcal{T}_{\alpha\beta}$ for different β are disjoint, we can use Assumption 3 to conclude

$$\sum_{\beta \in \mathcal{N}} \#\mathcal{T}_{\alpha\beta} \leq C_{\mathrm{sp}} \#\{\tau \in \mathcal{T}_\mathcal{I} : \mathcal{R}_\tau = \alpha\} \quad \text{for all } \alpha \in \mathcal{N}.$$

The same approach can be used to prove

$$\sum_{\beta \in \mathcal{N}} \#\mathcal{T}_{\beta\alpha} \leq C_{\mathrm{sp}} \#\{\tau \in \mathcal{T}_\mathcal{I} : \mathcal{R}_\tau = \alpha\} \quad \text{for all } \alpha \in \mathcal{N}.$$

We apply Lemma 1 to conclude

$$\sum_{\beta \in \mathcal{N}} \#\mathcal{T}_{\alpha\beta} \leq C_{\mathrm{sp}} \left(C_{\mathrm{st}} \frac{n}{kp} + \log_2 p \right) \quad \text{for all } \alpha \in \mathcal{N},$$

$$\sum_{\beta \in \mathcal{N}} \#\mathcal{T}_{\beta\alpha} \leq C_{\mathrm{sp}} \left(C_{\mathrm{st}} \frac{n}{kp} + \log_2 p \right) \quad \text{for all } \alpha \in \mathcal{N}.$$

These estimates provide us with the necessary bound for the amount of information that has to be transferred between processing nodes: a node $\alpha \in \mathcal{N}$ has to send not more than

$$\begin{aligned} \sum_{\beta \in \mathcal{N}} \sum_{\sigma \in \mathcal{T}_{\alpha\beta}} k &\leq C_{\mathrm{sp}} \left(C_{\mathrm{st}} \frac{n}{kp} + \log_2 p \right) k \\ &= C_{\mathrm{sp}} \left(C_{\mathrm{st}} \frac{n}{p} + k \log_2 p \right) \end{aligned}$$

units of data to other nodes, and it has to receive not more than

$$\sum_{\beta \in \mathcal{N}} \sum_{\sigma \in \mathcal{T}_{\beta\alpha}} k \leq C_{\mathrm{sp}} \left(C_{\mathrm{st}} \frac{n}{p} + k \log_2 p \right)$$

units of data from other nodes.

We split the algorithm "broadcast" into two supersteps: the first superstep contains all send operations, and we have just seen that it requires $\mathcal{O}(n/p + k \log_2 p)$ cycles. The second superstep contains all receive operations, and it also requires $\mathcal{O}(n/p + k \log_2 p)$ cycles. Combining both yields the estimate we require. □

Theorem 2 (Distributed evaluation) *The distributed computation of $y := \widetilde{G}x$ by the algorithms can be accomplished in $\mathcal{O}(nk/p + k^2 \log_2 p)$ cycles.*

Proof We have to investigate the substeps "par_forward", "broadcast", "par_multiply" and "par_backward". The Lemmas 5, 6 and 7 already imply that three of the four substeps have the necessary complexity, we only have to discuss the multiplication step of the algorithm "par_multiply". For a node $\alpha \in \mathcal{N}$, this algorithm performs not more than two arithmetic operations per element of the matrices $S_{\tau\sigma}$ and $G|_{\hat{\tau}\times\hat{\sigma}}$. Therefore, Lemma 3 implies that not more than $\mathcal{O}(nk/p + k^2 \log_2 p)$ operations are required, and since the algorithm requires no communication, this is also a bound for the number of cycles. □

5 Experiments

Since the model used in Sect. 4.3 is an abstraction of a real parallel computer, we have to verify whether its predictions,

in particular the complexity estimates of Lemma 4 and Theorem 2, coincide with practical experiments.

We perform our experiments on a parallel computer consisting of a number of PCs connected by a Gigabit Ethernet. Our model problem is the approximation of the classical single layer potential operator

$$\mathcal{G}[u](x) = -\frac{1}{2\pi} \int_\Gamma \log \|x - y\| u(y)\, dy$$

on the unit circle Γ. We approximate Γ by a regular polygon with $n \in \mathbb{N}$ edges and discretize $\mathcal{G}$ using Galerkin's method with basis functions that are piecewise constant on each of these edges. The cluster tree is constructed as in [2, Chap. 2] with a cardinality-balanced splitting strategy.

This strategy ensures that each cluster has either two or no sons, therefore the cluster tree satisfies our assumptions if n is not too small. The partition $P_{\mathcal{I}\times\mathcal{I}}$ is constructed using two-dimensional bounding boxes (this is the natural choice since Γ in embedded in $\mathbb{R}^2$), and the $\mathcal{H}^2$-matrix approximation is derived by tensor-product interpolation of the kernel function on the bounding boxes [4].

We use an interpolation order of $m = 7$, which corresponds to $k = m^2 = 49$ Lagrange polynomials. The leaves $\tau \in \mathcal{T}_\mathcal{I}$ of the cluster tree satisfy $\#\hat{\tau} \leq \lambda$ for $\lambda = 32$.

In a first experiment, we construct the $\mathcal{H}^2$-matrix approximation of the Galerkin matrix using the distributed algorithms "par_basis" and "par_matrix" for $p \in \{1, 2, 4, 8, 16\}$ processing nodes and $n \in \{2^{14}, \ldots, 2^{19}\}$ degrees of freedom. For each p, we measure the total runtime $t_{\mathrm{mat},p}$ and compute the speedup factor $t_{\mathrm{mat},1}/t_{\mathrm{mat},p}$ compared to the performance with only one processing node. The result can be found in Fig. 6: as the total number n of degrees of freedom increases, the speedup factor approaches the optimal value of p, as predicted by Lemma 4. Surprisingly, this value is even exceeded for $n = 2^{19}$, a fact that may be due to cache effects (the amount of cache increases with the number of processors, therefore larger parts of the matrix can be accessed more rapidly as p increases).

