Parallel Computing 71 (2018) 1–22

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

# Incomplete Sparse Approximate Inverses for Parallel Preconditioning

Hartwig Anzt[a,b,*], Thomas K. Huckle[c], Jürgen Bräckle[c], Jack Dongarra[b,d,e]

[a] Karlsruhe Institute of Technology, Germany
[b] Innovative Computing Lab, University of Tennessee, USA
[c] Department of Informatics, Technical University of Munich, Germany
[d] School of Computer Science, University of Manchester, UK
[e] Oak Ridge National Laboratory, USA

ARTICLE INFO

*Article history:*
Received 9 November 2016
Revised 20 July 2017
Accepted 24 October 2017
Available online 28 October 2017

*Keywords:*
Preconditioning
Incomplete Sparse Approximate Inverse
Incomplete LU factorization
Approximate sparse triangular solves
Parallel computing

ABSTRACT

In this paper, we propose a new preconditioning method that can be seen as a generalization of block-Jacobi methods, or as a simplification of the sparse approximate inverse (SAI) preconditioners. The "Incomplete Sparse Approximate Inverses" (ISAI) is in particular efficient in the solution of sparse triangular linear systems of equations. Those arise, for example, in the context of incomplete factorization preconditioning. ISAI preconditioners can be generated via an algorithm providing fine-grained parallelism, which makes them attractive for hardware with a high concurrency level. In a study covering a large number of matrices, we identify the ISAI preconditioner as an attractive alternative to exact triangular solves in the context of incomplete factorization preconditioning.



## 1. Introduction

In the iterative solution process of large, sparse linear systems via Krylov methods, preconditioners are an important building block facilitating satisfactory convergence. The idea is to turn the original system $Ax = b$ into a (left-) preconditioned system $MAx = Mb$ ($AMy = b$, $x = My$ for right-preconditioning), which allows for faster convergence of the Krylov solver. The convergence characteristics typically depend on the conditioning of the target system. For an ill-conditioned $A$, the preconditioner is also required to be ill-conditioned. Otherwise, the preconditioner can not be expected to improve the conditioning of the problem or the convergence of the Krylov solver.

At the same time, the preconditioner should be easy to derive and apply. On computing architectures with a high level of concurrency, a certain preconditioner is only attractive if its parallelism level is somewhat competitive to the parallelism level in the iterative solver. Otherwise, the preconditioner can become a computational bottleneck.

Sparse Approximate Inverses preconditioners (SAI [1]) minimize $||AM - I||_F$ in the Frobenius norm. They typically show good parallel performance but often fail to provide substantial convergence improvement [1–7]. Preconditioners based on incomplete factorizations of $A$ (like incomplete LU factorizations, ILU [8]), are often better preconditioners. Unfortunately, ILU-based preconditioners come at the cost of two (sparse) triangular solves in every preconditioner application. The sequential nature of exact triangular solves makes the preconditioner application particularly expensive on parallel architectures [9,10]. Also the generation of the incomplete factorization via a truncated Gaussian elimination process can become

* Corresponding author at: Karlsruhe Institute of Technology, Germany.
*E-mail address:* hanzt@icl.utk.edu (H. Anzt).

https://doi.org/10.1016/j.parco.2017.10.003
0167-8191/

a bottleneck: the approach of exploiting the inherent parallelism via "level scheduling" [8] often has limited scalability, as the sets of unknowns that can be computed in parallel are typically much smaller than the hardware concurrency. Other parallelization strategies, such as multicoloring and domain decomposition techniques, can be used to artificially enhance the available parallelism at the cost of degraded preconditioner quality [11–14]. Ultimately, all these efforts spent on parallelizing the incomplete factorization process are unable to exploit the computing performance of thousands of light-weight cores.

Chow recently proposed a different strategy for generating an incomplete factorization via a parallel algorithm [15]. The idea is to use fixed-point iterations for approximating the sparse triangular factors. Especially on manycore architectures that feature high concurrency levels, this approach can be much faster than the truncated Gaussian elimination process traditionally used [15–17].

The potential ill conditioning of the ILU factorizations is a minor problem if the occurring triangular linear systems are solved exactly via forward and backward triangular substitution. But these routines are inherently sequential and therefore unattractive on parallel architectures. Hence, as an alternative, fast-converging iterative methods for triangular matrices that allowing efficient parallelization are interesting. For problems where only a few relaxation steps of Jacobi generate a good approximate solution to the sparse triangular system, the preconditioner quality provided to the outer Krylov method is competitive. In this case, the Jacobi-based approach can be much faster than exact triangular solves [15,18]. Unfortunately, many Jacobi steps may be required for ill-conditioned triangular systems. For problems where the incomplete factors inherently carry some block structure, block-Jacobi can be more successful [19]. Recently, several other strategies for approximating the solution of the triangular systems coming with the incomplete LU factors have also been evaluated. This includes the use of a sparse matrix approximating the inverse of the triangular factors as well as the use of fixed-point iterations. The idea of using sparse approximate inverses (SAI) suffers from the fact that the inverse of a sparse matrix is not generally sparse, and approximating the inverse on a preset nonzero pattern can hamper the quality of the preconditioner.

In this paper we show the intimate connection between incomplete (ILU-like) preconditioners and sparse inverse approximations based on the minimization of the Frobenius norm. This leads to a new "Incomplete Sparse Approximate Inverse (ISAI)" preconditioner, that can be seen as a generalization of a block-Jacobi matrix or a simplification of SAI. For matrix $A$, the new preconditioner - similar to ILU - is derived by solving $(AM - I)_{\mathcal{S}} = 0$ on given pattern $\mathcal{S}$, where $\mathcal{S}$ in the simplest case is the pattern of $A$. The block-Jacobi method can be seen as solving this equation $(AM - I)_{\mathcal{S}} = 0$ for a block diagonal pattern $\mathcal{S}$, while SAI instead solves a related Least Squares Problem $min||AM - I||_F$. We call this preconditioner Incomplete Sparse Approximate Inverse (ISAI) because it combines the incompleteness approach $(A - LU)_{\mathcal{S}} = 0$ of ILU with the norm minimization of SAI.

Although the scope of the ISAI preconditioner goes beyond triangular systems, we in particular focus on its use and efficiency in the linear systems arising from the triangular factors in the context of ILU preconditioning. This combination of incomplete factorization preconditioning with approximate triangular solves using an ISAI preconditioner results in an efficient preconditioning strategy for solving linear systems.

This paper is structured as follows. Section 2 provides some background on preconditioning linear systems. We show that preconditioners are required to be ill-conditioned for significantly improving the iterative solution process. We relate to the concept of splitting methods and give an overview of some popular preconditioning strategies. We also address the challenge of solving sparse triangular systems in incomplete factorization preconditioning and list some efforts aiming at the efficient realization on parallel hardware architectures. The Incomplete Sparse Approximate Inverse (ISAI) preconditioner that we present in Section 3 derives as a combination of the incompleteness strategies used in Jacobi, ILU, or SAI, with the Frobenius norm minimization. We use some small example problems for illustrating the ISAI idea and comparing it with the block-Jacobi and the SAI strategies. In Section 4 we present an algorithm that allows for the efficient generation of ISAI preconditioners via "batched routines [20]." In the experimental part of the paper, we show in Section 5 that using ISAI for the triangular solves arising in ILU preconditioning can be very successful on parallel architectures. We show that the implementation based on batched routines succeeds in keeping the cost of the preconditioner generation low, and use a large set of test matrices from the SuiteSparse Matrix Collection to validate the practical benefit of the ISAI preconditioner. A summary of the findings and a short outlook on future research directions is given in Section 6.

## 2. Preconditioning for iterative solvers

In this section we give a short overview about the concept of preconditioning, show the need for the preconditioner to be ill-conditioned, and review some of the most popular preconditioning techniques.

### 2.1. Conditioning of preconditioners

For improving the convergence of an iterative solver, (e.g., a Krylov method), a preconditioner $M$ should ideally transform an ill-conditioned system $Ax = b$ into a well-conditioned system $MAx = Mb$. Precisely, the condition number $\text{cond}_2(MA)$ should be much smaller than $\text{cond}_2(A)$. This transformation can succeed only if the preconditioner itself is ill-conditioned:

**Theorem 1.** *Let $A$ and $M$ be nonsingular $n \times n$ matrices. Then the condition numbers satisfy the relation:*

$$\frac{\text{cond}_2(A)}{\text{cond}_2(M)} \leq \text{cond}_2(MA) \leq \text{cond}_2(A) \cdot \text{cond}_2(M). \tag{1}$$

**Proof.** The right side of the above inequality is obvious from the submultiplicativity of the matrix norm. To prove the left side, we apply the inequality for the right-hand side in the form:

$$\text{cond}_2(A) = \text{cond}_2(M^{-1}MA) \leq \text{cond}_2(M^{-1})\,\text{cond}_2(MA) = \text{cond}_2(M)\,\text{cond}_2(MA). \quad \square$$

Theorem 1shows that for preconditioning an ill-conditioned system, an efficient preconditioner will be ill-conditioned itself. For ILU preconditioning, this implies that a good incomplete factorization preconditioner will come with ill-conditioned triangular factors. The situation is somewhat improved as both factors can partly contribute to the ill-conditioning, but the product of the triangular factors $L \cdot U$ has to be ill-conditioned for significant convergence improvement. The best choice may be given by both factors equally contributing to the ill-conditioning ($\text{cond}_2(L) \approx \sqrt{\text{cond}} \approx \text{cond}_2(U)$). Nevertheless, a strong solution method for the resulting triangular systems in $L$ and $U$ is required, which can be an exact solve via forward and backward substitution, or a robust iterative method.

### 2.2. Relaxation methods

The origin of relaxation methods for linear systems – also called "stationary iterative solvers" is a matrix splitting of the form:

$$A = P - N, \tag{2}$$

and a rewriting of the given linear system in the form:

$$b = Ax = (P - N)x = Px - Nx.$$

For $P$ invertible, this can be formulated as fixed-point iteration:

$$\begin{aligned} x^{(k+1)} &= P^{-1}(b + Nx^{(k)}) \\ &= x^{(k)} + P^{-1}(b - Ax^{(k)}) \\ &= P^{-1}b + (I - P^{-1}A)x^{(k)}. \end{aligned}$$

For $P = I$ ($P = \tau \cdot I$ for $\tau \in \mathbb{R}$, respectively), this results in the Richardson iteration; for $P = D = diag(A)$ this results in the Jacobi iteration; and for $P = tril(A)$, this results in the Gauss-Seidel method (GS). Note, that for Gauss-Seidel (and all related methods like "Successive Over-Relaxation," SOR [8]) every relaxation step requires the solution of a sparse triangular system.

The Jacobi method can be generalized to the block-Jacobi iteration by allowing for blocks on the diagonal of $P$. Precisely, instead of using $P = D = diag(A)$, block-Jacobi sets $P = (D_1, D_2, \ldots, D_N)$, where $D_i, i = 1 \ldots N$ are diagonal blocks of $A$. $(-N)$ then contains the elements above and below the diagonal blocks. The block-Jacobi method is well-defined if all diagonal blocks are non-singular, and the resulting preconditioner is expected to work well if the blocks succeed in reflecting the nonzero structure of the coefficient matrix $A$. Particularly if the system matrix carries no inherent block-structure, larger blocks typically result in faster convergence of block-Jacobi. At the same time, larger blocks increase the computational effort of inverting $P$, which is needed if the iteration matrix $I - P^{-1}A$ is generated explicitly. In the extreme, for a Jacobi block covering the entire system matrix, the solution $x^{(k+1)} = A^{-1}b + (I - A^{-1}A)x^{(k)}$ is readily available, but the preconditioner generation requires forming $P^{-1}$, which is the exact inverse of $A$.

The convergence of any relaxation method depends on the spectral radius of $I - P^{-1}A$:

$$\rho(I - P^{-1}A) < 1. \tag{3}$$

An efficient splitting generates a preconditioner $M := P^{-1}$, replacing the original system $Ax = b$ with a better conditioned system $MAx = Mb$ (left preconditioning), or $AMy = b$, $x = My$ (right preconditioning). In that sense, every relaxation method can be interpreted as a left-preconditioned Richardson iteration. This allows for relating the convergence of relaxation methods to the observation that an ill-conditioned preconditioner is required for significant convergence improvement.

For illustrating this connection, we consider the model problem of a $30 \times 30$ matrix $A = tridiag(-1, 2, -1)$ with $\text{cond}_2(A) = 480$. In Table 1 we list the condition number $\text{cond}_2(M)$ for the 2-norm of different preconditioners $M = (\text{diag}_B(A))^{-1}$ arising as block-Jacobi with block size $B$. We relate these preconditioners to the spectral radius of $\rho(I - MA)$ and the convergence of the left-preconditioned Richardson iteration. The Richardson iteration is started with $x^0 \equiv 0$, the right-hand-side is chosen as $b \equiv 1$, and the relative residual stopping criterion is set to $10^{-10} \parallel b \parallel$.

As expected, choosing a larger block size increases the preconditioner conditioning and improves the convergence of the outer Richardson method.

For later use we observe that Jacobi and block-Jacobi matrices $M$ satisfy the equation :

$$(I - M \cdot A)_{i,j} = 0 \text{ for } (i, j) \in \mathcal{S}, \tag{4}$$

where $\mathcal{S}$ is the sparsity pattern of the (block-) Jacobi matrix.

**Table 1**
Interpreting block-Jacobi as a preconditioned Richardson iteration.

| Preconditioner | $\text{cond}_2(M)$ | $\rho(I-MA)$ | Iterations |
|---|---|---|---|
| $M=(I)^{-1}$ | 1.00 | 2.9897 | No convergence |
| $M=(\text{diag}(A))^{-1}$ | 1.00 | 0.9949 | 4657 |
| $M=(\text{diag}_2(A))^{-1}$ | 3.00 | 0.9898 | 2402 |
| $M=(\text{diag}_3(A))^{-1}$ | 8.00 | 0.9848 | 1634 |
| $M=(\text{diag}_5(A))^{-1}$ | 18.00 | 0.9751 | 1013 |
| $M=(\text{diag}_6(A))^{-1}$ | 24.00 | 0.9704 | 857 |
| $M=(\text{diag}_{10}(A))^{-1}$ | 60.00 | 0.9535 | 550 |
| $M=(\text{diag}_{15}(A))^{-1}$ | 128.00 | 0.9375 | 409 |
| $M=(\text{diag}_{30}(A))^{-1}$ | 480.00 | 0.0000 | 1 |

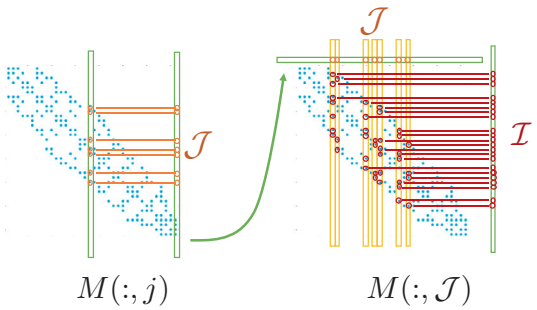


**Fig. 1.** Visualization of $\mathcal{J}$, the nonzero pattern in the $j$th column of $M$ (orange), and its shadow $\mathcal{I}$ (red) arising as $M(:,\mathcal{J})$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

### 2.3. Sparse Approximate Inverses (SAI)

In [3], Kolotilina and Yeremin considered a matrix $G$ fulfilling for given sparsity pattern $\mathcal{S}$:

$$\min_{G\in\mathcal{S}} \|GA-I\|_W = \min_{G\in\mathcal{S}} \text{trace}\left((GA-I)W(GA-I)^T\right).$$

For symmetric positive definite $A$, one may consider $W=A^{-1}$, which results in the incompleteness condition:

$$(GA=I)_{i,j}, \quad (i,j)\in\mathcal{S},$$

with $G_{i,j}=0$ for $(i,j)\notin\mathcal{S}$. Assuming that $\mathcal{J}$ is the sparsity pattern of column $A_j$, this leads to the equation:

$$G(i,\mathcal{J})A(\mathcal{J},\mathcal{J})=I(i,\mathcal{J})$$

for the $i$th row of $G$.

In the factorized case, one similarly derives the incomplete condition:

$$(G_LA)_{i,j}=(L_A)_{i,j} \; for \; (i,j)\in\mathcal{S}_L,$$

with Cholesky decomposition $A=L_AL_A^T$ and $A^{-1}\approx G_L^TG_L$, see [3]. Here, $\mathcal{S}_L$ denotes the sparsity pattern of $L$.

For general $A$, the choice of $W=I$ leads to the so called "Sparse Approximate Inverses (SAI)," minimizing the Frobenius norm for a given pattern $\mathcal{S}$:

$$\min_{M\in\mathcal{S}} \|AM-I\|_F^2 = \sum_{j=1}^{n} \min_{M_j\in\mathcal{S}} \|AM_j-I_j\|_2^2, \tag{5}$$

where $M_j$ and $I_j$ denote the $j$th column of $M$ and $I$, respectively [1].

A nice property of this approach is the minimization $\min||AM_j-I_j||_2$ for column $j$ being independent of the other columns. This allows to consider all columns simultaneously. