A SCALABLE PARALLEL FACTORIZATION OF FINITE ELEMENT MATRICES WITH DISTRIBUTED SCHUR COMPLEMENTS

D. MAURER AND C. WIENERS*

Abstract. We consider the parallel factorization of sparse finite element matrices on distributed memory machines. Our method is based on a nested dissection approach combined with a cyclic re-distribution of the interface Schur complements. We present a detailed definition of the parallel method and the well-posedness and the complexity of the algorithm is analyzed. A lean and transparent functional interface to existing finite element software is defined, and the performance is demonstrated for several representative examples.

1. Introduction. The computation of accurate and reliable finite element approximations require for many applications several million or even billion unknowns. Realizing this task in reasonable execution time, parallel algorithms are required. For this purpose parallel direct solvers are extremely favorable since they are more robust and less sensitive to problem parameters than iterative solvers. E.g., preconditioned CG-methods are restricted to symmetric positive definite systems, and also multigrid or domain decomposition preconditioners require an efficient coarse problem solver. A direct solver often performs well in cases where many iterative solvers fail, e.g., if the problem is indefinite, non-symmetric, or strongly ill-conditioned. Moreover, combining a direct solver on a reasonably fine coarse problem improves the multigrid performance considerably. Finally, a direct solver can be used very efficiently for problem classes where the same factorization can be applied to many right-hand sides. Examples are linear implicit solver for time-dependent problems and iterative eigenvalue solver.

Our concept for the parallel factorization of sparse matrices is designed for finite element problems, where all unknowns are linked to geometric objects. This allows to link the index distribution of the unknowns to a domain decomposition. The parallel finite element assembling process provides an additively decomposed matrix on overlapping distributed indices. The parallel index distribution defines a process set containing all process numbers where this index is represented. The introduction of these process sets allows for a transparent parallel data structure and for a precise formulation of parallel algorithms including all communication within the distributed memory model.

The first step of the factorization is equivalent to the preprocessing step for non-overlapping domain decomposition methods [35], where then the resulting Schur complement is solved by a suitable preconditioned iteration. Here, an exact parallel factorization for this Schur complement problem is introduced. Using a standard nested dissection method this is very efficient for 2D problems, but in 3D, the resulting Schur complement problems emerge into large dense problems, and they have to be distributed and factorized also in parallel. Extending the basic approach presented in [28] by an efficient parallel Schur complement factorization is the main algorithmic contribution of this work.

Block factorizations and its analysis have been discussed by various authors, see e.g. [20, 15, 26, 16]. A general purpose algorithm for sparse matrices is realized, e.g., in MUMPS [1] for distributed memory, and, e.g., in PARDISO [32, 33, 34] on shared memory machines. Parallel solvers also can be used in hybrid methods for solving

* Institute for Applied and Numerical Mathematics, Karlsruhe Institute of Technology, 76128 Karlsruhe, Germany. Email: daniel.maurer@kit.edu, christian.wieners@kit.edu

subproblems, e.g., in SPIKE [31]. The parallel sparse solver PaStiX [24] is included in PETSc [4] which provides a selection of sparse linear solvers, preconditioners and Krylov methods.

Sequential solvers for sparse matrices such as SuperLU [14] and UMFPACK [13] can be used to eliminate local degrees of freedom. A more general discussion of block factorizations and the treatment of linear systems on parallel machines can be found in [12, 17, 19]. A block LU decomposition method with iterative Schur complement solver is presented in [10].

A different approach is provided by hierarchical $\mathcal{H}$-matrices introduced by Hackbusch [23], where the matrix is approximated by low-rank matrices. Approximate $\mathcal{H}$-matrix factorizations also based on nested dissection are considered in [21, 22].

The paper is organized as follows. We start with the introduction of an abstract model for linear algebra computations on distributed indices in Section 2. Based on a suitable index numbering with respect to the nested dissection, we define a corresponding block factorization. For the application to finite elements the link between the overlapping distributed indices and the non-overlapping domain decomposition is specified in Section 3, and the well-posedness of the block factorization is established for a broad class of finite element discretizations. Then, in Section 4 the parallel realization of the factorization is discussed. For a regular finite element mesh, the complexity of the factorization is estimated in Section 5. Finally, a functional interface for the factorization is defined, and in Section 7 the parallel scalability of the method is demonstrated by representative examples of finite element problems in two and three dimensions.

2. A parallel programming model for distributed matrices. In our computations, the solution vector in $\mathbb{R}^N$ is the coefficient vector of a finite element basis, and the parallel distribution of the coefficient vector corresponds to a non-overlapping decomposition of the computational domain $\Omega \subset \mathbb{R}^D$ $(D = 2, 3)$ into subdomains. Every index of the coefficient vector corresponds to a basis function with small support in Ω, and this support intersects only a small number of subdomains. This yields to a parallel data structure using a consistent representation of the coefficient vector on an overlapping index set with parallel copies of indices at interfaces of the domain decomposition. The overlap of the index set is defined by the overlap of the support of the finite element basis functions.

The finite element stiffness matrix is assembled independently on all subdomains. In parallel this is represented additively, so that communication is required for the access to the full matrix entries.

Following [36, 37], we now introduce the abstract programming model, which can be used completely independent of the finite element application. Then we discuss the connection to the recursive decomposition of the computational domain which yields additional a priori information on the parallel distribution; this recursive structure is essential for the overall efficiency of the factorization.

2.1. Parallel index sets. Our programming model describes linear operations in $\mathbb{R}^N$ on P processes. For the definition of the parallel operations we introduce the set of processes $\mathcal{P} = \{1, \dots, P\}$ and the global set of indices $\mathcal{I} = \{1, \dots, N\}$. In parallel, we use overlapping index sets, i.e., we have an overlapping distribution $\mathcal{I} = \mathcal{I}^1 \cup \dots \cup \mathcal{I}^P$, where $\mathcal{I}^p \subset \mathcal{I}$ denotes the set of indices represented on process p. Thus, every index $i \in \mathcal{I}$ is associated with a process set $\pi(i) = \{p \in \mathcal{P} \colon i \in \mathcal{I}^p\} \subset \mathcal{P}$, where this index is represented.

Let $N^p = |\mathcal{I}^p|$ be the number of indices on process p, and let $x^p \in \mathbb{R}^{N^p}$ be the local coefficient vector. The local coefficient vectors $(x^p)_{p\in\mathcal{P}}$ correspond to a global vector $x \in \mathbb{R}^N$ if parallel consistency $x^p[i] = x^q[i]$ for $p, q \in \pi(i)$ is satisfied.

2.2. Parallel block matrices. We assume that the matrix $A \in \mathbb{R}^{N\times N}$ is represented additively by local contributions $A^p = (A^p[i,j])_{i,j\in\mathcal{I}^p} \in \mathbb{R}^{N^p\times N^p}$; this defines the global matrix entries by $A[i,j] = \sum_{p\in\pi(i)\cap\pi(j)} A^p[i,j]$.

Now, we introduce a further block structure which decomposes A^p. Let $\Pi = 2^{\mathcal{P}}$ be the set of all possible process sets. We define the subset of all active process sets $\Pi_{\mathcal{I}} = \{\pi(i) \in \Pi\colon i \in \mathcal{I}\}$ with respect to the corresponding index decomposition.

A suitable numbering of the active process sets

$$\Pi_{\mathcal{I}} = \{\pi_1, \pi_2, \ldots, \pi_K\} \tag{1}$$

directly induces a non-overlapping decomposition of the index set

$$\mathcal{I} = \mathcal{I}_1 \cup \cdots \cup \mathcal{I}_K \quad \text{with} \quad \mathcal{I}_k = \{i \in \mathcal{I}\colon \pi(i) = \pi_k\}.$$

The corresponding matrix blocks of dimensions $N_k = |\mathcal{I}_k|$ are

$$A^p_{km} = (A^p[i,j])_{i\in\mathcal{I}_k, j\in\mathcal{I}_m} \in \mathbb{R}^{N_k\times N_m}\,, \tag{2}$$

which are a part of the local stiffness matrix A^p on process p, and we define the consistent block matrices

$$A_{km} = \sum_{p\in\pi_k\cap\pi_m} A^p_{km}\,. \tag{3}$$

By construction, the blocks A^p_{km} are zero for $p \notin \pi_k \cap \pi_m$, cf. [28, Lem. 1].

2.3. The block factorization. Based on the numbering (1), we compute a factorization $A = LU$ of the block matrix $A = (A_{km})_{k,m=1,\ldots,K}$ with a regular lower triangular $K \times K$ block matrix $L = (L_{km})_{k,m=1,\ldots,K}$ and normalized upper block matrix $U = (U_{km})_{k,m=1,\ldots,K}$ with $U_{kk} = I_{N_k}$, see Alg. 1.

Algorithm 1 Block factorization. The algorithm ends with $L_{km} := A_{km}$, $k \geq m$ and $U_{km} := A_{km}$, $k < m$.

```
1: for k = 1, ..., K do
2:     A_kk := LU(A_kk)
3:     for m = k+1, ..., K do
4:         A_km := SOLVE(A_kk, A_km)
5:         for n = k+1, ..., K do
6:             A_nm := A_nm − A_nk A_km
```

Within the block algorithm we use the following block matrix operations:

- $A_{kk} := \mathrm{LU}(A_{kk})$
  Here (e.g., using a suitable LAPACK routine [3, 2]), we compute an LU decomposition of A_{kk}, and the result is stored in the matrix entries A_{kk}. Local pivoting may be performed by the subroutine.
- $A_{km} := \textsc{Solve}(A_{kk}, A_{km})$
  Here (using the previously determined LU decomposition) we compute the block matrix $A_{kk}^{-1}A_{km}$. Again, the result is stored in A_{km}.
- $A_{nm} := A_{nm} - A_{nk}A_{km}$
  This operation is realized by one BLAS3 call.

2.4. Recursive numbering. In order to take advantage of the sparsity structure (3), we introduce a suitable numbering of the process sets. This is realized with a nested dissection approach. Therefore, we introduce a hierarchy of $S+1$ decomposition steps, where $2^{S-1} < P \le 2^S$. We start with $\pi_k = \{k\}$ for $k = 1, \dots, P$ and $s = 0$. Moreover, we set $\mathcal{P}^{0,k} = \{k\}$, $\Pi_{\mathcal{I}}^{0,k} = \{\pi_k\}$ and $\Pi_{\mathcal{I}}^{0} = \{\pi_1, \dots, \pi_P\}$. In the case $P < 2^S$ we set $\mathcal{P}^{0,t} = \emptyset$ for $t > P$.

Recursively for $s = 1, \dots, S$, we define process subsets

$$\mathcal{P}^{s,t} = \mathcal{P}^{s-1,2t-1} \cup \mathcal{P}^{s-1,2t}\,, \qquad t = 1, \dots, T_s = 2^{S-s} \tag{4}$$

and

$$\Pi_{\mathcal{I}}^{s,t} = \{\pi \in \Pi_{\mathcal{I}} \setminus \bigcup_{r=0}^{s-1} \Pi_{\mathcal{I}}^{r} \colon \pi \subset \mathcal{P}^{s,t}\}\,, \qquad \Pi_{\mathcal{I}}^{s} = \Pi_{\mathcal{I}}^{s,1} \cup \dots \cup \Pi_{\mathcal{I}}^{s,T_s}\,. \tag{5}$$

This defines a numbering $\Pi_{\mathcal{I}}^{s,t} = \{\pi_k \colon k = K_{s,t-1}+1, \dots, K_{s,t}\}$ with $K_{-1} = 0$, $K_0 = T_0 = P$, $K_{s,0} = K_{s-1}$, and $K_s = K_{s,T_s}$. In the same way we define a numbering $\mathcal{P}^{s,t} = \{p \colon P_{s,t-1} < p \le P_{s,t}\}$ with $P_{s,0} = 0$ and $P_{s,t} = P_{s,t-1} + |\mathcal{P}^{s,t}|$.

Now, with respect to this numbering we observe that some operations in line 3 and line 5 of Alg. 1 can be omitted. Therefore, set

$$\mathcal{K}^{s,t} = \left\{k \colon K_{s-1} < k \le K \text{ and } \pi_k \cap \mathcal{P}^{s,t} \neq \emptyset\right\}\,.$$

For $k \in \Pi_{\mathcal{I}}^{s,t}$ we have $\pi_k \subset \mathcal{P}^{s,t}$ and thus $\pi_k \cap \pi_m = \emptyset$ for all $m \notin \mathcal{K}^{s,t}$, $m > K_{s,t}$; then, the condition (3) yields $A_{km} = 0$. Using this a priori information yields Alg. 2.

Algorithm 2 Block-LU decomposition with block sparsity.

```
1: for s = 0, ..., S do
2:     for t = 1, ..., T_s do
3:         for k = K_{s,t-1} + 1, ..., K_{s,t} do
4:             A_kk := LU(A_kk)
5:             for m ∈ K^{s,t}, m > k do
6:                 A_km := SOLVE(A_kk, A_km)
7:                 for n ∈ K^{s,t}, n > k do
8:                     A_nm := A_nm − A_nk A_km
```

3. Parallel finite element matrices. The parallel factorization only relies on the data model with distributed index sets, but the efficiency depends strongly on small overlaps of the index sets and thus of the sparsity structure (3). Now we show how this is achieved in the finite element context.

Let $\Omega \subset \mathbb{R}^D$ be a domain (with $D = 2$ or 3), and let $V_{\mathcal{I}} = \{\phi_i \colon i \in \mathcal{I}\}$ be a finite element space, where the basis functions ϕ_i have small support $\operatorname{supp}\phi_i \subset \bar{\Omega}$, and where the computational domain is covered by all supports, i.e., $\bigcup_{i\in\mathcal{I}} \operatorname{supp}\phi_i = \bar{\Omega}$.

Let $a_{\mathcal{I}} \colon V_{\mathcal{I}} \times V_{\mathcal{I}} \longrightarrow \mathbb{R}$ be a bilinear form defining a finite element problem. Inserting the basis functions yield the entries of the stiffness matrix $A[i,j] = a_{\mathcal{I}}(\phi_i, \phi_j)$ such that $A[i,j] = 0$ if $\operatorname{supp}\phi_i \cap \operatorname{supp}\phi_j = \emptyset$.

In parallel, we consider a non-overlapping domain decomposition

$$\bar{\Omega} = \bar{\Omega}^1 \cup \dots \cup \bar{\Omega}^P \tag{6}$$

into open subdomains with $\Omega^p \cap \Omega^q = \emptyset$ for $p \neq q$. Restricting the bilinear form to the subdomains yields $a_{\mathcal{I}}(\phi,\psi) = \sum a^p_{\mathcal{I}}(\phi,\psi)$ which allows for the parallel assembling of the local matrices $A^p[i,j] = a^p_{\mathcal{I}}(\phi_i,\phi_j)$ for $i,j \in \mathcal{I}$.

For the definition of the overlapping index sets we consider two cases:

Conforming discretizations In this case, the bilinear form $a_{\mathcal{I}}(\phi,\psi) = a(\phi,\psi)$ is independent of the discretization and we have for $a^p_{\mathcal{I}}(\phi,\psi) = a^p(\phi,\psi)$

$$a^p(\phi,\psi) = 0 \quad \text{if} \quad \operatorname{supp}\phi \cap \operatorname{supp}\psi \cap \Omega^p = \emptyset\,. \tag{7}$$

Then, we set $\mathcal{I}^p = \{i \in \mathcal{I}\colon \operatorname{supp}\phi_i \cap \Omega^p \neq \emptyset\}$.

Non-conforming discretizations For general finite element methods, the bilinear form $a_{\mathcal{I}}(\phi,\psi) = \sum a^p_{\mathcal{I}}(\phi,\psi)$ depends on the discretization. We assume

$$a^p_{\mathcal{I}}(\phi,\psi) = 0 \quad \text{if} \quad \operatorname{supp}\phi \cap \operatorname{supp}\psi \cap \bar\Omega^p = \emptyset\,. \tag{8}$$

Then, we set $\mathcal{I}^p = \{i \in \mathcal{I}\colon \operatorname{supp}\phi_i \cap \bar\Omega^p \neq \emptyset\}$.

In this definition, the overlap of the parallel index sets is larger for non-conforming discretizations; in Sect. 7 we will consider examples for both cases. Note that in some cases a smaller overlap is possible for the parallel matrix representation; here we restrict ourselves to these two cases which cover a broad class of applications.

Next, we derive a geometric characterization of the finite element subspaces corresponding to the index sets. Therefore, we define for $\mathcal{J} \subset \mathcal{I}$ and $\omega \subset \Omega$ the finite element subspaces

$$V_{\mathcal{J}} = \operatorname{span}\{\phi_i\colon i \in \mathcal{J}\} \subset V_{\mathcal{I}}\,, \qquad V_{\mathcal{I}}(\omega) = \{\phi \in V_{\mathcal{I}}\colon \operatorname{supp}\phi \subset \bar\omega\}\,,$$

and for a process set $\pi \subset \mathcal{P}$ we define the index set by $\mathcal{I}_\pi = \{i \in \mathcal{I}\colon \pi(i) = \pi\}$.

LEMMA 3.1. *We have*

$$\mathcal{I}_\pi = \bigcap_{p\in\pi} \mathcal{I}^p \setminus \bigcup_{q\notin\pi} \mathcal{I}^q \qquad \textit{and} \qquad V_{\mathcal{I}_\pi} = V_{\mathcal{I}}(\omega_\pi) \quad \textit{with} \quad \omega_\pi = \bigcap_{p\in\pi} \omega^p \setminus \bigcup_{q\notin\pi} \Omega^q\,,$$

where $\omega^p = \bigcup_{i\in\mathcal{I}^p} \operatorname{supp}\phi_i$.

Proof. We have for $i \in \mathcal{I}_\pi$, $p \in \pi$ and $q \notin \pi$ that $i \in \mathcal{I}^p$ but $i \notin \mathcal{I}^q$. This yields the first assertion. Furthermore, we have $\operatorname{supp}\phi_i \subset \omega^p$ and $\operatorname{supp}\phi_i \cap \Omega^q = \emptyset$. This shows $V_{\mathcal{I}_\pi} \subset V_{\mathcal{I}}(\omega_\pi)$.

Now, we consider $j \in \mathcal{I}$ with $\operatorname{supp}\phi_j \not\subset \omega_\pi$. We have to distinguish two cases. If $p \in \pi$ exists with $\operatorname{supp}\phi_j \not\subset \omega^p$, by construction $j \notin \mathcal{I}^p$. Otherwise, some $q \in \mathcal{P} \setminus \pi$ exists with $\operatorname{supp}\phi_j \cap \Omega^q \neq \emptyset$. Then, by construction $j \notin \mathcal{I}^q$. Together this shows $j \notin \mathcal{I}_\pi$ and thus $\phi_j \notin V_{\mathcal{I}_\pi}$. ∎

3.1. Well-posedness and stability of the factorization. The factorization in Alg. 1 requires in every step k the factorization of the block matrix in line 2. For $k > 1$, this matrix is the Schur complement S_{kk} of the block matrix

$$A_{1:k,1:k} = \big(A_{mn}\big)_{m,n=1,\dots,k} = \begin{pmatrix} L_{1:k-1,1:k-1}U_{1:k-1,1:k-1} & A_{1:k-1,k} \\ A_{k,1:k-1} & A_{kk} \end{pmatrix},$$

where $A_{1:k-1,1:k-1} = L_{1:k-1,1:k-1}U_{1:k-1,1:k-1}$ is the factorization of steps $1,\dots,k-1$, and the Schur complement is given by

$$S_{kk} = A_{kk} - A_{k,1:k-1}U^{-1}_{1:k-1,1:k-1}L^{-1}_{1:k-1,1:k-1}A_{1:k-1,k}\,;$$

note that we have

$$\det(A_{1:k,1:k}) = \det(A_{1:k-1,1:k-1}) \det(S_{kk}) .$$

Thus, the algorithm is well-defined, if all matrices $A_{1:k,1:k}$ for $k = 1, \dots, K$ are regular.

We show that this can be established if the finite element problems restricted to $V_{\mathcal{I}_1 \cup \dots \cup \mathcal{I}_k}$ are well-posed. Therefore, we assume $V_{\mathcal{I}_k} = V_{\mathcal{I}}(\omega_{\pi_k})$ for $k = 1, \dots, K$, which yields

$$V_{\mathcal{I}_1} + \dots + V_{\mathcal{I}_k} = V_{\mathcal{I}}(\Omega_k) \quad \text{with} \quad \Omega_k = \omega_{\pi_1} \cup \dots \cup \omega_{\pi_k} . \tag{9}$$

By construction, this assumption is satisfied for the index set constructions in Sect. 3, cf. Lem. 3.1.

THEOREM 3.2. *Let $\| \cdot \|_{V_{\mathcal{I}}}$ be a norm in $V_{\mathcal{I}}$.*
We assume that for all subdomains $\omega \subset \Omega$ constants $C_\omega \geq c_\omega > 0$ exist satisfying

a) $|a_{\mathcal{I}}(\phi, \psi)| \leq C_\omega \|\phi\|_{V_{\mathcal{I}}} \|\psi\|_{V_{\mathcal{I}}}$ for all $\phi, \psi \in V_{\mathcal{I}}(\omega)$;

b) $\inf\limits_{\phi \in V_{\mathcal{I}}(\omega) \setminus \{0\}} \sup\limits_{\psi \in V_{\mathcal{I}}(\omega) \setminus \{0\}} \dfrac{|a_{\mathcal{I}}(\phi, \psi)|}{\|\phi\|_{V_{\mathcal{I}}} \|\psi\|_{V_{\mathcal{I}}}} \geq c_\omega$;

c) for all $\phi \in V_{\mathcal{I}}(\omega) \setminus \{0\}$ a function $\psi \in V_{\mathcal{I}}(\omega)$ exists such that $a_{\mathcal{I}}(\phi, \psi) \neq 0$.

Then, the block matrices $A_{1:k,1:k}$ are regular with condition number bounded by $\frac{C_{\Omega_k}}{c_{\Omega_k}}$.

Proof. For the matrix $A_{1:k,1:k} = \Big(a_{\mathcal{I}}(\phi_i, \phi_j)\Big)_{i,j=\mathcal{I}_1 \cup \dots \cup \mathcal{I}_k}$ the assumption a) yields the bound $\|A_{1:k,1:k}\|_{V_{\mathcal{I}}} \leq C_{\Omega_k}$ in the associated operator norm, c) guarantees that $A_{1:k,1:k}$ is injective and thus regular, and b) yields the bound $\|A^{-1}_{1:k,1:k}\|_{V_{\mathcal{I}}} \leq c^{-1}_{\Omega_k}$. ∎

REMARK 3.3. *The assumptions a), b), and c) hold, e.g., for inf-sup stable discretizations of saddle point problems. For conforming discretizations of elliptic problems with $a_{\mathcal{I}}(\cdot, \cdot) = a(\phi, \phi)$ the assumptions a), b) are satisfied with $c_\omega = c_\Omega$ and $C_\omega = C_\Omega$ independent of $\omega \subset \Omega$ with respect to the energy norm $\|\phi\|^2_{V_{\mathcal{I}}} = a(\phi, \phi)$ in $V_{\mathcal{I}}$. This yields an uniform estimate for all submatrices.*

4. The parallel factorization. The main observation in Alg. 2 is the inherent parallel structure: all operations for $k \in \mathcal{K}^{s,t}$ can be performed on the process subset $\mathcal{P}^{s,t}$. In step $s = 0$, all blocks A_{kk} on $\pi_k = \{k\}$ for $k = 1, \dots, K_0 = P$ can be inverted simultaneously in parallel. Then, for steps $s = 1, \dots, S$, corresponding to the definition (4) we collect the results on $\mathcal{P}^{s,t}$ from $\mathcal{P}^{s-1,2t-1}$ and $\mathcal{P}^{s-1,2t}$. Therefore, we denote the intermediate results in Alg. 2 in step s on the process set $\mathcal{P}^{s,t}$ by $A^{s,t} = (A^{s,t}_{nk})_{n,k=\mathcal{K}^{s,t}}$. This results into the parallel factorization Alg. 3.

Algorithm 3 Parallel block factorization.

```
1:  for s = 0, ..., S do
2:      for t = 1, ..., T_s do in parallel on process set P^{s,t}
3:          if s > 0 then
4:              for m, n ∈ K^{s,t} do
5:                  A^{s,t}_{mn} = A^{s-1,2t-1}_{mn} + A^{s-1,2t-1}_{mn}
6:          for k = K_{s,t-1}, ..., K_{s,t} do
7:              A^{s,t}_{kk} := LU(A^{s,t}_{kk})
8:              for m ∈ K^{s,t}, m > k do
9:                  A^{s,t}_{km} := SOLVE(A^{s,t}_{kk}, A^{s,t}_{km})
10:                 for n ∈ K^{s,t}, n > k do
11:                     A^{s,t}_{nm} := A^{s,t}_{nm} − A^{s,t}_{nk} A^{s,t}_{km}
```

For $s = 0$, the blocks $A^{0,k} = A_{kk}$ are sparse finite element matrices corresponding to the finite element spaces $V_{\mathcal{I}_k}$, and their factorization is done locally on process k. For $s > 0$, the block matrices $A^{s,t}$ are smaller but dense.

REMARK 4.1. *Alg. 3 can directly be realized in parallel by assigning in step s all computations in $\mathcal{P}^{s,t}$ corresponding to one block column $(A_{nk})_{n=\mathcal{K}^{s,t}}$ to one process, see [28]. In 2D applications with small interfaces $\partial\Omega^p \cap \partial\Omega^q$ in 1D, this approach scales well up to several 100 processes. In particular for large 3D applications with 2D interfaces, a balanced distribution of the columns of $A^{s,t}$ is more efficient.*

For the parallel inversion of $A^{s,t}$ in step $s > 0$, we now introduce a new distribution. For this purpose, the set $\mathcal{K}^{s,t}$ is divided into $\mathcal{K}^{s,t}_{\text{fac}} = \{K_{s,t-1} + 1, \dots, K_{s,t}\}$ and its complement $\mathcal{K}^{s,t}_{\text{Sch}} = \mathcal{K}^{s,t} \setminus \mathcal{K}^{s,t}_{\text{fac}}$. Then, we define the corresponding index sets $\mathcal{I}^{s,t} = \bigcup_{k\in\mathcal{K}^{s,t}} \mathcal{I}_k$, $\mathcal{I}^{s,t}_{\text{fac}} = \bigcup_{k\in\mathcal{K}^{s,t}_{\text{fac}}} \mathcal{I}_k$, $\mathcal{I}^{s,t}_{\text{Sch}} = \bigcup_{k\in\mathcal{K}^{s,t}_{\text{Sch}}} \mathcal{I}_k$, and the block decomposition

$$A^{s,t} = \begin{pmatrix} Z^{s,t} & R^{s,t} \\ L^{s,t} & S^{s,t} \end{pmatrix} \in \mathbb{R}^{(\mathcal{I}^{s,t}_{\text{fac}} \cup \mathcal{I}^{s,t}_{\text{Sch}}) \times (\mathcal{I}^{s,t}_{\text{fac}} \cup \mathcal{I}^{s,t}_{\text{Sch}})}.$$

This matrix is now distributed on the process set $\mathcal{P}^{s,t}$ by introducing new decompositions of the indices

$$\mathcal{I}^{s,t}_{\text{fac}} = \bigcup_{p\in\mathcal{P}^{s,t}} \mathcal{I}^{s,t,p}_{\text{fac}}, \qquad \mathcal{I}^{s,t}_{\text{Sch}} = \bigcup_{p\in\mathcal{P}^{s,t}} \mathcal{I}^{s,t,p}_{\text{Sch}}$$

(such that approximately the same number of indices is assigned to every process) and distributing the columns of the matrices

$$Z^{s,t,p} = \big(Z^{s,t}[i,j]\big)_{i\in\mathcal{I}^{s,t}_{\text{fac}}, j\in\mathcal{I}^{s,t,p}_{\text{fac}}} \qquad L^{s,t,p} = \big(L^{s,t}[i,j]\big)_{i\in\mathcal{I}^{s,t}_{\text{fac}}, j\in\mathcal{I}^{s,t,p}_{\text{Sch}}}$$
$$R^{s,t,p} = \big(R^{s,t}[i,j]\big)_{i\in\mathcal{I}^{s,t}_{\text{Sch}}, j\in\mathcal{I}^{s,t,p}_{\text{fac}}} \qquad S^{s,t,p} = \big(S^{s,t}[i,j]\big)_{i\in\mathcal{I}^{s,t}_{\text{Sch}}, j\in\mathcal{I}^{s,t,p}_{\text{Sch}}}$$

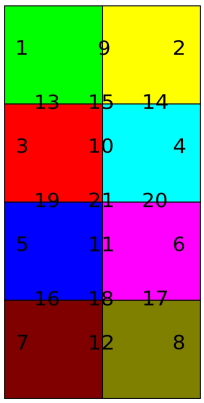
Furthermore, let $Z^{s,t,q,p} = \big(Z^{s,t,p}[i,j]\big)_{i\in\mathcal{I}^{s,t,q}_{\text{fac}}, j\in\mathcal{I}^{s,t,p}_{\text{fac}}}$ the corresponding block matrices of $Z^{s,t}$. Then, the parallel computation of the Schur complement on $\mathcal{P}^{s,t}$ is realized by Algorithm 4.

Algorithm 4 Parallel factorization of $A^{s,t}$ in step s on process set $\mathcal{P}^{s,t}$.

1: **for** $p = P_{s,t-1} + 1, \dots, P_{s,t}$ **do**
2: on p compute $Z^{s,t,p,p} := \text{LU}(Z^{s,t,p,p})$
3: send $Z^{s,t,p}$ to $q \in \mathcal{P}^{s,t}$
4: send $L^{s,t,p}$ to $q \in \mathcal{P}^{s,t}$
5: on $q \in \mathcal{P}^{s,t}$ receive $Z^{s,t,p}$ from p
6: receive $L^{s,t,p}$ from p
7: on $q \in \mathcal{P}^{s,t}$ with $q > p$ set $Z^{s,t,q,p} := \text{SOLVE}(Z^{s,t,p,p}, Z^{s,t,q,p})$
8: compute $Z^{s,t,r,q} := Z^{s,t,r,q} - Z^{s,t,r,p} Z^{s,t,p,q}$ for $r > p$
9: compute $L^{s,t,q} := L^{s,t,q} - L^{s,t,p} Z^{s,t,p,q}$
10: on $q \in \mathcal{P}^{s,t}$ set $R^{s,t,p,q} := \text{SOLVE}(Z^{s,t,p,p}, R^{s,t,p,q})$
11: compute $R^{s,t,r,q} := R^{s,t,r,q} - Z^{s,t,r,p} R^{s,t,p,q}$ for $r > p$
12: compute $S^{s,t,q} := S^{s,t,q} - L^{s,t,p} R^{s,t,p,q}$

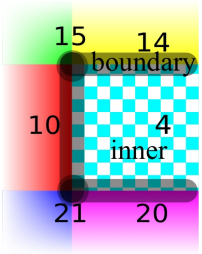
The full parallel algorithm is now obtained by replacing for $s > 0$ the lines 6–11 in Alg. 3 by Alg. 4; for details on the communication structure of the algorithm we refer to [27, Chap. 3].

Example. We illustrate the parallel block factorization for $\Omega = (0,1) \times (0,4)$ distributed to $P = 8$ processes by recursive coordinate bisection.
This results into the process sets



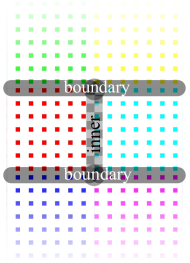
$$\pi_1 = \{1\}, \dots, \pi_8 = \{8\} \text{ in step } s = 0\,,$$
$$\pi_9 = \{1,2\}, \pi_{10} = \{3,4\}, \pi_{11} = \{5,6\}, \pi_{12} = \{7,8\} \text{ in step } s = 1\,,$$
$$\pi_{13} = \{1,3\}, \pi_{14} = \{2,4\}, \pi_{15} = \{1,2,3,4\},$$
$$\pi_{16} = \{5,7\}, \pi_{17} = \{6,8\}, \pi_{18} = \{5,6,7,8\} \text{ in step } s = 2\,,$$
$$\pi_{19} = \{3,5\}, \pi_{20} = \{4,6\}, \pi_{21} = \{3,4,5,6\} \text{ in step } s = 3\,.$$

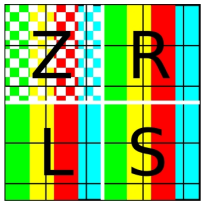
As an example, we consider in more detail the computations on process $p = 4$. In step $s = 0$ we have $\mathcal{K}^{0,4} = \{4, 10, 14, 15, 20, 21\}$ and $\mathcal{K}^{0,4}_{\text{fac}} = \mathcal{P}^{0,4}_{\text{fac}} = \{4\}$. Locally on process $p = 4$, the sparse block matrix A_{44} is decomposed and $A^{0,4}$ is computed by Alg. 3, lines 8–11.



In step $s = 1$ we consider the cluster $t = 2$ in more detail. Here, with $\mathcal{P}^{1,2} = \{3,4\}$ we have $\mathcal{K}^{1,2}_{\text{fac}} = \{10\}$ and $\mathcal{K}^{1,2}_{\text{Sch}} = \{13, 14, 15, 19, 20, 21\}$; the matrix $A^{1,2}$ has the form

$$\left(\begin{array}{c|cccccc} A_{10,10} & A_{10,13} & A_{10,14} & A_{10,15} & A_{10,19} & A_{10,20} & A_{10,21} \\ \hline A_{13,10} & A_{13,13} & 0 & A_{10,15} & A_{13,19} & 0 & A_{13,21} \\ A_{14,10} & 0 & A_{14,14} & A_{14,15} & 0 & A_{14,20} & A_{14,21} \\ A_{15,10} & A_{15,13} & A_{15,14} & A_{15,15} & A_{15,19} & A_{15,20} & A_{15,21} \\ A_{19,10} & A_{19,13} & 0 & A_{19,15} & A_{19,19} & 0 & A_{19,21} \\ A_{20,10} & A_{20,13} & A_{20,14} & A_{20,15} & A_{20,19} & A_{20,20} & A_{20,21} \\ A_{21,10} & 0 & A_{21,14} & A_{21,15} & 0 & A_{21,20} & A_{21,21} \end{array}\right).$$





This procedure is repeated on step $s = 2$. Here, for $t = 1$ we have $\mathcal{P}^{2,1} = \{1,2,3,4\}$ and $\mathcal{K}^{2,1} = \{13, 14, 15, 19, 20, 21\}$; the corresponding distribution of $A^{2,1}$ is illustrated on the left.
Finally, in the last step $s = 3$ with $\mathcal{K}^{3,1} = \{19, 20, 21\}$ we have to decompose $A^{3,1} = (Z^{3,1,1} \cdots Z^{3,1,8})$ in parallel on $\mathcal{P} = \{1, \dots, 8\}$.

5. Complexity analysis. We estimate the complexity of the matrix factorization for the 3D Poisson problem $-\Delta u = f$ in the unit cube $\Omega = (0,1)^3$ with homogeneous Dirichlet boundary conditions. Thus, we have $a(\phi, \psi) = \int_\Omega \nabla\phi \cdot \nabla\psi \, \mathrm{d}x$. We use trilinear finite elements $V_\mathcal{I} \subset \mathrm{H}^1_0(\Omega)$ on a regular hexahedral mesh with mesh width $h_l = 2^{-l}$ on refinement level $l \in \{0, \dots, l_{\max}\}$.

This yields $N = (2^l - 1)^3$ degrees of freedom. The system matrix is irreducible, and a suitable numbering yields a band structure with band width $N^{2/3}$; thus, a sequential factorization requires $O\big(N^{7/3}\big)$ arithmetic operations.

The mesh is distributed to $P = 2^S$ processes by recursive coordinate bisection [38]: starting with $\bar\Omega^{S,1} = \bar\Omega$ a recursive decomposition yields $\bar\Omega^{s,t} = \bar\Omega^{s-1,2t-1} \cup \bar\Omega^{s-1,2t}$ for $t = 1, \dots, 2^{S-s}$ and $s = S, S-1, \dots, 0$. Finally, we obtain a decomposition into P subdomains $\Omega^p = \Omega^{0,p}$. This corresponds to a nested dissection index distribution,

where every index is associated to a nodal point, i.e., the total number of degrees of freedom $N = O(2^{3l})$ is distributed to the hexahedral subdomains Ω^p. This yields $O(N/P)$ interior points and $O\big((2^s N/P)^{2/3}\big)$ boundary points of every subdomain $\Omega^{s,t}$. The index sets $\mathcal{I}^p = \mathcal{I}^{0,p}$ are the indices corresponding to interior nodal points in Ω^p, and for $s \geq 1$ all indices $\mathcal{I}^{s,t}$ are associated to boundary nodes on $\partial\Omega^{s,t}$. For this setting we now estimate the complexity of the parallel factorization in terms of N and P.

LEMMA 5.1. *We assume that* $|\mathcal{I}^p| = O(N/P)$ *for all* $p = 1, \ldots, P = 2^S$, *and that* $|\mathcal{I}^{s,t}| = \big((2^{s-S}N)^{2/3}\big)$ *for* $s \geq 1$. *Then, Algorithm 4 has the parallel arithmetic complexity*

$$C_{\text{comp}} = O\left(\left(\frac{N}{P}\right)^{7/3}\right) + O\left(\frac{N^2}{P}\right),$$

and the complexity of the required communication is

$$C_{\text{comm}} = O\left(N^{4/3} \log P\right).$$

Proof. In step $s = 0$, the matrix A_{pp} is the finite element stiffness matrix of the Poisson problem in $V_{\mathcal{I}_p} = V_{\mathcal{I}} \cap \mathrm{H}^1_0(\Omega^p)$ with band width $(N/P)^{2/3}$, so that the factorization requires $O\big((N/P)^{7/3}\big)$ arithmetic operations. The index set $\mathcal{I}_n$ for $n \in \mathcal{K}^{0,p}$ with $n > p$ corresponds to degrees of freedom associated to the boundaries $\partial\Omega^p$, i.e., $|\mathcal{I}_p| = O\big((N/P)^{2/3}\big)$. This yields an arithmetic expense of complexity $O\big((N/P)^{2/3}(N/P)^{5/3}\big) = O\big((N/P)^{7/3}\big)$ for the solution of the linear systems in line 9 of Alg. 3. The result is a dense matrix with $O(N/P)$ rows and $O\big((N/P)^{2/3}\big)$ columns. This is multiplied in line 11 by a matrix with $O\big((N/P)^{2/3}\big)$ rows and $O(N/P)$ columns, also resulting into $O\big((N/P)^{7/3}\big)$; employing the sparsity of A_{np}, the arithmetic expense of this operation can be further reduced. Since $|\mathcal{K}^{0,p}|$ is limited by the number of faces, edges, and corners of Ω^p, we obtain $|\mathcal{K}^{0,p}| = O(1)$. Together, this results in total arithmetic complexity $O\big((N/P)^{7/3}\big)$ in step $s = 0$.

Now we consider the steps $s > 0$. By assumption we have $|\mathcal{I}^{s,t}| = O\big((2^{s-S}N)^{2/3}\big)$, so that the Schur complement computation of $A^{s,t}$ requires $O\big((2^{s-S}N)^2\big)$ operations which are distributed in Alg. 4 to $|\mathcal{P}^{s,t}| = 2^s$ processes. This results in a total amount of arithmetic operations

$$\sum_{s=1}^{S} O\big((2^{s-S}N)^2\big)/2^s = O(N^2) \sum_{s=1}^{S} 2^{s-2S} = O(N^2/P).$$

Finally, we consider the communication complexity. The first step $s = 0$ requires no communication. Then, for $s \geq 1$, the data of the dense matrix $A^{s,t}$ has to be distributed to $|\mathcal{P}^{s,t}| = 2^s$. Employing again $|\mathcal{I}^{s,t}| = O\big((2^{s-S}N)^{2/3}\big)$ and broadcast communication on a binary tree in $\mathcal{P}^{s,t}$, we obtain a communication amount of

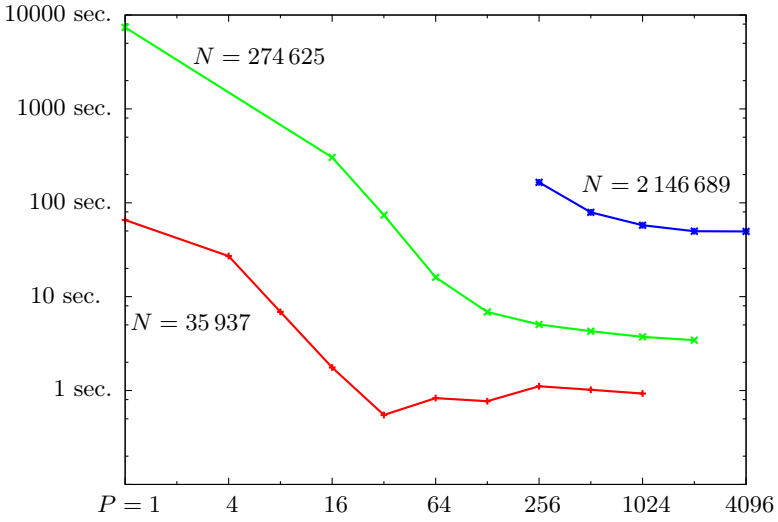
$$\sum_{s=1}^{S} O\big((2^{s-S}N)^{4/3}\big) \log 2^s = O(N^{4/3}) \sum_{s=1}^{S} (2^s/P)^{4/3} s = O(N^{4/3} \log P)$$

matrix entries in all steps $s = 1, \ldots, S$. □

The complexity estimates are clearly confirmed by the computational results of our implementation, see Tab. 5.1. We consider the Poisson problem in the unit cube

TABLE 5.1
Parallel execution time on the CRAY XE6 Cluster in Stuttgart for the factorization for the Poisson problem in the unit cube $(0,1)^3$ *on level* $l = 5, 6, 7$.

	$N = 35\,937$	$N = 274\,625$	$N = 2\,146\,689$
$P = 1$	1:05.58 min.	2:03:25 hrs.	
$P = 4$	27.05 sec.		
$P = 8$	6.89 sec.		
$P = 16$	1.76 sec.	5:06.06 min.	
$P = 32$	0.55 sec.	1:13.94 min.	
$P = 64$	0.83 sec.	16.06 sec.	
$P = 128$	0.77 sec.	6.87 sec.	
$P = 256$	1.11 sec.	5.05 sec.	2:45.30 min.
$P = 512$	1.02 sec.	4.29 sec.	1:19.11 min.
$P = 1024$	0.93 sec.	3.73 sec.	57.73 sec.
$P = 2048$		3.44 sec.	49.75 sec.
$P = 4096$			49.55 sec.



up to refinement level $l = 7$ and $N = 2\,146\,689$ degrees of freedom, and we compute the factorization for 1–4096 processes.

The estimates in Lem. 5.1 show that for small process numbers the computing time $O\left(\left(\frac{N}{P}\right)^{7/3}\right)$ for the first step $s = 0$ is dominant, and if the number of processes is too large, the communication time $O(N^{4/3} \log P)$ exceeds the computing time. If $P \log(P) \leq O(N^{2/3})$, the parallel factorization scales with $1/P$. E.g., for $N \approx 10^6$ we expect by this result that more than 1000 processes cannot be used efficiently.

For a more detailed estimate of the predicted computing and communication time in every step see [27, Chap. 5]. This is evaluated with machine-dependent pre-measured execution times for the arithmetic operations (such as matrix-matrix-multiplication) and the measured time for communication by broadcasting the data in $\mathcal{P}^{s,t}$. For two test cases of Tab. 5.1 the execution and communication time is evaluated for every step in Tab. 5.2. This clearly illustrates that for smaller process numbers the local factorization in the first step $s = 0$ and for larger process numbers

the communication in the last step $s = S$ limits the overall efficiency of the parallel factorization.

TABLE 5.2
Estimated times for computing $t^{\text{est}}_{\text{comp}}$ and communication $t^{\text{est}}_{\text{comm}}$ compared with the measured execution times $t^{\text{ex}}_{\text{comp}}$, $t^{\text{ex}}_{\text{comm}}$ for the 3D model problem with $N = 2\,146\,489$ on $P = 512$ and $P = 2048$ processes.

		$\lvert\mathcal{K}^{s,t}_{\text{fac}}\rvert$	$\lvert\mathcal{K}^{s,t}_{\text{Sch}}\rvert$	$t^{\text{est}}_{\text{comp}}$	$t^{\text{ex}}_{\text{comp}}$	$t^{\text{est}}_{\text{comm}}$	$t^{\text{ex}}_{\text{comm}}$
$P = 512$	$s = 0$	4096	1538	11.9	12.4	—	—
	$s = 1$	256	2562	0.48	0.38	0.25	0.01
	$s = 2$	512	4098	1.24	1.02	0.10	0.19
	$s = 3$	1024	6146	2.94	2.40	0.59	0.80
	$s = 4$	1024	9281	2.96	2.14	1.03	0.37
	$s = 5$	2048	12449	5.84	4.16	1.90	1.16
	$s = 6$	4096	12481	7.63	4.97	2.95	2.79
	$s = 7$	4096	16641	5.94	4.98	4.37	3.34
	$s = 8$	8256	16641	8.62	6.98	8.29	8.45
	$s = 9$	16641	0	3.89	2.14	11.96	11.23
$P = 2048$	$s = 0$	1024	642	0.50	0.42	—	—
	$s = 1$	128	1026	0.04	0.02	0.04	0.00
	$s = 2$	256	1538	0.10	0.06	0.02	0.01
	$s = 3$	256	2562	0.11	0.09	0.09	0.03
	$s = 4$	512	4098	0.30	0.27	0.21	0.07
	$s = 5$	1024	6146	0.71	0.63	0.47	0.19
	$s = 6$	1024	9281	0.73	0.79	0.85	0.34
	$s = 7$	2048	12449	2.41	1.65	2.80	0.92
	$s = 8$	4096	12481	2.96	2.43	3.59	2.50
	$s = 9$	4096	16641	1.48	2.56	4.85	3.23
	$s = 10$	8256	16641	2.15	4.59	9.87	9.15
	$s = 11$	16641	0	0.97	1.95	14.6	10.77

6. Implementation and finite element matrix interface. We realized the factorization of the distributed finite element matrix in the parallel framework M++ written in C++ based on the MPI model for distributed memory machines. We shortly comment on the basic concepts and data structures, and we define the software interface connecting the finite element code with the solution library. For simplicity, we restrict ourselves to the `double` version in 3D of the factorization.

In the first step we define the interface in the form as it is realized in M++; further realizations, e.g. in DUNE [8, 7] or DEAL.II [5], can be implemented easily.

The parallel data structure in M++ [36, 37] follows the concepts in UG [6], see also [8, 7]. A mesh consists of cells c, faces f, edges e and vertices z. In the first step, for the cells c of the mesh on level 0 a distribution is computed; this defines $\pi(c) = \{p_c\}$, where p_c is the process number assigned to the cell c. Recursively, we now determine the process sets for faces f, edges e, and vertices z. For the face f between the cells c and c' we set $\pi(f) = \pi(c) \cup \pi(c')$. Then, combining the process sets of all faces $f \supset e$ we define define $\pi(e)$, and finally $\pi(z)$ is the union of all $\pi(e)$ with $z \in e$.

For the mesh on level 0, the cell distribution and the process sets for faces, edges and vertices can be computed on one process; then, all geometric objects are sent to all processes in π. Now, further refinement of the mesh is done locally, and all geometry based process sets on refined meshes can be determined without further

communication. This includes the process sets for all nodal points $z \in \mathcal{N} \subset \bar{\Omega}$ of the finite element discretization $V_{\mathcal{I}}$. Let $\mathcal{I}_z$ be the indices associated to the nodal point z, so that $\mathcal{I} = \bigcup_{z \in \mathcal{N}} \mathcal{I}_z$ is the index set of the finite element basis. Then, the local index sets $\mathcal{I}^p \subset \mathcal{I}$ and the process sets $\pi(i)$ can be determined from the process sets $\pi(z)$ of the associated nodal points.

For the application of the parallel factorization we assume that a parallel distribution of the nodal points $\mathcal{N} = \mathcal{N}^1 \cup \cdots \cup \mathcal{N}^P$ is given together with a local numbering $\mathcal{N}^p = \{z_j \colon j = 1, \ldots, J^p\}$ and associated process sets $\pi_j = \{p_{jq} \colon q = 1, \ldots, P_j\}$.

We assume that every finite element basis function $\phi_i \in V_{\mathcal{I}}$ is associated to a nodal point z_j. Let n_j be the number of degrees of freedom at the nodal point z_j, so that $N^p = \sum_{j=1}^{J^p} n_j$ is the dimension of the local finite element space on process p. Furthermore, let $\mathcal{I}^{pj} = \{m_j + l \colon l = 1, \ldots, n_j\}$ be the local indices for degrees of freedom at the nodal point z_j (determined by the index shift m_j). In the assembling routine the block matrices corresponding to the degrees of freedom at the nodal points z_j and z_k are accessed by $A^{pjk}[s,t] = A^p[m_j + s][m_k + t]$ for $s = 1, \ldots, n_j$ and $t = 1, \ldots, n_k$.

For the matrix A^p we use the standard compressed row storage format: non-zero entries $a_1, \ldots, a_{M^p}$, row indices $r_0 = 0, r_1, \ldots, r_{N^p}$ and column indices $c_1, \ldots, c_{M^p}$ determine the matrix by $A^p[k, c_l] = a_l$ for $l = r_{k-1} + 1, \ldots, r_k$.

For the parallel factorization a C++ interface is defined by the header `LIB_PS.h` which has to be included in the finite element code:

```
class LIB_PS_Solver;
LIB_PS_Solver* LIB_PS_GET_SOLVER
  (vector<double>*, vector<short>*, int*,
   short*, int, int, int, double*, int*, int*, int,
   int min_matrix_size = 0, int maxP = 0);
void LIB_PS_SOLVE
  (LIB_PS_Solver*, double*, const double*);
void LIB_PS_DESTRUCT
  (LIB_PS_Solver*);
```

Now, provided that for the finite element matrix $A = \sum A^p$ the local matrix A^p on process p is given in the compressed row storage format and that the parallel overlap of the indices is defined by nodal points and associated process sets, the factorization and the solution of the linear system is called by

```
LIB_PS_Solver* PS_Solver =
     LIB_PS_GET_SOLVER (coordinate, procsets, ind, dof, numR,
                        p, P, nzval, rowind, colptr, Size);
LIB_PS_SOLVE(PS_Solver,u,b);
```

Here, we set `coordinate[j-1][d-1]` $= z_j[d]$ for $d = 1, 2, 3$, `procsets[j-1][q-1]` $= p_{jq}$ for $q = 1, \ldots, P_j$, `ind[j-1]` $= m_j$ and `dof[j-1]` $= n_j$ for $j = 1, \ldots,$`numR`$= J^p$, the process number `p` $= p$, the number of processes `P` $= P$, and for the matrix `a[l-1]` $= a_l$ for $l = 1, \ldots, M^p$, `rowind[k]` $= r_k$ for $k = 0, \ldots, N^p$, `colptr[l-1]` $= c_l$ for $l = 1, \ldots, M^p$, and the number of non-zero matrix entries `Size` $= M^p$.

The solver assumes that the process sets π_j are consistent, i.e., $z_j \in \mathcal{N}^q$ for $q \in \pi_j$. Since in our code the information of the parallel distribution of the nodal points is available from the mesh data, the process sets are accessible without further communication. Otherwise, before the factorization can be applied, a global exchange of the nodal points is required which then allows for a consistent construction of the process sets.

The parallel factorization starts with collecting all active process sets $\Pi_{\mathcal{I}}$ to process $p = 1$, determining recursively $\Pi_{\mathcal{I}}^{s,t}$ and a numbering, and then broadcasting this numbering to all processes. Locally on all processes the block matrices A_{km} are constructed. Using a total ordering of the nodal points it is guaranteed that the index numbering in $\mathcal{I}_k$ coincides on all processes $p \in \pi_k$.

For the factorization of the sparse matrix in the first step $s = 0$, any sparse method can be applied; we use the SuperLU package [14] for this task. For all further operations with dense matrices we directly call BLAS level 3 routines.

The parallel communication is realized with MPI. Since the active process set $\Pi_{\mathcal{I}}$ is globally distributed at the beginning, the full communication pattern is known a priori. All steps $s > 0$ start in line 5 of Alg. 3 by collecting the additive contributions $A_{mn}^{s,t}$; this is directly realized by one-to-one communication. Within the Schur complement construction of $A^{s,t}$, all communication in Alg. 4 is realized by broadcast in the process subset $\mathcal{P}^{s,t}$ and by defining corresponding communicators `MPI_Comm`.

7. Numerical experiments. We introduce several finite element applications where the matrix factorization is more challenging than for the 3D model problem (cf. Tab. 5.1). The first examples consider applications in fluid and solid mechanics with conforming discretizations; the wave equation applied to a problem in electrodynamics uses a non-conforming discretization (cf. Section 3.1).

7.1. Stokes equation. We consider a 2D Stokes problem

$$-\Delta \mathbf{u} + \nabla p = 0\,, \qquad \operatorname{div} \mathbf{u} = 0$$

for a backward facing step configuration on the domain $\Omega = (-1,5)\times(-1,1)\setminus[-1,0]^2$, cf. [18, Ex. 4.1.2]. We set $\mathbf{u}(-1,y) = (y^2(1-y^2),0)$ on the inflow boundary, homogeneous Dirichlet boundary conditions for p on the outflow boundary on the right, and homogeneous Dirichlet boundary conditions for $\mathbf{u}$ on the other boundaries. We use Taylor-Hood-Serendipity Q_2/Q_1 elements on a regular quadrilateral mesh. The resulting flow is illustrated in Figure 7.1.

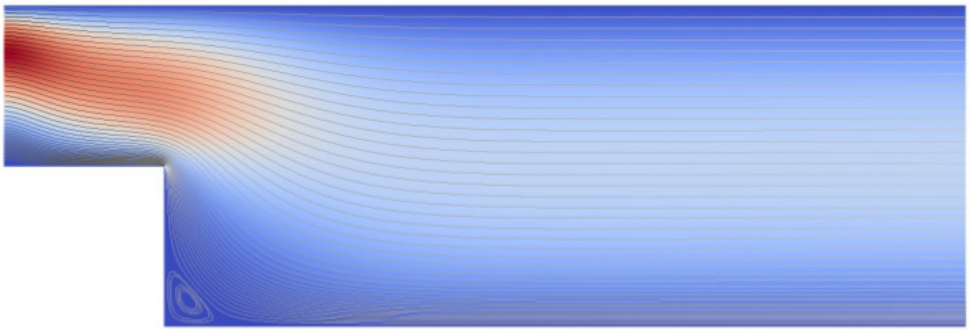


Fig. 7.1. *Streamlines of the flow of the Stokes example.*

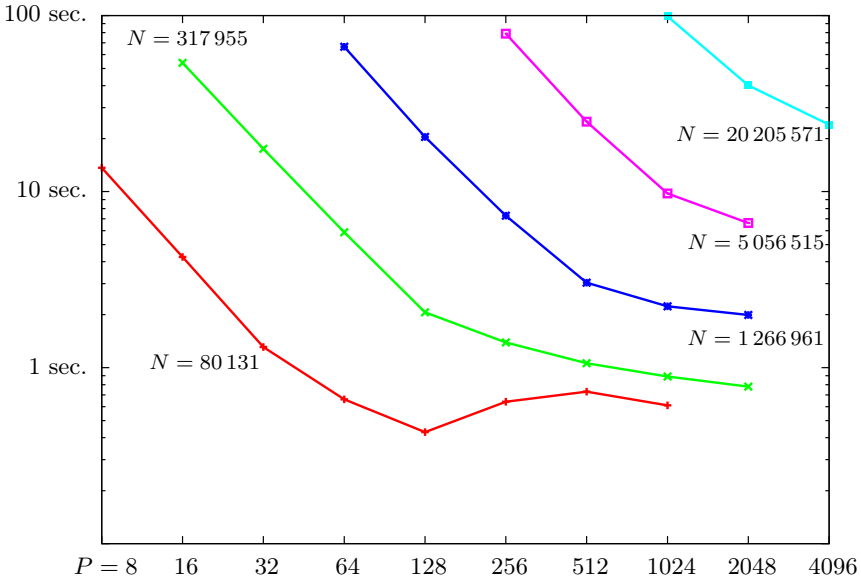
The corresponding bilinear form

$$a((\mathbf{u},p),(\mathbf{v},q)) = \int_\Omega \Big(\nabla\mathbf{u}\cdot\nabla\mathbf{v} - p\nabla\cdot\mathbf{v} - \nabla\cdot\mathbf{u}\,q\Big)\,\mathrm{d}x$$

is uniformly inf-sup stable for Taylor-Hood elements [9, Chap. IV.3], so that the assumptions of Thm. 3.2 are satisfied and the block factorization of the symmetric

TABLE 7.1
Parallel execution time on the CRAY XE6 Cluster in Stuttgart for the factorization for the Stokes problem on level $l = 5, 6, 7, 8, 9$.

	$N = 80\,131$	$317\,955$	$1\,266\,961$	$5\,056\,515$	$20\,205\,571$
$P =$ 8	13.94 sec.				
$P =$ 16	4.24 sec.	54.08 sec.			
$P =$ 32	1.31 sec.	17.98 sec.			
$P =$ 64	0.34 sec.	5.37 sec.	65.00 sec.		
$P =$ 128	0.16 sec.	1.63 sec.	19.69 sec.		
$P =$ 256	0.15 sec.	0.62 sec.	6.72 sec.	73.00 sec.	
$P =$ 512	0.16 sec.	0.49 sec.	2.62 sec.	24.83 sec.	
$P =$ 1024	0.25 sec.	0.72 sec.	1.96 sec.	10.71 sec.	
$P =$ 2048		1.38 sec.	2.15 sec.	8.53 sec.	41.61 sec.
$P =$ 4096					26.97 sec.



indefinite finite element matrix is well-defined. Note that the mixed discretization is conforming in $V_{\mathcal{I}} \subset H^1(\Omega)^2 \times L_2(\Omega)$ so that the conforming parallel index set construction (7) is applied.

Since for configuration in 2D with $O(N)$ unknowns the interfaces are in 1D with $|\mathcal{I}^{s,t}| = O\big((2^{s-S}N)^{1/2}\big)$, the parallel complexity of the 2D factorization is considerably better than in the 3D case, so that the algorithm scales well up to several thousand processes. The results are shown in Table 7.1.

7.2. Finite elasticity. In the next example, we consider the elastic deformation of the heart muscle. This is modeled by a nearly incompressible Neo-Hooke material determined by the free energy of the form

$$W(\mathbf{F}) = \frac{\mu}{2}|\mathbf{F}|^2 + \Sigma(J) \text{ with } \Sigma(J) = \frac{\lambda}{4}J^2 - \left(\frac{\lambda}{2} + \mu\right)\log J - \frac{\lambda}{4} - \frac{3\mu}{2},$$

depending on the deformation gradient $\mathbf{F}$ and $J = \det(\mathbf{F})$, cf. [11]. For given interior forces $\mathbf{f}$ and exterior forces $\mathbf{g}$, the deformation is determined by minimizing the total

energy

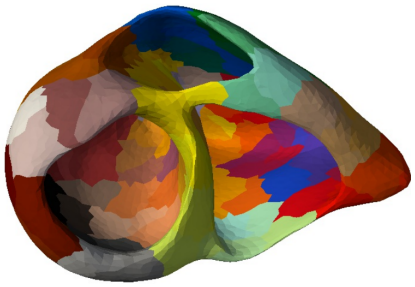
$$\mathcal{E}(\mathbf{u}) = \int_\Omega W(\mathbf{I} + \mathrm{D}\mathbf{u})\mathrm{d}x - \int_\Omega \mathbf{f}\cdot\mathbf{u}\mathrm{d}x - \int_{\partial\Omega\setminus\Gamma_\mathrm{D}} \mathbf{g}\cdot\mathbf{u}\mathrm{d}a$$

subject to the Dirichlet boundary condition $\mathbf{u} = \mathbf{u}_\mathrm{D}$ on Γ_D for the displacement vector. In our example we use $\mathbf{u}_\mathrm{D} = \mathbf{0}$ on a small part of the boundary, a fixed vector $\mathbf{f} \neq \mathbf{0}$ and $\mathbf{g} = \mathbf{0}$.

We use a tetrahedral mesh and conforming P_1 elements. The discrete minimum of the total energy is determined by a Newton iteration where the matrix factorization is applied to the linearization in every step. For the nearly incompressible material we have $\mu \ll \lambda$, so that the condition number of the Jacobi matrix is quite large. In this case, many iterative solution methods deteriorate, whereas the factorization time is independent of the Lamé parameters μ and λ. For parallel factorization results of a single Newton linearization see Tab. 7.2.

TABLE 7.2
Execution time for the parallel factorization of 3D elasticity on unstructured tetrahedral meshes.

	$N = 205\,884$	$N = 1\,424\,628$
$P =$ 16	66.96 sec.	
$P =$ 32	21.55 sec.	
$P =$ 64	6.98 sec.	
$P =$ 128	3.07 sec.	
$P =$ 256	1.96 sec.	84.02 sec.
$P =$ 512	1.75 sec.	33.10 sec.
$P =$ 1024	1.36 sec.	28.43 sec.
$P =$ 2048	1.89 sec.	38.48 sec.



7.3. Electromagnetic waves. For implicit time discretizations of linear time-dependent problems, one factorization can be used in every time step which makes the method in particular attractive if many time steps are computed. Here–as an example for linear hyperbolic PDEs–this is demonstrated for Maxwell's equations in linear materials

$$\varepsilon\frac{\partial \mathbf{E}}{\partial t} - \nabla\times\mathbf{H} = 0 \qquad \mu\frac{\partial \mathbf{H}}{\partial t} + \nabla\times\mathbf{E} = 0$$
$$\nabla\times(\mu\mathbf{H}) = 0 \qquad \nabla\times(\varepsilon\mathbf{E}) = 0\,.$$

Introducing $\mathbf{u} = (\mathbf{H}, \mathbf{E})$, $M\mathbf{u} = (\mu\mathbf{H}, \varepsilon\mathbf{E})$, and $A\mathbf{u} = (\operatorname{curl}\mathbf{E}, -\operatorname{curl}\mathbf{H})$ we obtain the linear evolution equation of the form

$$M\partial_t\mathbf{u} + A\mathbf{u} = 0 \qquad \text{in } [0,T]$$

starting with (divergence-free) vector fields $\mathbf{u}(0) = \mathbf{u}_0$. In order to avoid CFL limitations, we apply the implicit mid-point rule

$$\mathbf{u}^n = \mathbf{u}^{n-1} - \tau(M + \frac{\tau}{2}A)^{-1}A\mathbf{u}^{n-1}$$

with fixed time step $\tau > 0$. This requires the solution of a linear system in every time step but only one factorization of $M + \frac{\tau}{2}A$. We use a discontinuous Galerkin method in space with full upwind flux, so that this matrix is regular, see [25] for details.

Since we have a non-conforming setting where the indices are assigned to the midpoint of each cell and with non-zero matrix entries for index pairs corresponding to neighboring cells, a larger overlap is required than for the conforming case: if one cell with index i on process p has a neighbor cell from another process $q \neq p$, both processes are assigned to the process set $\pi(i)$, i.e., $p, q \in \pi(i)$; this is illustrated for an example in Figure 7.2.

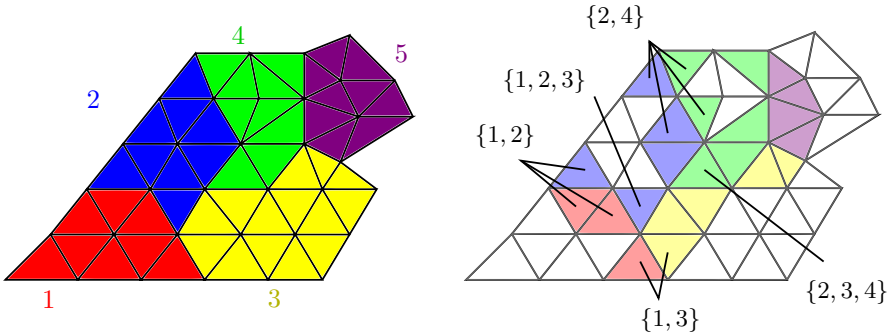


FIG. 7.2. *Example for the non-conforming setting: a domain $\Omega \subset \mathbb{R}^2$ is distributed to 5 processes (left figure). The right figure shows the process sets for indices corresponding to interface elements which provides the required overlap for the parallel factorization of a discontinuous Galerkin discretization.*

Numerically this is tested for the 2D reduction for Maxwell's equation using the configuration [25, Ex. 1]. The geometry distributed on 128 processes is illustrated in Fig. 7.3. The numerical expense for the matrix factorization and the solution time is presented in Table 7.3 for quadratic and cubic ansatz functions. Note that explicit methods require–due to the CFL condition–for this example on the finest level more than 10^5 time steps, see [25, Sect. 6].

TABLE 7.3
Factorization (fac.) and solution (sol.) times in seconds for P2 and P3 discontinuous Galerkin discretization of Maxwell's equations in 2D.

quadratic		fac.	sol.	cubic		fac.	sol.
$N = 108\,864$	$P = 256$	1.17	0.04	$N = 181\,440$	$P = 512$	3.90	0.12
	$P = 512$	1.26	0.04		$P = 1024$	4.19	0.15
	$P = 1024$	1.59	0.06		$P = 2048$	4.09	0.16
$N = 435\,456$	$P = 256$	4.72	0.20	$N = 725\,760$	$P = 512$	13.58	0.60
	$P = 512$	4.02	0.19		$P = 1024$	12.60	0.67
	$P = 1024$	3.97	0.20		$P = 2048$	12.41	0.72
$N = 1\,741\,824$	$P = 256$	40.62	1.25	$N = 2\,903\,040$	$P = 512$	111.62	3.93
	$P = 512$	24.25	1.04		$P = 1024$	80.34	3.86
	$P = 1024$	19.57	1.05		$P = 2048$	78.07	3.94

7.4. Further applications. In this work we focused on the parallel factorization only. Very efficiently, the factorization can also be combined with multigrid methods or domain decomposition methods (in particular if no sufficiently coarse base problem is available, see [30]), and with iterative eigenvalue solver [29], where one decomposition can be applied in many preconditioning steps.

FIG. 7.3. *Distribution of the geometry for the 2D Maxwell's equation on 128 processes.*

REFERENCES

[1] P. AMESTOY, I. DUFF, J. L'EXCELLENT, AND J. KOSTER, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM Journal on Matrix Analysis and Applications, 23 (2001), pp. 15–41.

[2] E. ANDERSON, Z. BAI, C. BISCHOF, J. DEMMEL, J. DONGARRA, J. D. CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, AND D. SORENSEN, *Lapack: A portable linear algebra library for high-performance computers*, 1990.

[3] E. ANDERSON, Z. BAI, C. BISCHOF, J. DEMMEL, J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, S. OSTROUCHOV, AND D. SORENSEN, *LAPACK's user's guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[4] S. BALAY, W. D. GROPP, L. C. MCINNES, AND B. F. SMITH, *Efficient Management of Parallelism in Object Oriented Numerical Software Libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhäuser Press, 1997, pp. 163–202.

[5] W. BANGERTH, R. HARTMANN, AND G. KANSCHAT, *deal.II—a general-purpose object-oriented finite element library*, ACM Transactions on Mathematical Software (TOMS), 33 (2007). Article 24.

[6] P. BASTIAN, K. BIRKEN, K. JOHANNSEN, S. LANG, N. NEUSS, H. RENTZ-REICHERT, AND C. WIENERS, *UG – a flexible software toolbox for solving partial differential equations*, Comput. Vis. Sci., 1 (1997), pp. 27–40.

[7] P. BASTIAN, M. BLATT, A. DEDNER, C. ENGWER, R. KLÖFKORN, R. KORNHUBER, M. OHLBERGER, AND O. SANDER, *A generic grid interface for parallel and adaptive scientific computing. Part II: Implementation and tests in dune*, Computing, 82 (2008), pp. 121–138.

[8] P. BASTIAN, M. BLATT, A. DEDNER, C. ENGWER, R. KLÖFKORN, M. OHLBERGER, AND O. SANDER, *A generic grid interface for parallel and adaptive scientific computing. Part I: Abstract framework*, Computing, 82 (2008), pp. 103–119.

[9] D. BOFFI, F. BREZZI, AND M. FORTIN, *Mixed Finite Element Methods and Applications*, Springer Series in Computational Mathematics, Vol. 44, Springer, New York, 2013.

[10] C. BOMHOF AND H. VAN DER VORST, *A parallel linear system solver for circuit simulation problems.*, Numer. Linear Algebra Appl., 7 (2000), pp. 649–665.

[11] P. CIARLET, *Three-Dimensional Elasticity*, vol. 1 of Studies in Mathematics and its Applications, Elsevier, Amsterdam, first ed., 1988.

[12] K. DACKLAND, E. ELMROTH, B. KÅGSTRÖM, AND C. V. LOAN, *Design and Evaluation of Parallel Block Algorithms: LU Factorization on an IBM 3090 VF/600J*, in Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing, Philadelphia, PA, USA, 1992, Society for Industrial and Applied Mathematics, pp. 3–10.

[13] T. DAVIS, *User's guide for the unsymmetric-pattern multifrontal package (umfpack)*, Tech. Report TR-95-004, Computer and Information Sciences Department, University of Florida, Gainesville, FL, January 1995, (1995).

[14] J. W. DEMMEL, S. C. EISENSTAT, J. R. GILBERT, X. S. LI, AND J. W. LIU, *A supernodal approach to sparse partial pivoting.*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 720–755.

[15] J. W. DEMMEL AND N. J. HIGHAM, *Stability of block algorithms with fast level-3 BLAS.*, ACM Trans. Math. Softw., 18 (1992), pp. 274–291.

[16] J. W. DEMMEL, N. J. HIGHAM, AND R. SCHREIBER, *Stability of block LU factorization*, Numerical Linear Algebra with Applications, 2 (1995), pp. 173–190.

[17] J. J. DONGARRA, I. S. DUFF, D. C. SORENSEN, AND H. V. D. VORST, *Solving Linear*

Systems on Vector and Shared Memory Computers, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1990.

[18] H. Elman, D. Silvester, and A. Wathen, *Finite Elements and Fast Iterative Solvers with applications in incompressible fluid dynamics*, Oxford University Press, 2005.

[19] K. A. Gallivan, R. J. Plemmons, and A. H. Sameh, *Parallel algorithms for dense linear algebra computations*, SIAM Rev., 32 (1990), pp. 54–135.

[20] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Md., 1996.

[21] L. Grasedyck, R. Kriemann, and S. Le Borne, *Parallel black box $\mathcal{H}$-LU preconditioning for elliptic boundary value problems*, Computing and Visualization in Science, 11 (2008), pp. 273–291.

[22] ———, *Domain decomposition based $\mathcal{H}$-LU preconditioning*, Numerische Mathematik, 112 (2009), pp. 565–600.

[23] W. Hackbusch, *Hierarchische Matrizen - Algorithmen und Analysis*, Springer, 2009.

[24] P. Henon, P. Ramet, and J. Roman, *Pastix: a high-performance parallel direct solver for sparse symmetric positive definite systems*, Parallel Computing, 28 (2002), pp. 301–321.

[25] M. Hochbruck, T. Pažur, A. Schulz, E. Thawinan, and C. Wieners, *Efficient time integration for discontinuous Galerkin approximations of linear wave equations*, tech. rep., Karlsruhe Institute of Technology, 2013.

[26] R. Mattheij, *Stability of block LU-decompositions of matrices arising from BVP.*, SIAM J. Algebraic Discrete Methods, 5 (1984), pp. 314–331.

[27] D. Maurer, *Ein hochskalierbarer paralleler direkter Löser für Finite Elemente Diskretisierungen*, PhD thesis, Karlsruhe Institute of Technology, 2013.

[28] D. Maurer and C. Wieners, *A parallel block LU decomposition method for distributed finite element matrices*, Parallel Comput., 37 (2011), pp. 742–758.

[29] ———, *Parallel multigrid methods and coarse grid solver for Maxwell's eigenvalue problem*, in Proc. of the CiHPC 2010: Competence in High Performance Computing, G. Wittum, ed., Springer, 2012, pp. 205–213.

[30] ———, *A highly scalable multigrid method with parallel direct coarse grid solver for Maxwell's equations*, in High Performance Computing in Science and Engineering '13, W. Nagel, D. Kröner, and M. Resch, eds., Springer-Verlag, 2013, pp. 671–677.

[31] E. Polizzi and A. Sameh, *SPIKE: A parallel environment for solving banded linear systems.*, Comput. Fluids, 36 (2007), pp. 113–120.

[32] O. Schenk and K. Gärtner, *Solving unsymmetric sparse systems of linear equations with PARDISO.* Sloot, Peter M. A. (ed.) et al., Computational science - ICCS 2002. 2nd international conference, Amsterdam, the Netherlands, April. 21–24, 2002. Proceedings. Part 2. Berlin: Springer. Lect. Notes Comput. Sci. 2330, 355-363 (2002)., 2002.

[33] O. Schenk, K. Gärtner, W. Fichtner, and A. Stricker, *PARDISO: A high-performance serial and parallel sparse linear solver in semiconductor device simulation.*, FGCS. Future Generation Computer Systems, 18 (2001), pp. 69–78.

[34] O. Schenk, A. Wächter, and M. Hagemann, *Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization*, Computational Optimization and Applications, 36 (2007), pp. 321–341. 10.1007/s10589-006-9003-y.

[35] A. Toselli and O. Widlund, *Domain Decomposition Methods - Algorithms and Theory*, Springer-Verlag, 2005.

[36] C. Wieners, *Distributed point objects. A new concept for parallel finite elements.* Kornhuber, Ralf (ed.) et al., Domain decomposition methods in science and engineering. Selected papers of the 15th international conference on domain decomposition, Berlin, Germany, July 21–25, 2003. Berlin: Springer. Lecture Notes in Computational Science and Engineering 40, 175–182, 2005.

[37] C. Wieners, *A geometric data structure for parallel finite elements and the application to multigrid methods with block smoothing*, Comput. Vis. Sci., 13 (2010), pp. 161–175.

[38] R. D. Williams, *Performance of dynamic load balancing algorithms for unstructured mesh calculations*, Concurrency: Practice and Experience, 3 (1991), pp. 457–481.