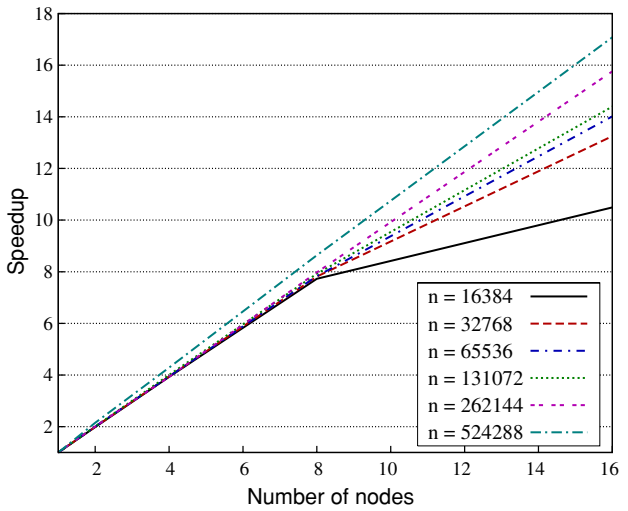

Fig. 6 Speedup factors for the construction of the $\mathcal{H}^2$-matrix

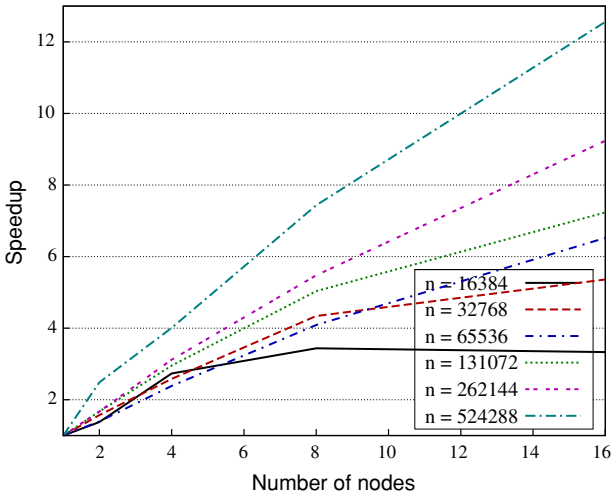

Fig. 7 Speedup factors for the matrix-vector multiplication

In a second experiment, we perform matrix-vector multiplications on the distributed $\mathcal{H}^2$-matrix constructed during the first experiment. The results are given in Fig. 7: as in the previous experiment, the speedup factor increases as n grows larger. We do not see the nearly perfect speedup encountered in the matrix construction algorithm, but we can see that the speedup improves significantly as n increases, as predicted by Theorem 2. For larger values of n, we expect to come arbitrarily close to the perfect speedup factor.

For the smallest problem size $n = 2^{14}$, the speedup factor even seems to drop as p increases. This suggests that the computation steps, i.e., the forward and backward transformation and the multiplication, are far less time-consuming than the communication steps: for a fixed n, the term $k^2 \log_2 p$ appearing in Theorem 2 can become dominant if p grows large enough.

We conclude that our theoretical predictions match the practical experiments: as n grows larger, the speedup factor approaches the optimal value of p, therefore our algorithm scales almost perfectly.

References

1. Beylkin, G., Coifman, R., Rokhlin, V.: The fast wavelet transform and numerical algorithms. Commun. Pure Appl. Math. **44**, 141–183 (1991)

2. Börm, S., Grasedyck, L., Hackbusch, W.: Hierarchical Matrices. Lecture Notes of the Max Planck Institute for Mathematics in the Sciences, vol. 21 (2003)
3. Börm, S., Hackbusch, W.: Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. Comput. **69**, 1–35 (2002)
4. Börm, S., Hackbusch, W.: $\mathcal{H}^2$-matrix approximation of integral operators by interpolation. Appl.Numer. Math. **43**, 129–143 (2002)
5. Börm, S., Krzebek, N., Sauter, S.A.: May the singular integrals in BEM be replaced by zero? Comput. Methods Appl. Mech. Eng. **194**(2–5), 383–393 (2005)
6. Dahmen, W., Harbrecht, H., Schneider, R.: Compression techniques for boundary integral equations—A symptotically optimal complexity estimates. SIAM J. Numer. Anal. **43**(6), 2251–2271 (2006)
7. Dongarra, J., Walker, D., et al.: MPI: A Message-Passing Interface Standard. Available at http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html
8. Giebermann, K.: Multilevel approximation of boundary integral operators. Comput. **67**, 183–207 (2001)
9. Grasedyck, L., Hackbusch, W.: Construction and arithmetics of $\mathcal{H}$-matrices. Comput. **70**, 295–334 (2003)
10. Greengard, L., Rokhlin, V.: A new version of the fast multipole method for the Laplace in three dimensions. In: Acta Numerica 1997, pp. 229–269. Cambridge University Press, Cambridge (1997)
11. Hackbusch, W., Khoromskij, B., Sauter, S.A.: On $\mathcal{H}^2$-matrices. In: Bungartz, H., Hoppe, R., Zenger, C. (eds.) Lectures on Applied Mathematics, pp. 9–29. Springer, Berlin (2000)
12. Hackbusch, W., Nowak, Z.P.: On the fast matrix multiplication in the boundary element method by panel clustering. Numer. Math. **54**, 463–491 (1989)
13. von Petersdorff, T., Schwab, C. : Fully discretized multiscale Galerkin BEM. In: Dahmen, W., Kurdila, A., Oswald, P. (eds.) Multiscale wavelet methods for PDEs, pp. 287–346. Academic Press, San Diego (1997)
14. Rokhlin, V.: Rapid solution of integral equations of classical potential theory. J. Comput. Phys. **60**, 187–207 (1985)
15. Sauter, S.A.: Variable order panel clustering. Comput. **64**, 223–261 (2000)
16. Sauter, S.A., Schwab, C.: Randelementmethoden. Teubner (2004)
17. Valiant, L.: A bridging model for parallel computation. Communications of the ACM **33**(8), 103–111 (1990). http://doi.acm.org/10.1145/79173.79181