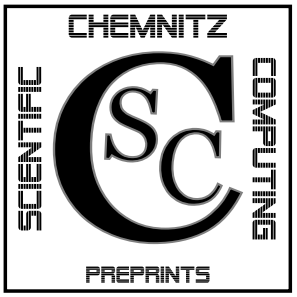
TECHNISCHE UNIVERSITÄT CHEMNITZ

Tino Eibner

A fast and efficient algorithm to compute BPX- and overlapping preconditioner for adaptive 3D-FEM

CSC/08-02



Chemnitz Scientific Computing Preprints

Impressum:

Chemnitz Scientific Computing Preprints — ISSN 1864-0087
(1995–2005: Preprintreihe des Chemnitzer SFB393)

Herausgeber:
Professuren für
Numerische und Angewandte Mathematik
an der Fakultät für Mathematik
der Technischen Universität Chemnitz

Postanschrift:
TU Chemnitz, Fakultät für Mathematik
09107 Chemnitz
Sitz:
Reichenhainer Str. 41, 09126 Chemnitz

`http://www.tu-chemnitz.de/mathematik/csc/`

TECHNISCHE UNIVERSITÄT CHEMNITZ

Chemnitz Scientific Computing Preprints

Tino Eibner

A fast and efficient algorithm to compute BPX- and overlapping preconditioner for adaptive 3D-FEM

CSC/08-02

Abstract

In this paper we consider the well-known BPX-preconditioner in conjunction with adaptive FEM. We present an algorithm which enables us to compute the preconditioner with optimal complexity by a total of only $O(\mathrm{DoF})$ additional memory. Furthermore, we show how to combine the BPX-preconditioner with an overlapping Additive-Schwarz-preconditioner to obtain a preconditioner for finite element spaces with arbitrary polynomial degree distributions. Numerical examples illustrate the efficiency of the algorithms.

CSC/08-02 ISSN 1864-0087 June 2008

1 Introduction

Adaptive FEM is one of the most powerful tool to solve boundary value problems. Whether we consider low order adaptive h-FEM [AO00, BS01, BR03, Ver96] or adaptive hp-FEM [AS97, AS98, AS99, Eib06, HS05, MW01], in both cases we obtain approximate solutions of high accuracy for a minimum of degrees of freedom. However, in order to solve the arrising system of linear equations efficiently we have to apply good preconditioners. In the last twenty years a lot of preconditioners leading to optimal condition number $O(1)$ or almost optimal condition numbers have been developed, e.g. see [BPX91, EM07, HKK05, KL04, Nep86, TW05].

In this paper we will not develop a new preconditioner. Instead we consider the well-known BPX-preconditioner [BPX90, Xu90, Xu92, Zha92] and present a fast and efficent algorithm to compute this preconditioner in the case of adaptive FEM. Moreover, since the applicability of the BPX-preconditioner is restricted to finite element spaces based on piecewise linear functions, we consider the combination of BPX- with an overlapping Additive-Schwarz-preconditioner (see [Osw94, EM07, Eib06, SMPZ08]) to overcome this restriction and obtain a preconditioner for adaptive FEM with arbitrary polynomial degree distribution.

For simplicity of exposition, we will restrict our attention to scalar reaction-diffusion equations in 3D and adaptive h-FEM on tetrahedral meshes without hanging nodes.

The key of our implementation is a properly chosen data-structure which enables us to compute the BPX-preconditioner with optimal complexity and a total of only $O(\mathrm{DoF})$ additional memory. To our knowledge, this is a new approach to compute the BPX-preconditioner for an adaptive FEM scenario and we want to emphasise that our implementation can easily be adapted to adaptive hp-FEM and other types of equations such as, for example, elasticity equations. Furthermore, our algorithm does not depend on the dimension of the domain, that is it can be applied to adaptive 2D-FEM on triangular meshes without changes. We also expect that our algorithm can be extented to hexahedral meshes as well as to meshes containing hanging nodes.

2 Model problem and FE-discretization

Let $\Omega \subset \mathbb{R}^3$ be a Lipschitz domain and assume that its boundary $\Gamma = \partial\Omega$ consists of two disjoint portions,

$$\Gamma = \Gamma_D \cup \Gamma_N,$$

where Γ_D and Γ_N consist of a finite number of disjoint parts of positive measure, or they are empty, respectively. Let $A = [a_{ij}]_{i,j=1}^3$ be a coefficient matrix and

a_0 a scalar valued function. Denote by f_r the right hand side function and by g_N the boundary conditions on Γ_N. Then the classical formulation of our model problem reads as follows:

Problem 2.1 (model problem - classical formulation)**.** *Find a solution u of*

$$\begin{aligned} -\nabla \cdot (A\nabla u) + a_0 u &= f_r \qquad \text{in } \Omega, \\ u &= 0 \qquad \text{on } \Gamma_D, \\ \langle A\nabla u, \eta \rangle &= g_N \qquad \text{on } \Gamma_N. \end{aligned}$$

Since the more appropriated formulation of this problem is the so called weak formulation, we write

$$H^1_D(\Omega) := \{u \in H^1(\Omega) \,|\, u|_{\Gamma_D} = 0\}$$

and consider henceforward the Galerkin formulation of Problem 2.1.

Problem 2.2 (model problem - weak formulation)**.** *Find $u \in H^1_D(\Omega)$ such that*

$$\int_\Omega \nabla u \cdot A\nabla v + a_0 uv d\Omega = \int_\Omega f_r v d\Omega + \int_{\Gamma_N} g_N v d\Gamma \qquad \forall\, v \in H^1_D(\Omega).$$

Throughout the paper we demand $A = A(x)$ to be uniformly symmetric positive definite for $x \in \overline{\Omega}$. We demand $a_{ij}, a_0 \in L^\infty(\Omega)$ with $a_0 \geq \alpha_0 > 0$ in the case of $\Gamma_D = \emptyset$ and $a_0 \geq 0$ otherwise. Finally we claim $g_N \in L^2(\Gamma_N)$ and $f_r \in L^2(\Omega)$.

Due to these assumptions the bilinear form $a : H^1_D(\Omega) \times H^1_D(\Omega) \mapsto \mathbb{R}$, given by

$$a(u, v) := \int_\Omega \nabla u \cdot \mathbf{A}\nabla v + a_0 uv \mathrm{d}\Omega,$$

is symmetric, continuous and coercive and the functional $f : H^1_D(\Omega) \mapsto \mathbb{R}$ with

$$f(v) := \int_\Omega f_r v \mathrm{d}\Omega + \int_{\Gamma_N} g_N v \mathrm{d}\Gamma$$

belongs to the dual space $[H^1_D(\Omega)]^*$. Thus the assumptions of Lax-Milgram [Bra97, Sch98] are satisfied which gives us the existence and uniqeness of a solution for Problem 2.2.

However, in order to solve Problem 2.2 numerically, we have to discretize it. To that end we cover the domain Ω with an admissible shape-regular FE-mesh $\mathcal{T}$ consisting of tetrahedra (see, e.g. [Cia76]) and choose a polynomial degree p. On the reference tetrahedron $\hat{K}$ we denote with $\mathcal{P}_p(\hat{K})$ the space of polynomials of

total degree less or equal p and with $F_K : \widehat{K} \to K$ we denote the affine linear element map of $K \in \mathcal{T}$ to $\hat{K}$. Then we define finite element spaces

$$\begin{aligned} S^p(\Omega, \mathcal{T}) &:= \{u \in H^1(\Omega) \mid u|_K \circ F_K \in \mathcal{P}_p(\hat{K}) \quad \forall K \in \mathcal{T}\}, \\ S^p_D(\Omega, \mathcal{T}) &:= S^p(\Omega, \mathcal{T}) \cap H^1_D(\Omega) \end{aligned}$$

and replace the infinite dimensional space $H^1_D(\Omega)$ by $\mathcal{V} := S^p_D(\Omega, \mathcal{T})$ in Problem (2.2). This gives the finite dimensional problem

Problem 2.3 (Discretized problem)**.** *Find $u \in \mathcal{V}$ such that*

$$a(u, v) = f(v) \qquad \forall\, v \in \mathcal{V}.$$

Now, if we define an Operator $\mathcal{A} : \mathcal{V} \mapsto \mathcal{V}^*$ via

$$[\mathcal{A}u](\cdot) := a(u, \cdot)$$

Problem 2.3 becomes equivalent to a finite dimensional operator equation in $\mathcal{V}^*$.

Problem 2.4 (Operator equation)**.** *Find $u \in \mathcal{V}$ such that*

$$\mathcal{A}u = f. \tag{1}$$

3 PCG-algorithm

Since the operator $\mathcal{A}$ with $[\mathcal{A}u](v) := a(u, v)$ is related to a symmetric positive definite bilinear form $a(\cdot, \cdot)$ we can solve Problem (2.4) by the well-known PCG-algorithm. At this point one usually introduces a basis for the FE-space $\mathcal{V}$ (see Remark (3.2) below). Thus, $\mathcal{V}$ and the dual space $\mathcal{V}^*$ equiped with the corresponding dual basis become equivalent to $\mathbb{R}^N$ and the PCG-algorithm can be considered in $\mathbb{R}^N$ using matrix-vector operations. However, for this paper it is more convenient to postpone the definition of a basis and to consider the PCG-algorithm in a slightly more abstract way, namely as an algorithm which operates in the FE-space $\mathcal{V}$ and its dual space $\mathcal{V}^*$. Before we formulate the algorithm, we define for given $u \in \mathcal{V}$ a corresponding residuum functional $r_u \in \mathcal{V}^*$ via

$$r_u := \mathcal{A}u - f.$$

and omit the index u whenever it is clear from the context.

Algorithm 3.1 (PCG)**.**

- *Choose an initial guess $u \in \mathcal{V}$ and compute its residuum functional $r \in \mathcal{V}^*$.*
- *Apply a preconditioner $\mathcal{C}^{-1} : \mathcal{V}^* \mapsto \mathcal{V}$:*

$$q = w = \mathcal{C}^{-1} r \quad \in \mathcal{V}.$$

- *Compute* $\gamma = r(w)$.
- *While not converged do*

1.	v	$=$	$\mathcal{A}q$	$\in \mathcal{V}^*$
2.	α	$=$	$-\gamma/v(q)$	$\in \mathbb{R}$
3.	u	$=$	$u + \alpha q$	$\in \mathcal{V}$
4.	r	$=$	$r + \alpha v$	$\in \mathcal{V}^*$
5.	w	$=$	$\mathcal{C}^{-1} r$	$\in \mathcal{V}$
6.	$\hat{\gamma}$	$=$	$r(w)$	$\in \mathbb{R}$
7.	q	$=$	$w + (\hat{\gamma}/\gamma) q$	$\in \mathcal{V}$
8.	γ	$=$	$\hat{\gamma}$	$\in \mathbb{R}$

An important issue concerning an iterative solver is its convergence rate. In the case of the PCG-Algorithm (3.1) this convergence rate is strongly related to the precondioner $\mathcal{C}^{-1}$. If we define the energy norm $\|\cdot\|_A$ and the condition number κ of $\mathcal{C}^{-1}\mathcal{A}$ via

$$\|w\|_A^2 := a(w,w) \quad \text{and} \quad \kappa := \frac{\lambda_{max}(\mathcal{C}^{-1}\mathcal{A})}{\lambda_{min}(\mathcal{C}^{-1}\mathcal{A})},$$

then it is a well-known fact that with the initial guess u^0 and the exact solution u^* of $\mathcal{A}u = f$ the i-th initerate u^i of the PCG-algorithm (3.1) satisfies

$$\|u^i - u^*\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^i \|u^0 - u^*\|_A.$$

Remark 3.2. *Later we introduce a basis* $\Phi = [\phi_1, \ldots, \phi_N]$ *for our FE-space* $\mathcal{V}$ *and the dual basis* $\Psi = [\psi_1, \ldots, \psi_N]$ *for* $\mathcal{V}^*$. *With such bases each* $u \in \mathcal{V}$ *becomes equivalent to a vector* $\underline{u} \in \mathbb{R}^N$

$$u = \sum_{i=1}^{N} \underline{u}_i \phi_i \in \mathcal{V} \quad \Leftrightarrow \quad \underline{u} = (\underline{u}_1, \ldots, \underline{u}_N)^T \in \mathbb{R}^N$$

and, moreover, each $f \in \mathcal{V}^*$ *becomes equivalent to a vector* $\underline{f} \in \mathbb{R}^N$

$$f = \sum_{i=1}^{N} \underline{f}_i \psi_i \in \mathcal{V}^* \quad \Leftrightarrow \quad \underline{f} = (\underline{f}_1, \ldots, \underline{f}_N)^T = (f(\phi_1), \ldots, f(\phi_N))^T \in \mathbb{R}^N.$$

Hence, since the matrix representation of the operator $\mathcal{A}$ *is a matrix whose k-th column is the vector representations of* $\mathcal{A}\phi_k \in \mathcal{V}^*$ *with respect to* Ψ, *we obtain*

$$\mathcal{A} : \mathcal{V} \mapsto \mathcal{V}^* \quad \Leftrightarrow \quad A = \begin{bmatrix} a(\phi_1, \phi_1) & \ldots & a(\phi_N, \phi_1) \\ \vdots & & \vdots \\ a(\phi_1, \phi_N) & \ldots & a(\phi_N, \phi_N) \end{bmatrix} \in \mathbb{R}^{N,N}.$$

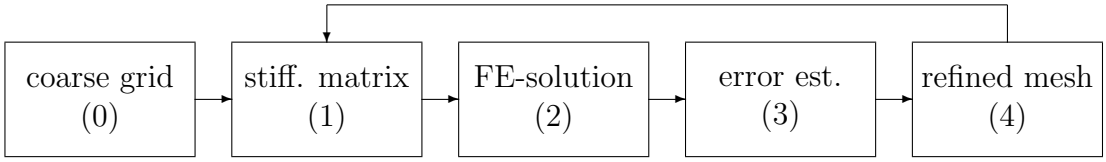
With these matrix and vector representations we obtain the equivalences

$$v = \mathcal{A}q \quad \Leftrightarrow \quad \underline{v} = A\underline{q}, \qquad \gamma = r(w) \quad \Leftrightarrow \quad \gamma = \underline{w}^T \underline{r}, \qquad \ldots$$

and we end up with the well-known $\mathbb{R}^N$*-version of the PCG-algorithmn.*

4 Adaptive FEM

Considering Problem 2.2 it is not common practice to start with a highly refined mesh to compute an approximate solution. Instead we start on a coarse grid with as few nodes as possible. We compute the corresponding FE-solution and, thereafter, in order to improve the accuracy of the approximation, we employ an a-posteriori error estimator which gives us information about the error distribution and flags all elements with large local error. Next we refine all flagged elements and obtain a finer mesh. We compute a new approximation of higher accuracy, estimate the error for this new approximation and in case the approximation is not sufficently accurate a new iteration of the adaptive loop is begun. Thus, we obtain a sequence of refined meshes together with FE-solutions of increasing accuracy. That is we have



In this paper we focus on tetrahedral meshes without hanging nodes. A suited algorithm for local mesh refinement which avoids hanging nodes can be found in [AMP00].

5 Additive-Schwarz-Method(ASM)

As we have already seen in Section 3, the key to solve the arrising system of linear equations with only a few PCG-iterations is the definition of a good preconditioner $\mathcal{C}$. The preconditioners which we will consider are the BPX-preconditioner for the piecewise linear functions of $S^1(\Omega, \mathcal{T})$ and an overlapping preconditioner for the higher polynomial degrees of $S^p(\Omega, \mathcal{T})$. Both preconditioners can be considered as members of the class of Additive-Schwarz-preconditioner which means they are completely determined by a not necessarily direct decomposition

$$\mathcal{V} := S^p_D(\Omega, \mathcal{T}) = \sum_{i=0}^{K} \mathcal{V}_i$$

of our finite element space $\mathcal{V}$ into subspaces $\mathcal{V}_i$. The action of such a preconditioner is best described by the realization of $\mathcal{C}^{-1}r$, where $r \in \mathcal{V}^*$ denotes the residuum.

Definition 5.1 (ASM-preconditioner)**.** *For each subspace $\mathcal{V}_i$ find $w_i \in \mathcal{V}_i$ such that*

$$a(w_i, v) = r(v) \qquad \forall v \in \mathcal{V}_i.$$

Then

$$\mathcal{C}^{-1}r := \sum_{i=0}^{K} w_i.$$

We see, the computation of $\mathcal{C}^{-1}r$ is equivalent to solve local versions of the original problem for all subspaces $\mathcal{V}_i$. However, these subspaces are ordinarily low-dimensional in contrast to the FE-space $\mathcal{V}$.

In order to analyze ASM-preconditioners we define C_0^2 such that

$$\min\left\{\sum_{i=0}^{K} a(u_i,u_i) \;\middle|\; u=\sum_{i=0}^{K} u_i,\quad u_i\in\mathcal{V}_i\right\} \le C_0^2 a(u,u) \quad \forall\, u\in\mathcal{V} \tag{2}$$

and denote with $\rho(E)$ the spectral radius of the matrix $E=[e_{ij}]_{i,j=1}^{K}$ with entries

$$e_{ij} = \sup_{u\in\mathcal{V}_i}\sup_{v\in\mathcal{V}_j} \frac{|a(u,v)|}{\sqrt{a(u,u)}\sqrt{a(v,v)}} \in [0,1], \tag{3}$$

which are the angles between the subspaces $\mathcal{V}_1,...,\mathcal{V}_K$. With these settings we are able to derive a bound for the condition number of $\mathcal{C}^{-1}\mathcal{A}$.

Theorem 5.2.

$$\kappa(\mathcal{C}^{-1}\mathcal{A}) \le C_0^2(1+\rho(E)).$$

Proof. See Section 10. □

6 MDS-BPX

In the following two sections we will introduce the Multilevel-Diagonal-Scaling-BPX (MDS-BPX) as an additive Schwarz preconditioner for $\mathcal{V}=S_D^1(\Omega,\mathcal{T})$ and describe its implementation in detail.

To that end we assume that we are within an adaptive FE-process (see Section 4) which started on a coarse grid $\mathcal{T}_0$ and produced a sequence of nested tetrahedral meshes

$$\mathcal{T}_0 \subset \mathcal{T}_1 \subset \ldots \subset \mathcal{T}_M = \mathcal{T}.$$

We define

$$\begin{aligned} V^m &:= \text{Set of all free vertices on } \mathcal{T}_m \\ &:= \text{Set of all vertices of } \mathcal{T}_m \text{ which are not located on } \Gamma_D, \end{aligned}$$

and we denote with ϕ_v^m the hat function on $\mathcal{T}_m$ with respect to the vertex v.

We define one-dimensional subspaces

$$\mathcal{V}_v^m := \text{span}\phi_v^m.$$

and equip the finite element space $\mathcal{V}$ with the basis $\{\phi_v^M \mid v \in V^M\}$.

Then, in the context of ASM, the MDS-BPX-preconditioner is defined by the decomposition

$$\mathcal{V} = \sum_{m=0}^{M} \sum_{v \in V^m}^{*} \mathcal{V}_v^m, \tag{4}$$

where the "$*$" indicates that the summation is only over pairwise distinct spaces $\mathcal{V}_v^m$. For $\{\mathcal{T}_m\}_{m=0}^M$ arising from a total or a boundary concentraded mesh refinement it is shown that the BPX-preconditioner yields

$$\kappa(\mathcal{C}^{-1}\mathcal{A}) = O(1) \qquad \text{for } M \to \infty, \tag{5}$$

independent on the dimension of Ω (see [Eib06, Xu90, Xu92, Zha92]) and for adaptive mesh refinements all numerical examples indicate that (5) is valid too.

Due to Definition 5.1 of general ASM-preconditioners, the finite element space decomposition (4) and the fact that all subspaces $\mathcal{V}_v^m$ are one-dimensional spaces, we compute for a given residuum $r \in \mathcal{V}^*$ the BPX-preconditioner as

$$\mathcal{C}^{-1}r = \sum_{m=0}^{M} \sum_{v \in V^m}^{*} \left(\frac{r(\phi_v^m)}{a(\phi_v^m, \phi_v^m)} \right) \phi_v^m. \tag{6}$$

To illustrate Splitting (4) we want to close this section with a one-dimensional example.

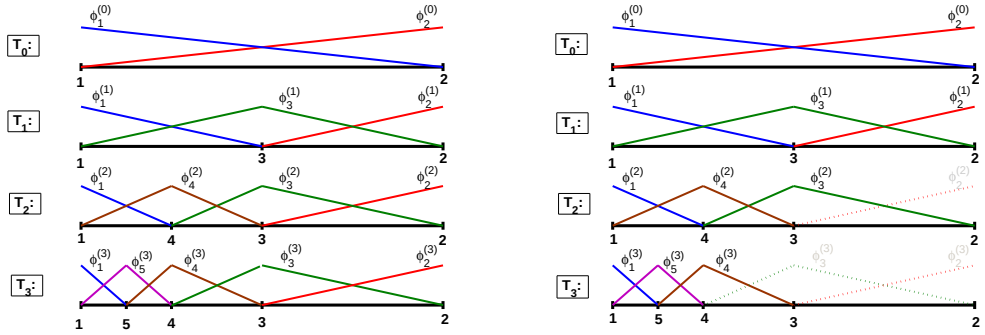
Example 6.1. *For $\Omega = (0,1)$, $\Gamma_D = \emptyset$, given bilinear form $a(\cdot,\cdot)$, and given right hand side functional f we want to solve Problem 2.2. Assume we started with a coarse grid $\mathcal{T}_0$ and that our adaptive FE-algorithm produced a sequence of meshes $\mathcal{T}_0, ..., \mathcal{T}_3$ as in shown in Figure 1 so far.*

According to (4) Example 6.1 induces the splitting

$$\mathcal{V} = S^1(\Omega, \mathcal{T}_3) = (\mathcal{V}_1^0 + \mathcal{V}_2^0) + (\mathcal{V}_1^1 + \mathcal{V}_2^1 + \mathcal{V}_3^1) + (\mathcal{V}_1^2 + \mathcal{V}_3^2 + \mathcal{V}_4^2) + (\mathcal{V}_1^3 + \mathcal{V}_4^3 + \mathcal{V}_5^3).$$

That is, we start with adding up all subspaces of the coarse grid and thereafter, since no subspace of $\mathcal{T}_1$ is equivalent to $\mathcal{V}_1^0$ or $\mathcal{V}_2^0$, we have to add all subspaces of $\mathcal{T}_1$. When we come to the subspaces of $\mathcal{T}_2$ we only add $\mathcal{V}_1^2$, $\mathcal{V}_3^2$, $\mathcal{V}_4^2$ and omit $\mathcal{V}_2^2$ since this subspace is equivalent to $\mathcal{V}_2^1$ and hence already a part of our splitting. Finally we consider the subspaces of $\mathcal{T}_3$ and it remains to add $\mathcal{V}_1^3$, $\mathcal{V}_4^3$, $\mathcal{V}_5^3$.

Figure 1: 1D-example of a sequence of adaptively refined meshes



Remark 6.2. *We number the nodes of $\mathcal{T}_M$ consecutively from 1 to N_M such that if N_m denotes the number of nodes of $\mathcal{T}_m$ the nodes of $\mathcal{T}_m$ correspond to the numbers $1, \dots, N_m$ for $m = 0 \dots M$. Thus, due to this isomorphism between the nodes of $\mathcal{T}_m$ and the numbers $\{1, \dots, N_m\}$ we also write ϕ_i^m instead of ϕ_v^m for a vertex v with number i. Similarly we may write $\mathcal{V}_i^m$ for $\mathcal{V}_v^m$.*

7 Implementation of the MDS-BPX

Now, after we have seen the definition of the BPX-preconditioner, the remaining question is, how to evaluate the double sum (6) fast and efficiently, which means with at most $O(N_M)$ work and at most $O(N_M)$ additional memory, independent on the number of mesh refinements.

The key of our implementation is a properly chosen data-structure the so called extended vertex tree table or short EVT-table, which will be introduced in the following subsection.

7.1 Preliminary work

In order to compute $C^{-1}r$ for a given residuum $r \in \mathcal{V}^*$ we need $r(\phi_v^m)$ and $a(\phi_v^m, \phi_v^m)$ for several but not all basis functions ϕ_v^m.

Since the summation (4) is only over pairwise distinct spaces $\mathcal{V}_v^m$ we observe, that starting with all subspaces $\mathcal{V}_v^0$ for $v \in V^0$ we have to add for $m = 1, \dots, M$ exactly all those subspaces $\mathcal{V}_v^m$ which correspond to a vertex v, newly created on $\mathcal{T}_m$, or one of its parents. That is, for fixed $\mathcal{T}_m$ we need $r(\phi_v^m)$ and $a(\phi_v^m, \phi_v^m)$ for all the basis functions ϕ_v^m, where v is a newly created vertex or a parent of such a vertex.

Remark 7.1. *Newly created vertex with respect to $\mathcal{T}_m$ means that the vertex is created within the refinment process $\mathcal{T}_{m-1} \mapsto \mathcal{T}_m$. In the case $m = 0$ all vertices of $\mathcal{T}_0$ are called newly created.*

Thus, in order to provide all the required $r(\phi_v^m)$ and $a(\phi_v^m, \phi_v^m)$, we generate an EVT-table synchronously to the adaptive finite element process. Our EVT-table contains one row for each node and each of these rows stores two integers and six real numbers. To be more precisely let us consider the node with number i and assume that this node appears first on mesh $\mathcal{T}_m$. Furthermore we assume that its parents have the numbers p_1 and p_2. Then the i-th row of our table provides storage space for

$r(\phi_i^m)$	$a(\phi_i^m, \phi_i^m)$	p_1	$r(\phi_{p_1}^m)$	$a(\phi_{p_1}^m, \phi_{p_1}^m)$	p_2	$r(\phi_{p_2}^m)$	$a(\phi_{p_2}^m, \phi_{p_2}^m)$

Since the nodes of $\mathcal{T}_0$ have no parents the last columns of the first N_0 rows remain empty.

All but the $r(\phi_*^*)$ cells have to be initialized once and thereafter its contents will never change again. Thus we call the set of cells which do not contain an $r(\phi_*^*)$ entry static cells and the prelimiary work is to initialize the static cells of our EVT-table.

The static cells of the first N_0 rows are initialized directly after setting up the stiffness matrix for the initial mesh $\mathcal{T}_0$. For $v \in V^0$ we only have to copy $a(\phi_v^0, \phi_v^0)$ from the diagonal of the stiffness matrix into our EVT-table. For $v \notin V^0$ we set $a(\phi_v^0, \phi_v^0) := 1$. Thereafter the mesh refinement procedure has to record all newly created nodes together with the numbers of its parents and after setting up the stiffness matrix for the refined mesh we run through the rows of our EVT-table associated with the newly created nodes and copy the required entries $a(\cdot, \cdot)$ from the diagonal of the stiffness matrix into our table or set $a(\phi_v^m, \phi_v^m) := 1$ for $v \notin V^m$, respectively.

Considering Example 6.1 this yields to:

Table 1: EVT-table for Example 6.1.

	N			Father1			Father2		
$\mathcal{T}_0$	1	*	$a(\phi_1^0, \phi_1^0)$	*	*	*	*	*	*
	2	*	$a(\phi_2^0, \phi_2^0)$	*	*	*	*	*	*
$\mathcal{T}_1$	3	*	$a(\phi_3^1, \phi_3^1)$	1	*	$a(\phi_1^1, \phi_1^1)$	2	*	$a(\phi_2^1, \phi_2^1)$
$\mathcal{T}_2$	4	*	$a(\phi_4^2, \phi_4^2)$	1	*	$a(\phi_1^2, \phi_1^2)$	3	*	$a(\phi_3^2, \phi_3^2)$
$\mathcal{T}_3$	5	*	$a(\phi_5^3, \phi_5^3)$	1	*	$a(\phi_1^3, \phi_1^3)$	4	*	$a(\phi_4^3, \phi_4^3)$

(Note that only the cells with a white background color are part of our data structure.)

7.2 Computation of $C^{-1}r$ - Step I

To compute $C^{-1}r$ for a given residuum $r \in \mathcal{V}^*$ we assume that the preliminary work is completed, that is we have a fully initialized EVT-table, such as Table 7.1, at hand. Then the first step to compute $C^{-1}r$ is to fill all the $r(\phi_*^*)$ cells of our EVT-table. Since we use the standard nodal basis for $\mathcal{V}$ and its dual basis for $\mathcal{V}^*$ the residuum r is given via a vector $[r(\phi_v^M)]_{v\in V^M}$ (see Remark 3.2). We also need $r(\phi_i^m)$ for $m \neq M$. These values are not given directly but since we know the refinement history, we can compute them recursively from $\underline{r}^M$. At this point and in order to simplify notation we define the extended version of the residual vector

$$\underline{r}^m = \left[\underline{r}_1^m, \ldots, \underline{r}_{N_m}^m\right] \quad \text{with } \underline{r}_i^m := \begin{cases} r(\phi_i^m) & : \text{ if vertex } i \text{ belongs to } V^m \\ 0 & : \text{ otherwise} \end{cases}$$

and we assume that our finite element code provides the extended $\underline{r}^M$ rather than the compact version which omits the entries $r(\phi_v^m)$ for $v \notin V^m$.

Considering Example 6.1 the $\mathcal{T}_3$ section of the corresponding EVT-table 7.1 tells us:

$$\phi_1^2 = \phi_1^3 + \frac{1}{2}\phi_5^3, \qquad \phi_4^2 = \phi_4^3 + \frac{1}{2}\phi_5^3, \quad \text{and} \quad \phi_i^2 = \phi_i^3 \quad \text{for } i \notin \{1,4\}.$$

Thus we have

$$r\left(\phi_1^2\right) = r\left(\phi_1^3\right) + \frac{1}{2}r\left(\phi_5^3\right), \qquad r\left(\phi_4^2\right) = r\left(\phi_4^3\right) + \frac{1}{2}r\left(\phi_5^3\right),$$

and

$$r\left(\phi_i^2\right) = r\left(\phi_i^3\right) \quad \text{for } i \notin \{1,4\}.$$

Generally spoken, in order to obtain $\underline{r}^{m-1}$ from $\underline{r}^m$ we go through the rows $i = N_{m-1}+1, \ldots, N_m$ of our EVT-table and compute for the parents p_1 and p_2 of vertex i

$$\begin{aligned} \underline{r}_{p1} &+= \frac{1}{2}\underline{r}_i \qquad \text{if the vertex with number } p_1 \text{ belongs to } V^{m-1} \text{ and} \\ \underline{r}_{p2} &+= \frac{1}{2}\underline{r}_i \qquad \text{if the vertex with number } p_2 \text{ belongs to } V^{m-1}. \end{aligned} \tag{7}$$

These operations are performed in place, directly on $\underline{r}^m$ without any auxiliary vector and finally the first N_{m-1} positions of the input vector $\underline{r}^m$ contain the entries of $\underline{r}^{m-1}$.

That is, given an initialized EVT-table and the extended residuum vector $\underline{r}^M$, we fill all $r(\phi_i^m)$ cells of our EVT-table by the following algorithm:

Algorithm 7.2.

for $m = M, \dots, 0$
{
  for $i = N_{m-1}+1, \dots, N_m$
    copy $r(\phi_i^m)$, $r(\phi_{p_1}^m)$, $r(\phi_{p_2}^m)$ *from* $\underline{r}^m$ *into the* i*-th row of our EVT-table*
  if $m > 0$
    compute $\underline{r}^{m-1}$ *from* $\underline{r}^m$
}

Considering Example 6.1 once again, we arrive at:

	N			Father1			Father2		
$\mathcal{T}_0$	1	$r(\phi_1^0)$	$a(\phi_1^0,\phi_1^0)$	*	*	*	*	*	*
	2	$r(\phi_2^0)$	$a(\phi_2^0,\phi_2^0)$	*	*	*	*	*	*
$\mathcal{T}_1$	3	$r(\phi_3^1)$	$a(\phi_3^1,\phi_3^1)$	1	$r(\phi_1^1)$	$a(\phi_1^1,\phi_1^1)$	2	$r(\phi_2^1)$	$a(\phi_1^1,\phi_1^1)$
$\mathcal{T}_2$	4	$r(\phi_4^2)$	$a(\phi_4^2,\phi_4^2)$	1	$r(\phi_1^2)$	$a(\phi_1^2,\phi_1^2)$	3	$r(\phi_3^2)$	$a(\phi_1^2,\phi_1^2)$
$\mathcal{T}_3$	5	$r(\phi_5^3)$	$a(\phi_5^3,\phi_5^3)$	1	$r(\phi_1^3)$	$a(\phi_1^3,\phi_1^3)$	4	$r(\phi_4^3)$	$a(\phi_4^3,\phi_4^3)$

7.3 Computation of $C^{-1}r$ - Step II

Now with a complete EVT-table at hand we want to add up all

$$\hat{w}_v^m := \left(\frac{r(\phi_v^m)}{a(\phi_v^m,\phi_v^m)}\right)\phi_v^m$$

of sum (6). To that end we start with $w = 0$. The variable m goes step-by-step from 0 to M and in each step we add all the required contributions $\hat{w}_v^m$ of level m to w. Obviously, the intermediate result

$$w^k := \sum_{m=0}^{k} \sum_{v\in V^m}^{*} \hat{w}_v^m$$

is an element of $S^1(\Omega,\mathcal{T}_k)$ and thus its most appropriate representation is the vector $\underline{w}^k = [\underline{w}_1^k, \dots, \underline{w}_{N_k}^k]$ with

$$w^k = \sum_{i=1}^{N_k} \underline{w}_i^k \phi_i^k.$$

That is, we start with $\underline{w} = [0, 0, ..., 0]$, the vector representation of $w = 0$ with respect to the basis $\{\phi_1^0, \dots, \phi_{N_0}^1\}$ of $S^1(\Omega,\mathcal{T}_0)$ and apply the following algorithm

Algorithm 7.3.

for $m = 0, \ldots, M$

(0) Clear all flags.

(1) Add $\sum_{v \in V^m}{}^{*} \hat{w}_v^m$ *to the current* w *given as* $\underline{w} = [\underline{w}_1^m, \ldots, \underline{w}_{N_m}^m]$ *with*

$$w = \sum_{i=1}^{N_m} \underline{w}_i^m \phi_i^m.$$

(2) If $m < M$

decompose w *after the* $S^1(\Omega, \mathcal{T}_{m+1})$ *basis such that*

$$w = \sum_{i=1}^{N_{m+1}} \underline{w}_i^{m+1} \phi_i^{m+1}$$

and set $\underline{w} = [\underline{w}_1^{m+1}, \ldots, \underline{w}_{N_{m+1}}^{m+1}]$.

Let us consider parts (1) and (2) of Algorithm 7.3 in detail.

In part (1), all we have to do is to go through the rows $i = N_{m-1} + 1, ..., N_m$ of our EVT-table. From each row we read the numbers p_1 and p_2 of the parents of vertex i, compute

$$\begin{array}{ll} \underline{w}_i^m += r(\phi_i^m)/a(\phi_i^m, \phi_i^m) & \text{if vertex } i \text{ belongs to } V^m \ , \\ \underline{w}_{p_1}^m += r(\phi_{p_1}^m)/a(\phi_{p_1}^m, \phi_{p_1}^m) & \text{if vertex } p_1 \text{ is not flagged and belongs to } V^m, \\ \underline{w}_{p_2}^m += r(\phi_{p_2}^m)/a(\phi_{p_2}^m, \phi_{p_2}^m) & \text{if vertex } p_2 \text{ is not flagged and belongs to } V^m, \end{array}$$

and finally flag the vertices p_1 and p_2. All required data can be found in the i-th row of our EVT-table, the flags are necessary to ensure that we add the $r(\phi_{p_k}^m)/a(\phi_{p_k}^m, \phi_{p_k}^m)$ terms only once, see Remark 7.5.

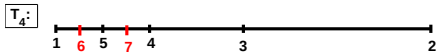
In part (2) we convert the current vector representation of w with respect to the $S^1(\Omega, \mathcal{T}_m)$ basis into a vector representation of w with respect to the $S^1(\Omega, \mathcal{T}_{m+1})$ basis. To do so we have to extent the vector representation $[\underline{w}_1^m, \ldots, \underline{w}_{N_m}^m]$ to $[\underline{w}_1^{m+1}, \ldots, \underline{w}_{N_{m+1}}^{m+1}]$ with

$$\underline{w}_i^{m+1} := \begin{cases} \underline{w}_i^m & \text{for} \quad i \leq N_m \\ \frac{1}{2}(\underline{w}_{p_1}^m + \underline{w}_{p_2}^m) & \text{for} \quad i \in \{N_m + 1, \ldots, N_{m+1}\} \end{cases},$$

where p_1, p_2 are the parents of vertex i. Now we have the desired vector representation of w with respect to the $S^1(\Omega, \mathcal{T}_{m+1})$ basis and once again, all the required data could be obtained directly from the $i = N_m + 1, \ldots, N_{m+1}$ rows of our EVT-table.

Remark 7.4. *The extension of* $\underline{w}$ *to length* N_{m+1} *is mere formality. In general we start with a vector* $\underline{w}$ *of length* N_M *from the very beginning and an extension as above is simply a change of the upper array index.*

Remark 7.5. *Consider Example 6.1 and assume that we obtain a mesh* $\mathcal{T}_4$ *by subdividing the elements* $(1,5)$ *and* $(4,5)$ *of* $\mathcal{T}_3$*. That is we get*



and our EVT-table looks like

	N			*Father1*			*Father2*		
$\vdots$	$\vdots$	$\vdots$		$\vdots$			$\vdots$		
$\mathcal{T}_4$	6	$r(\phi_6^4)$	$a(\phi_6^4,\phi_6^4)$	*1*	$r(\phi_1^4)$	$a(\phi_1^4,\phi_1^4)$	*5*	$r(\phi_5^4)$	$a(\phi_5^4,\phi_5^4)$
	7	$r(\phi_7^4)$	$a(\phi_7^4,\phi_7^4)$	*4*	$r(\phi_4^4)$	$a(\phi_4^4,\phi_4^4)$	*5*	$r(\phi_5^4)$	$a(\phi_5^4,\phi_5^4)$

Now, adding all w_v^4 *contributions of (6) means to perform*

$$w+ = \sum_{i\in I}\left(\frac{r(\phi_i^5)}{a(\phi_i^5,\phi_i^5)}\right)\phi_i^5 \qquad \text{with } I=\{1,4,5,6,7\}.$$

However, if we went through the rows of the $\mathcal{T}_4$ *section of our EVt-table the vertex with number* 5 *appears twice as a parent and consequently we would add*

$$\left(\frac{r(\phi_5^5)}{a(\phi_5^5,\phi_5^5)}\right)\phi_5^5$$

twice. The easiest way to avoid such an undesired doubling is to use a flag for each node.

8 Numerical example

In this section we present some numerical examples to demonstrate the efficiency of our BPX-implementation. All examples start with a coarse grid $\mathcal{T}_0$ of the given domain and then we create a sequence of nested meshes by applying the mesh refinement algorithm of [AMP00]. This algorithm is based on the bisection of tetrahedra and allows us to construct locally refined meshes without hanging nodes.

Example 8.1. *Let* $\Omega=(0,1)^3$, $\Gamma_D=(0,1)\times(0,1)\times\{0,1\}$ *and* $\Gamma_N=\partial\Omega\backslash\Gamma_D$. *Let* $\{\mathcal{T}_i\}_{i=0}^M$ *be a sequence of nested meshes on* Ω*. Find* $u\in\mathcal{V}:=S_D^1(\Omega,\mathcal{T}_M)$ *such that*

$$\int_\Omega \nabla u\cdot\nabla v + uv\,d\Omega = \int_\Omega f_r v\,d\Omega \qquad \forall\, v\in\mathcal{V},$$

where the right hand side is given by $f_r = 1+\sum_{i=1}^3 x_i^2$.

Table 2: Total mesh refinement, $C_{res}^2 = 10^{-6}$

M	Elements	DoF	It	Total[s]	Precond[s]	Ratio
0	6	8	7	6.1e-05	1.8e-05	0.29
1	14	10	8	8.4e-05	2.4e-05	0.28
2	30	17	8	1.2e-04	3.3e-05	0.28
3	70	31	9	2.2e-04	5.4e-05	0.24
4	166	58	11	5.0e-04	1.0e-04	0.20
5	414	120	12	5.1e-04	9.1e-05	0.17
6	990	256	15	1.4e-03	2.1e-04	0.14
7	2.420	554	16	3.5e-03	4.3e-04	0.12
8	5.718	1.251	17	8.8e-03	1.1e-03	0.12
9	13.460	2.768	19	2.6e-02	3.4e-03	0.13
10	30.960	6.172	19	6.1e-02	7.7e-03	0.12
11	70.932	13.617	20	1.5e-01	1.9e-02	0.12
12	158.904	30.809	20	3.6e-01	5.1e-02	0.14
13	355.674	65.614	21	8.5e-01	1.3e-01	0.15
14	782.288	146.532	21	1.9e+00	3.1e-01	0.15
15	1.717.898	317.675	21	4.4e+00	7.0e-01	0.16
16	3.723.834	681.607	22	1.0e+01	1.7e+00	0.16
17	8.067.178	1.466.444	22	2.3e+01	3.8e+00	0.16
18	17.284.898	3.191.462	22	5.4e+01	8.5e+00	0.15
19	37.051.776	6.646.901	22	1.4e+02	2.9e+01	0.21

We consider two different scenarios. At first we consider the case where the mesh $\mathcal{T}_{i+1}$ is derived from $\mathcal{T}_i$ by a total mesh refinement. Thereafter, in a second run, we employ a standard residuum error estimator to refine only those elements with high local error and hence obtain a sequence of locally refined meshes.

All iterations start with an initial guess $u = 0$ and we use the PCG-algorithm in conjunction with our implementation of the BPX-preconditioner to reduce the initial residual (measured in the Euklidean norm) by a given factor C_{res}. That is we iterate until $r_i^2 < C_{res}^2 r_0^2$.

The results of the experiments are collected in Table 2 and Table 3.

The column "Elements" shows the number of tetrahedra and "DoF" denotes the dimension of the finite element space $S_D^1(\Omega, \mathcal{T}_M)$. The column "It" gives the number of PCG iterations required to reduce the residual by C_{res}. "Total" is the computing time of the PCG algorithm including the application of the preconditioner and the column "Precond" contains the time to execute the preconditioner. Finally the column "Ratio" gives the quotient of "Total" and "Precond".

As we can see, the preconditioner leads to small iteration numbers and its eval-

Table 3: Adaptive mesh refinement, $C_{res}^2 = 10^{-8}$

M	Elements	DoF	It	Total[s]	Precond[s]	Ratio
1	6	8	8	1.1e-04	2.2e-05	0.20
2	11	9	7	7.7e-05	2.0e-05	0.26
3	16	11	9	9.2e-05	2.7e-05	0.29
4	23	14	9	1.1e-04	3.2e-05	0.29
5	32	18	10	1.4e-04	4.4e-05	0.31
6	46	24	10	9.1e-05	3.0e-05	0.33
7	108	39	12	1.7e-04	4.5e-05	0.26
8	182	58	13	2.7e-04	6.4e-05	0.23
9	298	97	14	4.4e-04	9.2e-05	0.21
10	508	152	16	8.1e-04	1.5e-04	0.18
15	8.040	1.694	22	1.7e-02	2.3e-03	0.13
20	108.055	21.227	26	3.3e-01	4.4e-02	0.13
25	1.209.941	227.229	28	4.1e+00	6.9e-01	0.16
30	12.934.018	2.382.662	28	5.3e+01	9.5e+00	0.17

uation takes only a fraction of the total computing time of the PCG-algorithm. Most computing time is required for the multiplication of the stiffness matrix A with a vector q, where in our implementation this multiplication is done by multiplications of the stored local stiffness matrices A_K with subvectors q_K of q and assemblation of all the local vectors $v_K = A_K q_K$.

The memory to store our EVT-table is $O(N_M)$, where N_M denotes the number of vertices of the current mesh $\mathcal{T}_M$ and considering our algorithm as well as the results of our numerical experiments we see that the computing time to execute the preconditioner is of order $O(N_M)$ too. However, this computing time is to some extent affected by memory cache effects.

9 Higher polynomial degrees

Up to now we considered finite element spaces $\mathcal{V} = S^1(\Omega, \mathcal{T})$. However, often it is useful to apply $S^p(\Omega, \mathcal{T})$ with $p > 1$ or even a *hp*-version of FEM, where different elements may have different polynomial degrees. Thus the question arrises how to construct a preconditioner for such a setting where higher polynomial degrees appear.

As in the previous sections, we assume that we have a sequence of nested meshes $\mathcal{T}_0 \subset \mathcal{T}_1 \subset \ldots \subset \mathcal{T}_M$ and we define for $v \in V^M$ the patch $\omega_v^M := \cup\{K \in \mathcal{T}_M \mid v \in \overline{K}\}$ which is the closure of the union of all elements $K \in \mathcal{T}_M$ with v as one of its

vertices. Moreover we define for each $v \in V^M$ a subspace

$$S_v = \{u \in S^p(\Omega, \mathcal{T}_M) \mid \operatorname{supp} u \subset \overline{\omega_v^M}\}.$$

Then, in the context of ASM, the splitting

$$S^p(\Omega, \mathcal{T}_M) = S^1(\Omega, \mathcal{T}_M) + \sum_{v \in V^M} S_v$$

defines an overlapping preconditioner with $\kappa(\mathcal{C}^{-1}\mathcal{A}) = O(1)$. If we decompose $S^1(\Omega, \mathcal{T}_M)$ further, such that

$$S^p(\Omega, \mathcal{T}) = \sum_{m=0}^{M} \sum_{v \in V^m}^{*} \mathcal{V}_v^m + \sum_{v \in V^M} S_v,$$

we obtain a preconditioner for $S^p(\Omega, \mathcal{T}_M)$ which can be considered as a combination of BPX- and overlapping preconditioner. Thus, due to Definition (5.1) we get for a given residuum $r \in \mathcal{V}^*$ the preconditioned residuum $C^{-1}r$ by using the BPX-routine from the previous sections to compute

$$\sum_{m=0}^{M} \sum_{v \in V^m}^{*} \left(\frac{r(\phi_v^m)}{a(\phi_v^m, \phi_v^m)} \right) \phi_v^m$$

and adding $w_v \in S_v$ for all $v \in V^M$, given by

$$a(w_v, u) = r(u) \; \forall u \in S_v.$$

That is, after applying the BPX-routine we have to solve a linear system of equations for each subspace S_v. However, these linear systems of equations are low-dimensional and for a fixed subspace S_v, the corresponding system matrix is s.p.d. and always the same. Only right hand sides change.

There are two possible strategies to implement the preconditioner

1. Runtime optimal strategy: We compute a Choleky decomposition once for each system matrix of the S_v-subproblems. We store these Choleky decompositions and reduce solving the linear systems of equations to forward and backward substitutions. Since the polynomial degree p is fixed and the number of tetrahedra sharing a vertex is bounded by a constant, the dimension of S_v is also bounded by a constant and hence we have:

$$\text{additional memory} = O(N_M).$$

2. Memory optimal strategy: We compute the Choleky decompositions anew whenever we have to solve a S_v-subproblem. Since the dimension of S_v is bounded by a constant we have

$$\text{computing time for all Cholesky decompositions} = O(N_M).$$

Table 4: Results for Example 9.1

	p = 1, $C_{res}^2 = 10^{-5}$				p = 3, $C_{res}^2 = 10^{-10}$			
Elements	DoF	Iterations		H1-Err	DoF	Iterations		H1-Err
6	8	2.0e-05	0	5.5e-01	64	7.3e-04	6	4.3e-02
14	10	4.1e-05	1	4.1e-01	105	2.6e-03	15	2.4e-02
30	17	4.0e-05	1	3.3e-01	211	7.2e-03	18	1.3e-02
70	31	8.1e-05	2	2.7e-01	443	2.6e-02	29	1.1e-02
166	58	2.8e-04	5	2.0e-01	950	7.8e-02	40	5.6e-03
414	120	6.4e-04	5	1.5e-01	2.248	2.1e-01	51	3.7e-03
990	256	2.7e-03	8	1.1e-01	5.077	3.0e-01	61	1.9e-03
2.420	554	2.2e-03	9	8.6e-02	11.994	8.1e-01	67	1.1e-03
5.718	1.251	5.2e-03	9	6.6e-02	27.796	2.7e+00	75	6.8e-04
13.460	2.768	1.5e-02	10	5.0e-02	63.978	6.7e+00	81	3.9e-04
30.960	6.172	3.9e-02	11	3.8e-02	145.609	1.5e+01	83	2.4e-04
70.932	13.617	9.3e-02	12	2.9e-02	330.319	3.8e+01	91	1.5e-04
158.904	30.809	2.2e-01	12	2.3e-02	736.750	8.7e+01	92	9.3e-05
355.674	65.614	4.9e-01	12	1.7e-02	1.635.004	2.0e+02	97	6.5e-05
782.288	146.532	1.2e+00	13	1.4e-02	3.594.777	4.7e+02	101	4.8e-05
1.717.898	317.675	2.5e+00	12	1.1e-02	7.864.521	1.2e+03	101	3.5e-05
3.723.834	681.607	6.1e+00	13	8.6e-03	—	—	—	—
8.067.178	1.466.444	1.3e+01	13	6.8e-03	—	—	—	—
17.284.898	3.191.462	3.5e+01	13	5.5e-03	—	—	—	—
37.051.776	6.646.901	8.0e+01	13	4.2e-03	—	—	—	—

Let us consider the following example.

Example 9.1. *Let* $\Omega = (0,1)^3$, $\Gamma_D = \partial\Omega$. *Let* $\{\mathcal{T}_i\}_{i=0}^M$ *be a sequence of nested meshes on* Ω. *For* $p = 1, 2, 3$ *find* $u \in S^p(\Omega, \mathcal{T}_M)$ *with* $u|_{\partial\Omega} = \cos(x)\cos(y)\cos(z)$ *such that*

$$\int_\Omega \nabla u \cdot \nabla v + uv d\Omega = 4 \int_\Omega v d\Omega \qquad \forall\, v \in S_D^p(\Omega, \mathcal{T}_M).$$

We start with $u = 0$ on a coarse grid and apply a total mesh refinement strategy which gives us a sequence of nested meshes. We use the FE-solution of the current mesh as initial guess for the FE-solution on the refined mesh and iterate until the initial residual r_0 is reduced by a given factor C_{res}. That is we iterate until $r_i^2 < C_{res}^2 r_0^2$. In the first run we consider the finite element space $S^p(\Omega, \mathcal{T}_M)$ for $p = 1$ and therafter we compute the runtime optimal strategy for $p = 3$.

The results of the experiments are collected in Table 4.

The column "Elements" shows the number of tetrahedra of the considered mesh and "DoF" denotes the dimension of the corresponding finite element space. The column "Iterations" gives the number of PCG iterations and the computing time

required to reduce the initial residual by C_{res}. Finally the column "H1-Err" shows the error $\|u_{FEM} - u_{exact}\|_{H^1(\Omega)}$ of our finite element solution.

Since the exact solution $u = \cos(x)\cos(y)\cos(z)$ of Example 9.1 is a very smooth function, the advantage of using $S^3(\Omega, \mathcal{T}_M)$ instead of $S^1(\Omega, \mathcal{T}_M)$ is a much smaller error at equal number of degrees of freedom. However, due to the smaller error, we have to increase C_{res}^2 for $p = 3$ which in turn yields to higher iteration numbers and, moreover, solving all S_v-subproblems takes a lot of computing time even if we apply the runtime optimal strategy. On the other hand we have a good approach for a possible parallelization. These S_v-subproblems are independent local problems, thus, if we compute them in parallel the computing time for higher polynomial degrees will decrease significantly.

10 Proof of Theorem 5.2

In this section we want to give a proof for Theorem 5.2. We make use of the notation of Section 5 and define operators $P_i : \mathcal{V} \mapsto \mathcal{V}_i$ and $P : \mathcal{V} \mapsto \mathcal{V}$ via

$$a(P_i u, v) = a(u, v) \quad \forall v \in \mathcal{V}_i \quad \text{and} \quad P := \sum_{i=0}^{K} P_i. \tag{8}$$

Thus $P = \mathcal{C}^{-1}\mathcal{A}$ and we have to derive a bound for

$$\kappa(P) = \frac{\lambda_{max}(P)}{\lambda_{min}(P)}, \tag{9}$$

$$\text{where} \quad \lambda_{min}(P) = \inf_{u \in \mathcal{V},\, u \neq 0} \frac{a(Pu, u)}{a(u, u)} \qquad \text{and} \qquad \lambda_{max}(P) = \sup_{u \in \mathcal{V},\, u \neq 0} \frac{a(Pu, u)}{a(u, u)}.$$

Lemma 10.1 (Lemma of Lions)**.** *Let C_0 be given by (2). Then for all $u \in \mathcal{V}$*

$$\|u\|_A^2 \leq C_0^2 a(Pu, u).$$

Proof. Let $u \in \mathcal{V}$ and choose a representation $u = \sum_{i=0}^{K} u_i$ such that

$$\sum_{i=0}^{K} \|u_i\|_A^2 \leq C_0^2 \|u\|_A^2 \quad \text{with } u_i \in \mathcal{V}_i. \tag{10}$$

Then, applying (8) and Cauchy-Schwarz-inequality we obtain

$$\|u\|_A^2 = \sum_{i=0}^{K} a(u, u_i) = \sum_{i=0}^{K} a(P_i u, u_i) \leq \left(\sum_{i=0}^{K} \|P_i u\|_A^2 \right)^{1/2} \left(\sum_{i=0}^{K} \|u_i\|_A^2 \right)^{1/2}$$

and the claim follows in combination with (10). □

Lemma 10.2. *Let $\rho(E)$ be given by (3). Then for all $u \in \mathcal{V}$*

$$a(Pu, u) \leq (1 + \rho(E))\|u\|_A^2.$$

Proof. We observe that for all $u \in \mathcal{V}$ and $i \in \{0, .., K\}$

$$\|P_i u\|_A^2 = a(P_i u, P_i u) = a(u, P_i u).$$

Hence we have $\|P_i u\|_A^2 \leq \|u\|_A \|P_i u\|_A$ which is

$$\|P_i u\|_A \leq \|u\|_A \quad \forall u \in \mathcal{V}.$$

Next we consider

$$\|(P - P_0)u\|_A^2 = \sum_{i,j=1}^{K} a(P_i u, P_j u) \leq \sum_{i,j=1}^{K} e_{ij} \|P_i u\|_A \|P_j u\|_A.$$

Since $E = [e_{ij}]_{i,j=1}^K$ is a symmetric matrix we obtain

$$\begin{aligned} \|(P - P_0)u\|_A^2 &\leq \rho(E) \sum_{i=1}^{K} \|P_i u\|_A^2 = \rho(E) \sum_{i=1}^{K} a(u, P_i u) \\ &\leq \rho(E)\|u\|_A \|(P - P_0)u\|_A. \end{aligned}$$

Thus, we finally arrive at

$$\begin{aligned} a(Pu, u) &= a(P_0 u, u) + a\left((P - P_0)u, u\right) \\ &\leq a(P_0 u, P_0 u) + \|(P - P_0)u\|_A \|u\|_A \\ &\leq (1 + \rho(E))\|u\|_A^2. \end{aligned}$$

□

Lemma 10.1 and Lemma 10.2 together with (9) give he desired bound for the condition number of P

Theorem 10.3.

$$\kappa(P) = \kappa(\mathcal{C}^{-1}\mathcal{A}) \leq C_0^2(1 + \rho(E)).$$

References

[AMP00] Douglas N. Arnold, Arup Mukherjee, and Luc Pouly. Locally adapted tetrahedral meshes using bisection. *SIAM J. Sci. Comput.*, 22(2):431–448 (electronic), 2000.

[AO00] M. Ainsworth and T.J. Oden. *A posteriori error estimation in finite element analysis*. Wiley, 2000.

[AS97] M. Ainsworth and B. Senior. Aspects of an adaptive *hp*-finite element method: adaptive strategy, conforming approximation and efficient solvers. *Comput. Meth. Appl. Mech. Engrg.*, 150:65–87, 1997.

[AS98] M. Ainsworth and B. Senior. An adaptive refinement strategy for *hp*-finite-element computations. *Appl. Numer. Math.*, 26:165—178, 1998.

[AS99] M. Ainsworth and B. Senior. *hp*-finite element procedures on non-uniform geometric meshes. In M. Bern, J. Flaherty, and M. Luskin, editors, *Grid generation and adaptive algorithms*, pages 1–27. Springer Verlag, 1999. IMA Vol. Math. Appl. 113.

[BPX90] J. H. Bramble, J. E. Pasciak, and J. Xu. Parallel multilevel preconditioners. *Mathematics of Computation*, 55(191):1–22, 1990.

[BPX91] J. Bramble, J. Pasciak, and J. Xu. Parallel multilevel preconditioners. *Math. Comput.*, 55:1–22, 1991.

[BR03] Wolfgang Bangerth and Rolf Rannacher. *Adaptive finite element methods for differential equations*. Lectures in Mathematics ETH Zürich. Birkhäuser Verlag, Basel, 2003.

[Bra97] D. Braess. *Finite Elements*. Cambridge University Press, 1997.

[BS01] I. Babuška and T. Strouboulis. *The finite element method and its reliability*. Oxford University Press, 2001.

[Cia76] P. G. Ciarlet. *The Finite Element Method for Elliptic Problems*. North-Holland Publishing Company, 1976.

[Eib06] T. Eibner. *Randkonzentrierte und adaptive hp-FEM*. PhD thesis, TU Chemnitz, 2006.

[EM07] T. Eibner and J. M. Melenk. Multilevel preconditioning for the boundary concentrated hp-fem. *Comp. Meth. Mech. Eng. 196 (2007), pp. 3713-3725.*, 2007.

[HKK05] W. Hackbusch, B.N. Khoromskij, and R. Kriemann. Direct Schur complement method by domain decomposition based on $\mathcal{H}$-matrix approximation. *Comput. Vis. Sci.*, 8(3-4):179–188, 2005.

[HS05] P. Houston and E. Süli. A note on the design of *hp*-adaptive finite element methods for elliptic partial differential equations. *Comput. Meth. Appl. Mech. Engrg.*, 194:229–243, 2005.

[KL04] V.G. Korneev and U. Langer. Domain decomposition and preconditioning. In E. Stein, R. de Borst, and Th.J.R. Hughes, editors, *Encyclopedia of Computational Mechanics*, volume 1 of *Chapter 22*. John Wiley & Sons, 2004.

[MW01] J.M. Melenk and B. Wohlmuth. On residual-based a posteriori error estimation in *hp*-FEM. *Advances in Comp. Math.*, 15:311–331, 2001.

[Nep86] S.V. Nepomnyaschikh. *Domain Decomposition and Schwarz Methods in a Subspace for the Approximate Solution of Elliptic Boundary Value Problems*. PhD thesis, Computing Center of Siberian Division of Russian Academy of Sciences, 1986.

[Osw94] P. Oswald. *Multilevel Finite Element Approximation*. Teubner Skripten zur Numerik. Teubner, 1994.

[Sch98] C. Schwab. *p- and hp-Finite Element Methods*. Oxford University Press, 1998.

[SMPZ08] J. Schöberl, J.M. Melenk, C. Pechstein, and S. Zaglmayr. Additive schwarz preconditioning for p-version triangular and tetrahedral finite elements. *IMA J. Numer. Anal. 28 (2008), pp. 1-24*, 2008.

[TW05] A. Toselli and O. Widlund. *Domain Decomposition Methods — Algorithms and Theory*. Springer Verlag, 2005.

[Ver96] R. Verfürth. *A review of a posteriori error estimation and adaptive mesh refinement techniques*. Teubner-Wiley, 1996.

[Xu90] J. Xu. Iterative methods by space decomposition and subspace correction: A unifying approach. Report No. AM 67, Dep. of Mathematics, Penn. State University, 1990.

[Xu92] Jinchao Xu. Iterative methods by space decomposition and subspace correction. *SIAM Review*, 34:581–613, 1992.

[Zha92] S. Zhang. Multilevel schwarz methods. *Numer. Math.*, 63:512–539, 1992.

Chemnitz Scientific Computing Preprints – ISSN 1864-0087