An Efficient Parallel Implementation of the MSPAI Preconditioner[☆]

T. Huckle[*,a], A. Kallischko[a], A. Roy[a], M. Sedlacek[a], T. Weinzierl[a]

[a] *Technische Universität München, Boltzmannstr. 3, 80748 Garching, Germany*

Abstract

We present an efficient implementation of the Modified SParse Approximate Inverse (MSPAI) preconditioner. MSPAI generalizes the class of preconditioners based on Frobenius norm minimizations, the class of modified preconditioners such as MILU, as well as interface probing techniques in domain decomposition: it adds probing constraints to the basic SPAI formulation, and one can thus optimize the preconditioner relative to certain subspaces. We demonstrate MSPAI's qualities for iterative regularization problems arising from image deblurring.

Such applications demand for a fast and parallel preconditioner realization. We present such an implementation introducing two new optimization techniques: First, we avoid redundant calculations using a dictionary. Second, our implementation reduces the runtime spent on the most demanding numerical parts as the code switches to sparse QR decomposition methods wherever profitable. The optimized code runs in parallel with a dynamic load balancing.

Key words: preconditioners, iterative methods, sparse matrices, regularization, dictionary, parallel computing

1. Introduction

We consider the iterative solution of a large, sparse, and ill-conditioned linear system of equations $Ax = b$ with $A \in \mathbb{R}^{n\times n}$, $x,\ b \in \mathbb{R}^n$. Hereby, the choice of the preconditioner often has a stronger impact on the convergence behavior than the choice of the iterative solver such as BiCGstab, GMRES, or QMR. In [14, 15], we extend and generalize the class of *SParse Approximate Inverse (SPAI)* preconditioners to the *Modified SPAI (MSPAI)* formulation: here, the combination of targeting approaches [12] and classical probing methods [4] enables us to optimize the preconditioner on problem-dependent subspaces, and, thus, to derive

[☆]This work has partially been funded by Bund der Freunde der Technischen Universität München e.V. (BdF), Antrag 2008-09.

[*]Corresponding author

Email addresses: huckle@in.tum.de (T. Huckle), sedlacek@in.tum.de (M. Sedlacek)

filtering preconditioners. MSPAI's effectiveness is already demonstrated for several fields of applications such as preconditioning Schur complements in domain decomposition, conventional PDE matrices, or Stokes problems [14, 15].

In this paper, we apply MSPAI as promising preconditioner in iterative regularization methods arising from image deblurring. Hereby, the image has to be reconstructed fast and as accurate as possible, while the noise in the image is to be eliminated. The image and the noise are two different problem-dependent subspaces, which can be addressed by MSPAI in different ways.

Afterwards, we present an efficient, i.e. fast, parallel, and scalable, MSPAI code implementing three sophisticated features:

1. Matrices with a structured sparsity pattern entail the solution of many identical Least Squares (LS) problems throughout the computation of an MSPAI. We eliminate these redundancies (Section 3.1) as we introduce a dictionary holding the intermediate results. A hash table is a straightforward implementation of such a dictionary. Yet, it turns out that a simpler cache-type realization with a last recently used update strategy works as well, but comes along with fixed memory requirements. Fixed memory requirements are important for huge-sized problems where storing all the intermediate results exceeds the memory.
2. A data decomposition approach makes our MSPAI implementation run in parallel on a distributed memory machine (Section 3.2). The individual nodes benefit from the dictionary introduced before, too. Our code supports the prescription of a maximum sparsity pattern for the preconditioner. This idea introduced in [13] predicts the sparsity structure of a SPAI-type preconditioner, i.e. it predicts the data to be exchanged and enables the code to exchange a smaller number of bigger messages which minimizes the communication overhead.
3. Sparse matrices often lead to sparse LS problems throughout an MSPAI computation. We speed up this expensive computational subtask as we switch from a full matrix QR decomposition to a sparse one (see Davis [6]) wherever suitable (Section 3.3).

Our performance improvements exploit the structure and sparsity of the matrices arising from image deblurring. Yet, image processing, where a single operator is applied on all pixels, is only one example application domain where such structured matrices occur. Matrices stemming from PDEs that are discretized on a regular grid are another example. Such matrices are taken into account throughout the numerical experiments. In the PDE world, also matrices from block-structured adaptive grids reveal the same (hierarchical) pattern over and over again.

The remainder is organized as follows: In Section 2, we explain the SPAI preconditioner and generalize it to the MSPAI method. To demonstrate the usefulness of MSPAI, we introduce an adaption for image deblurring in Section 2.2. Afterwards, we show that a realization of the MSPAI does not differ substantially from a SPAI implementation, and we write down the algorithmic

steps of such a realization (Section 2.3). The description places emphasis on the algorithm's redundant calculations, the inherent parallelism, and the computational demanding subtasks. In Section 3, we finally present an implementation exploiting computational redundancies, a tailored parallelization, and the sparse QR decompositions. Runtime results, a conclusion, and a short outlook close the discussion.

2. SPAI and SPAI variants

Benson and Frederickson [2] introduce the idea to use Frobenius norm minimizations for preconditioning purposes. The underlying principle is to construct an approximation $M \approx A^{-1}$ to the inverse of the system matrix $A \in \mathbb{R}^{n \times n}$ by

$$\min_M \|AM - E\|_F . \tag{1}$$

Sparsity is ensured by imposing a certain sparsity pattern $\mathcal{P}(M)$, which has usually a number of nonzero entries (nnz) in the order of magnitude of A's nnz. The Frobenius norm is a sum of Euclidean norms

$$\|AM - E\|_F^2 = \sum_{k=1}^{n} \|Am_k - e_k\|_2^2 .$$

Therefore, the minimization (1) decouples into the independent solution of multiple minimization problems, one for each column of M:

$$\min_{m_k} \|Am_k - e_k\|_2 , \qquad k = 1, \ldots, n. \tag{2}$$

This aspect gives preconditioners based on Frobenius norm minimizations a clear advantage over other classical preconditioners: they are inherently parallel. Since we only allow a few nonzero entries defined by $\mathcal{P}(M)$ in the solution m_k, (2) is a Least Squares (LS) problem. Grote and Huckle [8] address the problem of computing such a Frobenius norm minimization with a simple diagonal or empty start pattern and add, step by step, the most promising additional nonzero entries in order to improve the approximation. Their Frobenius norm minimization enriched by pattern updates is well-known as the SPAI preconditioner.

2.1. Modified SParse Approximate Inverses – MSPAI

In [14], we generalize the SPAI minimization (1) similar to the targeting by Holland, Shaw, and Wathen [12]. We combine this approach with classical probing techniques [4], which are, for example, applied for preconditioning Schur complements in domain decomposition problems. In contrast to the classical probing, our basic formulation

$$\min_M \|CM - B\|_F = \min_M \left\| \begin{pmatrix} C_0 \\ \rho e^T C_0 \end{pmatrix} M - \begin{pmatrix} B_0 \\ \rho e^T B_0 \end{pmatrix} \right\|_F \tag{3}$$

is not restricted to special probing subspaces as it allows any choice of e. The resulting preconditioner M satisfies

$$\begin{aligned} C_0 M &\approx B_0, \qquad \text{and} \\ e^T C_0 M &\approx e^T B_0. \end{aligned} \tag{4}$$

We refer to the first n rows of (3), i.e. $C_0 M - B_0$, as *full approximation part* and to the additional rows as *probing part*. The weight $\rho \geq 0$ enables us to control how much emphasis is put on the *probing constraints* (4), and the matrix $e \in \mathbb{R}^{n \times k}$ contains the k subspaces on which the preconditioner should be optimal. Choosing $\rho = 0$ and $C_0 = A,\ B_0 = E$ in (3) leads to the classical SPAI formulation. Setting $C_0 = E$ and $B_0 = A$, we end up with a formulation computing explicit approximations to A. This approximation can have considerably fewer nnz than A but an equivalent action on e. The field of applications is versatile: we can improve preconditioners resulting from ILU, IC, FSAI, FSPAI, or AINV (see [3] for an overview) by adding probing information. For a given ILU factorization $LU \approx A$, e.g., we set $C_0 = L,\ B_0 = A$ and restrict the sparsity pattern of M to upper triangular form ($M := \tilde{U}$):

$$\min_{\tilde{U}} \left\| \begin{pmatrix} L \\ \rho e^T L \end{pmatrix} \tilde{U} - \begin{pmatrix} A \\ \rho e^T A \end{pmatrix} \right\|_F .$$

If we choose $e = (1, \ldots, 1)^T$, we obtain an $\tilde{U}$ preserving the row sums. Therefore, our formulation is closely related to the class of modified preconditioners (modified ILU, modified IC), and we refer to (3) as Modified SPAI (MSPAI). We also overcome the main drawbacks of MILU and classical probing such as the restriction to the vector of all ones as probing subspace and the rather difficult efficient implementation on parallel computers. The numerical examples in [14, 15] demonstrate MSPAI's effectiveness for preconditioning various PDE matrices and preconditioning Schur complements arising from domain decompositions. The regularization of image deblurring problems as a new promising field of application is the topic of the following section.

2.2. MSPAI for preconditioning in iterative regularization methods

For ill-posed problems, as they arise in image restoration, regularization techniques are important in order to recover the original information. Let us consider the linear system

$$g = Hf + \eta \tag{5}$$

where f is the original image, H is the blur operator, η is the noise, and g is the observed image. We want to recover f as good as possible and as fast as possible. Because H may be extremely ill-conditioned and because of the noise, (5) cannot be solved directly. Consequently, a regularization technique has to be applied.

We use an iterative solver such as the Conjugate Gradient method (CG) [7]. From the CG convergence analysis it is known that in the first steps the method reduces the error relative to large eigenvalues. In later steps, the eigenspectrum

related to noise and small eigenvalues dominates the evolution of the approximate solution. Therefore, the restoration has to stop after a few iterations before the method starts to reduce the error relative to the noise space.

Preconditioning accelerates the convergence while it should improve the quality of the reconstruction [9, 10, 17]. Until now, structured preconditioners like Toeplitz or circulant matrices are considered typically, whenever structured isotropic blur operators are treated. For general H, a preconditioner like ILU will lead to faster convergence but the quality of the reconstruction will deteriorate because the preconditioner also improves the solution relative to the unwanted noise subspace. Therefore, usually no preconditioners are applied for general H. We propose MSPAI probing as regularizing preconditioner for general H: Following [10], such a preconditioner M should have the following properties:

- $M \approx |H|^{-1}$ on the signal subspace with $|H| = (H^T H)^{1/2}$, and
- $M \approx E$ or $M \approx 0$ on the noise subspace.

For circulant matrices, the eigendecomposition is known and, therefore, these conditions can be satisfied by manipulating the eigenvalues. For general matrices, this is usually not possible, and we thus use the probing facility of MSPAI in order to derive a different approximation quality on the signal or noise subspace, respectively:

- For the signal space, we use the smooth vector $e_S = (1, 1, \ldots, 1)^T$. In order to allow larger subspaces, we can add numerical eigenvector estimates to k large eigenvalues, or we add k vectors of the form $(\sin\left(\frac{\pi j s}{n+1}\right)_{j=1,\ldots,n}$, $s = 1, \ldots, k)$. They also represent smooth components, and therefore the important part of the signal subspace.
- For the noise space, we use $e_N = (1, -1, 1, -1, \ldots)^T$. To allow larger subspaces we can also consider eigenvector estimates to near singular eigenvalues, or vectors of the form $(\sin\left(\frac{\pi(2j+1)(n+s)}{2(n+1)}\right)_{j=1,\ldots,n}$, $s = 1, \ldots, k)$ related to fast oscillations.

For higher dimensional problems, probing vectors typically result from a Kronecker product of 1D probing vectors. The probing conditions in MSPAI are given by $\rho_S e_S^T(AM - E)$ for the signal subspace in order to derive a good preconditioner and fast convergence on the signal subspace, and $\rho_N e_N^T(M - E)$ or $\rho_0 e_N^T(M - 0)$ for the noise subspace in order to avoid a deterioration of the reconstruction by the preconditioner. Only nonnegative weight factors are allowed.

In many examples, the signal space is large and the noise subspace is small. It then makes sense to approximate the inverse in the signal subspace and add a few probing conditions relative to the noise subspace. In other cases, the noise subspace is much larger, i.e. the preconditioner should approximate the identity, and we add only a few probing conditions relative to significant vectors

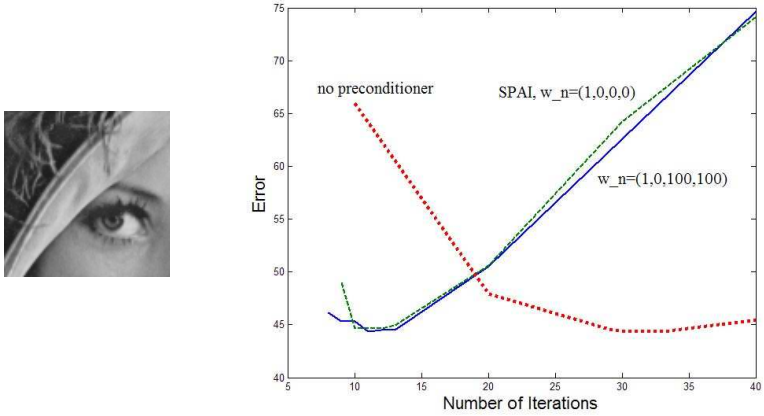

Figure 1: Image section of Lena (left) and Example 1 (right) with preconditioned CG on H_1: no preconditioner (dotted line); MSPAI to weights $w_n = (1, 0, 0, 0)$ (dashed line, equivalent to plain SPAI); MSPAI to weights $w_n = (1, 0, 100, 100)$ (solid line). The minimal error has been achieved by MSPAI probing with parameters $w_n = (1, 0, 100, 100)$ after 11 iterations.

representing the signal subspace. The general approximation condition reads as $\min_M \|AM - E\|_F^2$ on the full space in the form

$$\min_M \left(\rho_A^2 \|AM - E\|_F^2 + \rho_E^2 \|M - E\|_F^2\right) \quad \text{or} \quad \min_M \left\| \begin{pmatrix} \rho_A A \\ \rho_E E \end{pmatrix} M - \begin{pmatrix} \rho_A E \\ \rho_E E \end{pmatrix} \right\|_F^2 .$$

Together with the probing part, we end up with the minimization problem

$$\min_M \left\| \left(\begin{array}{c} \rho_A A \\ \rho_E E \\ \hline \rho_S e_S^T A \\ \rho_N e_N^T \end{array}\right) M - \left(\begin{array}{c} \rho_A E \\ \rho_E E \\ \hline \rho_S e_S^T \\ \rho_N e_N^T \end{array}\right) \right\|_F^2 \quad \text{or} \quad \min_M \left\| \left(\begin{array}{c} \rho_A A \\ \rho_E E \\ \hline \rho_S e_S^T A \\ \rho_0 e_N^T \end{array}\right) M - \left(\begin{array}{c} \rho_A E \\ \rho_E E \\ \hline \rho_S e_S^T \\ 0 \cdot e_N^T \end{array}\right) \right\|_F^2$$

with the weights $w_n := (\rho_A, \rho_E, \rho_S, \rho_N)$ for the case $M \approx E$ on the noise subspace, or $w_0 := (\rho_A, \rho_E, \rho_S, \rho_0)$ for $M \approx 0$ relative to noise, respectively. For our tests, we consider the following blur matrices:

1. With $T_1 := \text{tridiag}(1, 2, 1)$ and the identity matrix E, we set $H_1 = E \otimes T_1 + T_1 \otimes E$ relative to the filter mask $\begin{bmatrix} & 1 & \\ 1 & 4 & 1 \\ & 1 & \end{bmatrix}$. H_1 has a large signal subspace.
2. With $T_2 = \text{pentadiag}(1, 1, 1, 1, 1)$, we set $H_2 = E \otimes T_2 + T_2 \otimes E$.
3. With $t_j = \exp(-0.1j^2)$, we define the band matrix T_3 for the vector $(t_{L-1}, \ldots, t_1, t_0, t_1, \ldots, t_{L-1})$. $H_3 := T_3 \otimes T_3$ representing a 2D Gauss filter. It has a large noise subspace.

Our test image is a 100×100 section of lena.bmp (Figure 1), where we apply the operator H_k and add random noise with level 0.001. A CG method

reconstructs the original image on the normal equations, and the sparsity pattern of H acts as sparsity pattern for the preconditioner. The error is given by the difference between the recovered image and the original information in the Frobenius norm.

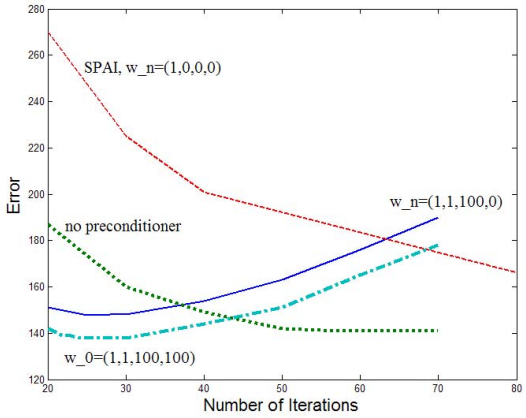

Figure 2: Example 2 with blur matrix H_2: no preconditioner (dotted line); MSPAI to weights $w_n = (1, 0, 0, 0)$ (dashed line); MSPAI with weights $w_n = (1, 1, 100, 0)$ (solid line). The best image recovering was achieved by probing $M \approx 0$ on the noise subspace with weights $w_0 = (1, 1, 100, 100)$ (dash-dot curve). For this method the minimal error has been achieved after 27 iterations.

Using MSPAI probing, it is possible both to reduce the number of iterations (Figure 1) and to recover the original image more accurately (Figures 2 and 3). The faster convergence with the MSPAI preconditioner leads to a faster deterioration of the reconstruction if we apply too many iteration steps. Without a preconditioner, the number of iterations that yields an almost optimal reconstruction is bigger, and the region of optimal approximation quality is broader and smoother, i.e. here, the CG approximation is almost stationary. Our regularization in turn should be combined with an appropriate stopping criterion [11]. The weights in MSPAI should be chosen heuristically either

- as an approximate inverse part plus probing on the noise space,
- as identity for the approximation part plus a probing part relative to the signal space,
- or as a compromise between the above.

Following Figures 1 – 3, these preliminary examples show the advantage of MSPAI probing for regularization problems. However, there are still a lot of open questions such as choosing optimal probing vectors, sparsity patterns, the stopping criterion, or the individual weights ρ.

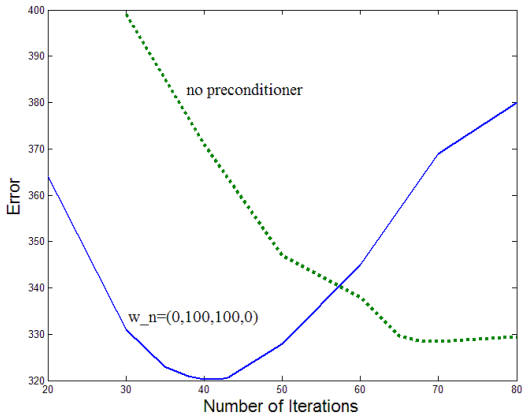

Figure 3: Example 3 with matrix H_3, $L = 3$: no preconditioner (dotted line); MSPAI with weights $w_n = (0, 100, 100, 0)$ (solid line). The minimal error has been achieved by MSPAI probing with parameters $w_n = (0, 100, 100, 0)$ after 40 iterations.

2.3. Solution of SPAI-type least squares problems

We can compute the columns of M independently from each other for both SPAI and MSPAI. For each column, the MSPAI formulation (3) leads to the solution of one LS problem of type $\min_{m_k} \|Cm_k - b_k\|_2^2$, i.e. n LS problems in total. This is exactly the same type of LS problem as (2), i.e. it is sufficient to restrict ourselves to the classical SPAI notation and to refer to this type as SPAI-type LS problem. We prescribe or let the pattern update steps identify a sparsity pattern $\mathcal{P}(m_k) \subset \mathcal{P}(M)$ $(k = 1, \ldots, n)$ for the location of nonzero entries in M. Let $\mathcal{J}$ denote the index set of entries in column m_k:

$$\mathcal{J} := \{j : \ m_k(j) \neq 0, \ j = 1, \ldots, n\}, \qquad q := |\mathcal{J}|.$$

$\mathcal{J}$ contains the q columns of A which we need when a matrix-vector product of A and $m_k(\mathcal{J})$ is evaluated. Most of the index sets, matrices, and vectors here and on the forthcoming pages are generic, i.e. they depend on k, too. Besides e_k and m_k, we however omit the k-subscripts to improve the readability.

The reduced matrix $A(., \mathcal{J})$ typically contains a lot of zero rows if A is sparse. Therefore, we define the set $\mathcal{I}$ of nonzero rows of $A(., \mathcal{J})$ as

$$\mathcal{I} := \left\{ i : \ \sum_{j \in \mathcal{J}} |a_{ij}| \neq 0, \ i = 1, \ldots, n \right\}, \qquad p := |\mathcal{I}|.$$

$\mathcal{I}$ is the *shadow of* $\mathcal{J}$, and it yields

$$\begin{aligned} \hat{A} &:= A(\mathcal{I}, \mathcal{J}) \in \mathbb{R}^{p \times q}, \\ \hat{m}_k &:= m_k(\mathcal{J}) \in \mathbb{R}^{q \times 1}, \\ \hat{e}_k &:= e_k(\mathcal{I}) \in \mathbb{R}^{p \times 1}. \end{aligned}$$

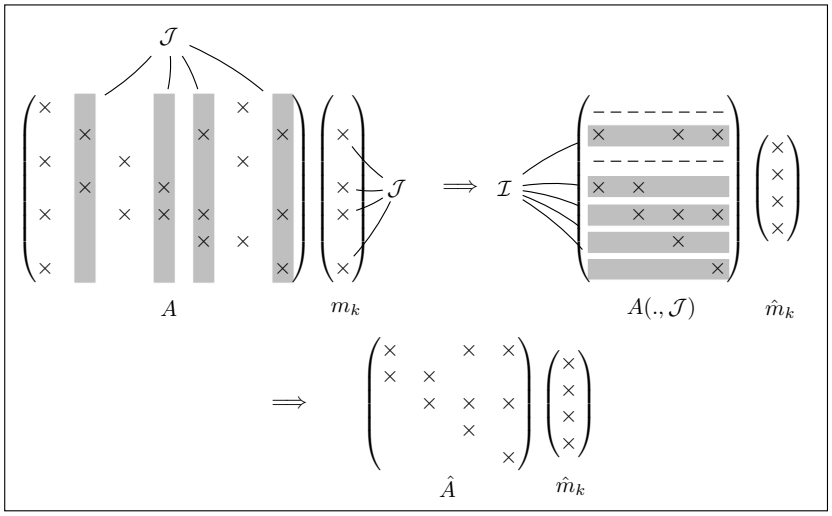

Figure 4: $\hat{A}$ is defined by the index sets $\mathcal{J}$ and the shadow $\mathcal{I}$ ($\mathcal{J} = \{2, 4, 5, 7\}$ and $\mathcal{I} = \{2, 4, 5, 6, 7\}$).

$\mathcal{I}$ and $\mathcal{J}$ formulate the LS problem (2) in its reduced form (see Figure 4) as

$$\min_{\hat{m}_k} \left\| \hat{A}\hat{m}_k - \hat{e}_k \right\|_2^2.$$

Usually, the dimension of $\hat{A}$ is considerably smaller than n, as we allow only a few entries in each column. Since $\hat{A}$ has full column rank q if A is nonsingular, we can use a QR decomposition to compute the LS solution. Let $\hat{A} = QR_0$ with orthogonal $Q \in \mathbb{R}^{p\times p}$ and $R_0 = \begin{pmatrix} R \\ 0 \end{pmatrix} \in \mathbb{R}^{p\times q}$. R holds the first q rows of R_0. We then compute the solution $\hat{m}_k$ by

$$\begin{aligned} \hat{c} &= Q^T \hat{e}_k, \qquad \text{and} \\ \hat{m}_k &= R^{-1}\hat{c}(1:q). \end{aligned} \tag{6}$$

The Householder QR decomposition of $\hat{A}$ requires $2q^2\left(p - \frac{q}{3}\right)$ operations, and it dominates the preconditioner's runtime. Its matrix Q is implicitly given by the Householder vectors. Once the entries $\hat{m}_k$ are determined, both SPAI and MSPAI improve the approximation by updating the sparsity pattern of each column. Based on the residual $r = Am_k - e_k$ and its sparsity pattern, they identify the most profitable new indices and add them to the pattern. We introduce the notation $\Upsilon_{\alpha,\beta}$ which symbolizes a maximum of α pattern update steps, each adding exactly β new indices. $\bar{\Upsilon}_{\alpha,\beta}$ indicates that only those indices are added which are more profitable than the average of all the β additional indices taken into consideration.

Updating the pattern leads to an enlarged $\hat{A}$ and another LS solution by QR decomposition. The update steps are continued until $\|r\|_2$ falls under a predefined ε or the nnz of m_k exceeds an upper bound specified by the user. For the details on the update criterion see [8].

3. Implementation

The MSPAI implementation can be reduced to a SPAI implementation for an augmented system. This system is sparse but tends to be huge. Hence, the corresponding solution task is computational demanding due to the large number of small LS problems. It requires for an efficient implementation.

3.1. Eliminating redundant calculations with a dictionary

The linear equation systems in image restoration typically exhibit a very regular structure: In the full approximation part of (3) the diagonal element is set and, relative to the diagonal, entries occur often at the same positions. These entries' values moreover are the same for many columns. In (6), reoccurring values and sparsity patterns in A lead to the same $\hat{A}$, if the index pattern is also aligned along the diagonal. This is usually the case. Same $\hat{A}$ imply the same LS problems, and the corresponding identical QR decompositions thus are computed multiple times. It is an obvious idea to avoid the redundant computations and to exploit the reoccurring patterns.

We extend our MSPAI algorithm: After a QR decomposition, the algorithm stores the decomposition and the corresponding $\hat{A}$ in a dictionary. For the subsequent $\hat{A}$, i.e. the next preconditioner column to compute, the code looks up the result in the dictionary. If it is not computed yet, the algorithm computes the decomposition and notes it down. Thus, we avoid redundant QR decompositions.

Although computations are eliminated, naively searching in the dictionary will almost for certain slow down the application. We introduce a function $key : \hat{A} \mapsto \{0, \ldots, M-1\}$, $M \in \mathbb{N}$ resembling a hash key [16] for the dictionary entries. Each entry in the dictionary then equals a four-tuple $(\hat{A}, Q, R, key(\hat{A}))$. With $\hat{A}_1 = \hat{A}_2 \Rightarrow key(\hat{A}_1) = key(\hat{A}_2)$, we apply a two-step search: If the key is not stored yet, the algorithm terminates the search and starts the QR decomposition. Otherwise, all the dictionary entries $\hat{A}$ with an equal key are compared bit-wise to the current input $\hat{A}$ until a decomposition is found. If all comparisons fail, the decomposition is computed. The comparisons and the decomposition can run in parallel; they compete with each other.

The performance improvement of the MSPAI optimization all depends on the key computation. The key has to be simple to compute, and, for a given number of different $\hat{A}$, the number of identical keys (key collisions) should be small. This is a classical hash key challenge where one long bit stream consisting of $\hat{A}$'s sparsity pattern and the matrix's values act as input sequence. For our purposes, a simple 32-bit key proved of sufficient value: Hereby, the upper and lower part of each matrix entry's bit representation are combined via an

1. Given $\hat{A}$.
2. Compute $key(\hat{A})$.
3. $key(\hat{A}) \in$ dictionary?
   (a) For each dictionary entry with same k: Compare $\hat{A}$ bit-wise to entry's $\hat{A}$.
   (b) Found same $\hat{A}$? Take result QR decomposition from dictionary and return.
   (c) Otherwise proceed to next step.
4. $key(\hat{A}) \notin$ dictionary or from (3c)?
   (a) Compute result QR decomposition.
   (b) Add $(\hat{A}, Q, R, key(\hat{A}))$ to dictionary.
   (c) Return result.

1. Given $\hat{A}$. Dictionary of size D.
2. Compute $key(\hat{A})$.
3. $key(\hat{A}) \in$ dictionary?
   (a) For each dictionary entry with same k: Compare $\hat{A}$ bit-wise to entry's $\hat{A}$. Start with first entry.
   (b) Found same $\hat{A}$ at position d?
      i. Extract result decomposition from dictionary.
      ii. Reinsert decomposition at begin of dictionary.
      iii. Return result.
   (c) Otherwise proceed to next step.
4. $key(\hat{A}) \notin$ dictionary or from (3c)?
   (a) Compute result decomposition.
   (b) If size of dictionary equals D: remove last entry.
   (c) Add $(\hat{A}, Q, R, key(\hat{A}))$ as first entry to dictionary.
   (d) Return result decomposition.

Figure 5: SPAI's QR decomposition augmented with a dictionary (left). Instead of a complete dictionary, one could also use a cache of fixed size with a last recently used update scheme (right). The latter one comes along with bounded memory requirements and a reduced number of key collisions.

exclusive OR. All the $p \cdot q$ resulting keys $key_i,\ i \in \{1, \ldots, p \cdot q\}$, $\hat{A} \in \mathbb{R}^{p \times q}$, are then combined into one key according to

$$\begin{aligned} h_1 &= key_1, \\ h_i &= h_{i-1} \cdot (32 - 1) + key_i \qquad 2 \leq i \leq p \cdot q \quad \text{and} \\ key(\hat{A}) &= h_{p \cdot q}. \end{aligned}$$

A straightforward implementation of the dictionary uses a hash table to store the calculations. Such tables come along with two drawbacks: Throughout the computation they grow and the probability of a key collision increases with the number of entries, if the algorithm does not adapt both the hash key and the hash table size. Yet, one can characterize the different $\hat{A}$ for many matrices: there is a couple of $\hat{A}$ whose entries occur very often. A second implementation of our algorithm thus implements the dictionary in a last recently used (LRU) cache manner (Figure 5, right), i.e. the dictionary's size is fixed. If the dictionary runs out of size because of an insert, the code removes the element used last recently. For this implementation, the required memory is bounded[1].

[1] We implemented two cache realizations based either on a linked list or on a fixed-sized

3.2. Parallelization

SPAI is inherently parallel due to the decoupled minimization problems in (2) and, hence, demands for a parallel implementation. This property holds for MSPAI, too. Our MSPAI parallelization strategy follows [1]: First, the different columns of M are cut into chunks of the same number of columns. These numbers differ by at most one column. The responsibilities for the individual chunks then are distributed equally among the computing nodes. Second, C's and B's columns are distributed in the same manner. Third, every node computes the columns of M it is responsible for. Finally, each node carries over responsibilities for additional columns from other working nodes as soon as it runs out of work.

We end up with a plain, low-overhead, dynamic load balancing for our parallel MSPAI implementation. A static load balancing would not be sufficient as, on the one hand, the pattern of the inverse is not known a priori and the computational work thus is not predictable. On the other hand, augmenting MSPAI with a dictionary makes it difficult to foresee the computing time required per column.

While MSPAI's column computations run independently of each other, the computation might demand for columns of B and C for which the local node is not responsible. Such columns are not available on the local node a priori. Instead, the node has to copy them from other nodes. To avoid redundant requests, all the copied data are cached.

The data exchange is optimistic, asynchronous, and overlapping, i.e. the algorithm predicts remote column accesses and exchanges data in the background: In [13] the idea of a maximum sparsity pattern is introduced. Hereby, a pattern for the approximate inverse is derived a priori. This pattern guarantees for a sufficient initial approximation. We use it to enlarge the remote column request's size, i.e. instead of requesting solely data required for the current computation, we request additional data imposed by the pattern. Thus, too many data might be exchanged for the moment, but it is likely that these columns will be used later on.

As we run our experiments on a distributed memory machine, each node holds a decomposition dictionary of its own. Dictionary entries hence are computed and stored multiple times on different nodes. Searching in the dictionary comes along without any communication and data exchange. However, for a cluster with shared memory nodes, shared dictionaries might be of great use.

3.3. Sparse QR decomposition

The QR decomposition in (6) is the most expensive computation of the overall algorithm. Although $\hat{A}$ is small compared to A and, typically, much denser, it still exposes a sparse pattern if $\mathcal{P}(M)$ is sufficiently sparse. It is an obvious idea to make the algorithm benefit from a sparse QR decomposition implementation for $\hat{A}$.

---

array. The linked list yields better results.

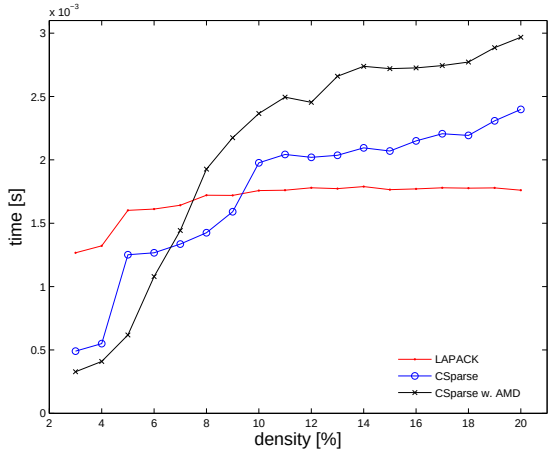

Figure 6: Runtime for different implementations of the QR decomposition. One 100×100 random matrix acts as input matrix. The plot compares LAPACK's implementation with the CSparse[2] [6] package with and without AMD preordering. The break-even here is lower than the one we observed in tests with matrices from Matrix Market[3].

Yet, for sufficiently dense matrices, sparse QR decomposition approaches come along with a lower MFLOP rate and a higher overhead if one compares them to QR decompositions working on full matrices (Figure 6). As we do not know the structure of all the matrices a priori, we can not restrict ourselves to one hardwired type of QR decomposition. Instead, we examine $\hat{A}$'s nnz compared to $p \cdot q$ to choose an appropriate implementation for each computation.

If $\frac{1}{p \cdot q}\text{nnz}(\hat{A}) < \varepsilon_{\text{QR}}$, we invoke a sparse QR decomposition based upon a compressed sparse column format. Our implementation uses CSparse, since this package implements Householder transformations using sparse data structures. In case of denser submatrices, we convert the internal representation of $\hat{A}$ into a full matrix and pass it to standard LAPACK/Atlas routines. Extensive runtime tests with a large set of test matrices from Matrix Market reveal $\varepsilon_{\text{QR}} = 15\%$ being a meaningful default value in our implementation. Of course, this value can be modified by the user, and, of course, it depends on the actual machine and the library implementation used. An approximate minimum degree reordering as preprocessing step in order to reduce the fill-in during the QR decomposition is also offered by CSparse but not taken over into our current implementation.

4. Runtime results

Here, we study the optimization techniques introduced in the previous sections. All the experiments were conducted on AMD Opteron 850 processors at

[2] `http://www.cise.ufl.edu/research/sparse/CSparse`
[3] `http://math.nist.gov/MatrixMarket`

Table 1: MSPAI 1.1's behavior for different matrices. All measurements are normalized with respect to the standard SPAI without optimizations, i.e. they give speedups. MSPAI's dictionary is realized either as hash table or as a LRU cache of size 60. The QR decompositions are based upon LAPACK with $\bar{\Upsilon}_{0,0}$, $\epsilon = 10^{-2}$, and the SPAI pattern of A.

matrix		hash table		LRU cache
name	dimension	speedup	dictionary entries	speedup
laplace2d 1o10	100	1.37	24	1.37
laplace2d 2o10	100	0.87	24	0.87
orsirr 2	886	0.78	776	0.88
laplace3d 1o10	1000	1.85	125	1.84
pores 2	1224	0.68	1221	0.78
olm2000	2000	1.93	5	1.92
laplace2d 1o50	2500	2.92	24	2.92
fidap029	2870	0.67	2692	0.80
rdb3200L	3200	2.64	50	2.60
laplace2d 2o60	3600	2.19	81	2.16
CFD small	4096	2.25	145	2.23
CFD large	8192	2.14	385	2.14
laplace2d 1o100	10^4	3.02	24	2.95
laplace2d 2o100	10^4	2.32	81	2.26
laplace3d 1o22	$1.0648 \cdot 10^4$	2.42	125	2.36
laplace2d 1o317	$1.00489 \cdot 10^5$	3.14	24	3.06
laplace2d 2o317	$1.00489 \cdot 10^5$	2.38	81	2.35
laplace3d 1o47	$1.03823 \cdot 10^5$	2.37	125	2.32
laplace2d 1o1000	10^6	2.54	24	2.87
laplace2d 2o1000	10^6	2.09	81	2.05
laplace3d 1o100	10^6	2.39	125	2.38

2.4 GHz with 8 GByte of main memory. For our parallel MSPAI implementation with MPI, we used a cluster of such processors connected by InfiniBand. It is referred as InfiniCluster.

As we wanted to apply our efficient preconditioner to problems from other fields of applications, to problems yielding more irregular matrices than those from image deblurring (laplace2d 2o10, laplace2d 1o50, laplace2d 2o60, ...), as well as matrices of a bigger size, we decided to study the implementation's behavior for several matrices from Matrix Market, matrices resulting from computational fluid dynamics [15], and laplacian matrices generated on our own. Besides the standard 2D and 3D laplacian matrices, named laplace2d 1^* and laplace3d 1^*, we built generalized 2D laplacian matrices using a 13 point stencil. They are called laplace2d 2^*.

We give the runtime improvement resulting from a dictionary for a SPAI without pattern updates in Table 1 and with pattern updates in Table 2. Both results stem from one node of the InfiniCluster. This performance optimization

Table 2: MSPAI 1.1's behavior for different matrices. All measurements are normalized with respect to the standard SPAI without optimizations, i.e. they give speedups. Both preconditioners use pattern updates with a tolerance of $\epsilon = 10^{-2}$ and a diagonal start pattern. MSPAI's dictionary is realized either as hash table or as a LRU cache with user-defined cache size. The QR decompositions are based upon LAPACK.

matrix			hash table	LRU cache	
name	dimension	setting	speedup	speedup	cache size
laplace2d 1o10	100	$\bar{\Upsilon}_{6,5}$	1.01	1.06	60
laplace2d 2o10	100	$\bar{\Upsilon}_{6,5}$	0.76	0.81	60
orsirr 2	886	$\bar{\Upsilon}_{10,6}$	0.80	0.80	20
laplace3d 1o10	1000	$\bar{\Upsilon}_{12,8}$	1.12	1.11	1000
pores 2	1224	$\bar{\Upsilon}_{12,8}$	0.76	0.80	60
olm2000	2000	$\bar{\Upsilon}_{10,8}$	4.60	4.55	500
laplace1o50	2500	$\bar{\Upsilon}_{10,8}$	3.36	3.32	1000
fidap029	2870	$\bar{\Upsilon}_{12,10}$	0.71	0.82	20
rdb3200L	3200	$\bar{\Upsilon}_{10,8}$	3.47	3.38	1000
laplace2o60	3600	$\bar{\Upsilon}_{8,8}$	2.78	2.75	800
CFD small	4096	$\bar{\Upsilon}_{8,6}$	1.25	1.24	1000
CFD large	8192	$\bar{\Upsilon}_{6,6}$	1.25	1.20	800
laplace2d 1o100	10^4	$\bar{\Upsilon}_{5,8}$	1.70	1.69	60
laplace2d 2o100	10^4	$\bar{\Upsilon}_{5,8}$	1.81	1.79	1000
laplace3d 1o22	$1.0648 \cdot 10^4$	$\bar{\Upsilon}_{7,8}$	1.81	1.79	1000
laplace2d 1o317	$1.00489 \cdot 10^5$	$\bar{\Upsilon}_{5,5}$	2.00	2.00	60
laplace2d 2o317	$1.00489 \cdot 10^5$	$\bar{\Upsilon}_{3,4}$	1.25	1.25	60
laplace3d 1o47	$1.03823 \cdot 10^5$	$\bar{\Upsilon}_{4,5}$	1.48	1.47	1000
laplace2d 1o1000	10^6	$\bar{\Upsilon}_{6,8}$	2.28	2.24	1000
laplace2d 2o1000	10^6	$\bar{\Upsilon}_{6,8}$	1.62	1.61	1000
laplace3d 1o100	10^6	$\bar{\Upsilon}_{6,8}$	1.81	1.80	1000

proves to be robust for almost all matrices, i.e. the algorithm is seldom slower than an implementation without a dictionary. The more identical decompositions occur – column four displays the maximum number of different patterns, i.e. the maximum size of the hash table – the faster the implementation; the memory overhead hereby is determined by the number and memory footprint of the individual dictionary entries.

An LRU scheme with a prescribed maximum dictionary size and a fixed upper threshold of fill-ins bounds the memory requirements of the implementation. This reduction to a fixed set of dictionary entries does not lead to a significant performance breakdown for our test matrices – the breakdowns in the measurements are due to small experiment sizes where measurement noise pollutes the figures. Besides the results listed, we also compared the runtimes for different cache sizes. It turned out that highly structured matrices benefit from a big cache that was able to hold all decompositions (cf. Table 1). Unstructured

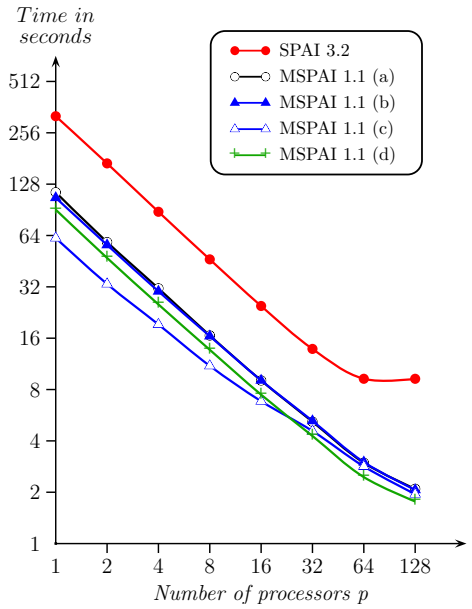

Figure 7: Time to compute M for SPAI 3.2[4] and our MSPAI 1.1 realization. We give the runtime achieved through MSPAI without any optimization (a), a dictionary based upon a LRU cache with default cache size 60 (b) and increased cache size 2000 (c). Finally, we present runtimes for the sparse QR decompositions using CSparse without any cache (d). All experiments result from the computational fluid dynamics test matrix CFD large [15] with dimension $n = 8192$, $\bar{\Upsilon}_{12,5}$, $\epsilon = 10^{-3}$ and a diagonal start pattern.

matrices could not take advantage of any dictionary, as, here, the algorithm's performance suffers from the additional key computations. In an improved version, one can deploy both the QR decomposition and the key computation to a thread of their own, and make them compete to deliver the decomposition. This multicore parallelization is switched off here. All the insights also hold for the MSPAI with pattern updates (Table 2).

As the pattern of the MSPAI is not fixed in advance, the parallel algorithm exhibits a sophisticated communication scheme with nodes requesting columns from other nodes throughout the computation. While the SPAI implementations, both the well-established SPAI 3.2 and our new realization, scale almost linearly – the deterioration for more then 32 or 64 nodes, respectively, is due to the strong scaling, i.e. the problem size is not increased with the number of nodes – our optimization techniques speed up the standard parallel SPAI implementation by at least a factor of two (Figure 7): the dictionary modification with one dictionary per computing node and the sparse QR decomposition implementation selection hereby fit perfectly into the parallelization concept.

[4] `http://www.computational.unibas.ch/software/spai`

However, using a dictionary has two side effects. On the one hand, both the dictionary entries and the remote columns fetched throughout the previous QR decompositions are cached locally and thus increase the memory consumption per node. Prescribing an upper memory footprint of these caches, we ensure that no node of the cluster runs out of memory: Whenever the cache size exceeds the given threshold, we clear it. In the experiments, we chose the threshold such that MSPAI, the dictionary, and the remote cache fit into the local memory of 8 GByte per node. The memory side effect hence is controllable and deterministic. On the other hand, the individual dictionaries typically hold some entries redundantly as the dictionaries act independently of each other, i.e. multiple nodes compute the same entries. As a result, the averaged dictionary hit rate, i.e. the global number of successful dictionary searches on all nodes divided by the global total number of dictionary accesses, decreases; from 84% on a single node in the example (c) down to 57% on 64 nodes and only 53% on 128 nodes. This decay is the bigger the bigger the dictionary is and measurements for different dictionary sizes thus converge to each other (measurement (b) and (c)).

Finally, we study the impact of the QR decomposition realization on the total runtime of the MSPAI (Table 3) in detail. MSPAI's implementation can switch from a dense realization to a sparse one, and the figures reveal that this speeds up the computation for sufficiently sparse systems. For dense systems, the standard LAPACK/Atlas routines are of greater value. While an empirical study of an optimal switch threshold is beyond the scope of this paper, the figures in Table 3 give a first impression of the advantage of a well-chosen ϵ_{QR}.

5. Conclusion and outlook

In this paper, we show how iterative regularization methods in image deblurring benefit from MSPAI compared to the non-preconditioned case. The benefit is twofold as both the number of required iterations reduces and the accuracy of the reconstructed image improves. The latter aspect holds due to MSPAI's new probing and targeting extension. Probing has already proved of great value for other applications. Besides the new application domain, we place special emphasis on the combination of different probing vectors allowing us to determine the precision of the solution in a solution subspace. Thus, we end up with a multiscale approach in the Fourier space, i.e. we were able to control the preconditioner's behavior on different frequency subspaces.

Any sophisticated preconditioner is of great value if and only if its computation is sufficiently cheap and simple. SPAI-type preconditioners reduce the computation of the preconditioner to a large number of independent simple minimization problems, and, thus, they exhibit a very simple, embarrassingly parallel structure compared to other preconditioners. This simplicity is preserved by the MSPAI extension.

The relevance to practice of a MSPAI implementation sinks or swims with the runtime performance of its implementation. We present an efficient realization introducing two new MSPAI optimizations. First, we analyze the density of

Table 3: MSPAI 1.1's behavior for different matrices from Matrix Market, computational fluid dynamics [15] and laplacian matrices generated on our own. The table compares LAPACK with CSparse for solving the LS problems in SPAI. The tests were performed on one node of the InfiniCluster with $\epsilon = 10^{-3}$ and the SPAI pattern of A.

matrix			CSparse
name	dimension	setting	speedup
laplace2d 1o10	100	$\bar{\Upsilon}_{10,8}$	1.48
laplace2d 2o10	100	$\bar{\Upsilon}_{10,8}$	1.14
orsirr 2	886	$\bar{\Upsilon}_{14,8}$	2.20
laplace3d 1o10	1000	$\bar{\Upsilon}_{12,8}$	1.84
pores 2	1224	$\bar{\Upsilon}_{16,10}$	4.39
olm2000	2000	$\bar{\Upsilon}_{15,8}$	4.55
laplace1o50	2500	$\bar{\Upsilon}_{14,10}$	3.15
fidap029	2870	$\bar{\Upsilon}_{18,15}$	1.57
rdb3200L	3200	$\bar{\Upsilon}_{15,10}$	3.09
laplace2o60	3600	$\bar{\Upsilon}_{10,8}$	1.30
CFD small	4096	$\bar{\Upsilon}_{10,6}$	1.30
CFD large	8192	$\bar{\Upsilon}_{8,6}$	1.21
laplace2d 1o100	10^4	$\bar{\Upsilon}_{10,8}$	1.74
laplace2d 2o100	10^4	$\bar{\Upsilon}_{10,8}$	1.29
laplace3d 1o22	$1.0648 \cdot 10^4$	$\bar{\Upsilon}_{8,8}$	1.27
laplace2d 1o317	$1.00489 \cdot 10^5$	$\bar{\Upsilon}_{10,8}$	1.77
laplace2d 2o317	$1.00489 \cdot 10^5$	$\bar{\Upsilon}_{10,8}$	1.14
laplace3d 1o47	$1.03823 \cdot 10^5$	$\bar{\Upsilon}_{10,8}$	1.50

every underlying minimization problem. For both sufficient sparse and dense systems, optimized libraries do exist. Thus, we take into account the sparsity and switch to an appropriate implementation. Second, intermediate calculations occurring over and over again are stored in a dictionary and, thus, redundant QR decompositions are avoided. While both optimizations reduce the runtime required, they do not harm the parallel efficiency and introduce a bounded memory overhead. The implementation is available at our website[5]. Besides the features presented in this paper, the version online also supports complex valued systems.

We believe that other applications can benefit from the MSPAI extension and the idea of a multispace probing, too. Besides multiscale behavior in the frequency spaces, systems arising from PDEs in Physics, e.g., could benefit from different probing vectors if the preconditioner also preserves conservation laws. Apart from properties of the linear equation system that has to be approximated, the interplay of a MSPAI preconditioner optimized on certain solution spaces and multiscale solvers is yet to be studied.

[5] `http://www5.in.tum.de/wiki/index.php/MSPAI`

The improved parallel implementation allows to experiment with huge systems arising from real world problems on massive parallel clusters. While results for such systems have been studied carefully [1, 5] for other SPAI variants and implementations, such a study is beyond the scope of this paper. Complex valued systems are beyond the scope, too, although our implementation supports them. We place great emphasis to preserve MSPAI's parallel properties for all optimizations. Nevertheless, we have to examine concepts for shared memory systems in more detail. With all the multicore systems coming up, a hybrid implementation supporting both shared and distributed memory systems is of importance.

References

[1] S. T. Barnard, L. M. Bernardo, H. D. Simon, An MPI implementation of the SPAI preconditioner on the T3E, International Journal of High Performance Computing Applications 13 (2) (1999) 107–123.

[2] M. W. Benson, P. O. Frederickson, Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems, Utilitas Mathematica 22 (1982) 127–140.

[3] M. Benzi, M. Tůma, A comparative study of sparse approximate inverse preconditioners, Appl. Numer. Math. 30 (1999) 305–340.

[4] T. F. Chan, T. P. Mathew, The interface probing technique in domain decomposition, SIAM J. Mat. Anal. Appl. 13 (1) (1992) 212–238.

[5] E. Chow, Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns, Int. J. High Perf. Comput. Appl 15 (2001) 56–74.

[6] T. Davis, Direct Methods for Sparse Linear Systems, SIAM, 2006.

[7] H. W. Engl, M. Hanke, A. Neubauer, Regularization of Inverse Problems, Kluwer, 1996.

[8] M. Grote, T. Huckle, Parallel preconditioning with sparse approximate inverses, SIAM J. Sci. Comput. 18 (1997) 838–853.

[9] M. Hanke, Iterative regularization techniques in image reconstruction, in: Proceedings of the Conference Mathematical Methods in Inverse Problems for Partial Differential Equations, Mt. Holyoke, 1998.

[10] M. Hanke, J. G. Nagy, R. J. Plemmons, Preconditioned iterative regularization for ill-posed problems, Numerical Linear Algebra and Scientific Computing (1993) 141–163.

[11] P. Hansen, Analysis of discrete ill-posed problems by means of the l-curve, SIAM Review 34 (1992) 561–580.

[12] R. M. Holland, G. J. Shaw, A. J. Wathen, Sparse approximate inverses and target matrices, SIAM J. Sci. Comp. 26 (3) (1992) 1000–1011.

[13] T. Huckle, Approximate sparsity patterns for the inverse of a matrix and preconditioning, Appl. Numer. Math. 30 (2003) 291–303.

[14] T. Huckle, A. Kallischko, Frobenius norm minimization and probing for preconditioning, International Journal of Computer Mathematics 84 (8) (2007) 1225–1248.

[15] A. Kallischko, Modified sparse approximate inverses (MSPAI) for parallel preconditioning, Ph.D. thesis, Fakultät für Mathematik, Technische Universität München (Mar. 2008).

[16] D. E. Knuth, The Art of Computer Programming Vol. 3: Sorting and Searching, 2nd ed., Addison-Wesley, 1998.

[17] J. G. Nagy, R. J. Plemmons, T. C. Torgersen, Iterative image restoration using approximate inverse preconditioning, IEEE Transactions on Image Processing 5 (1996) 1151–1162.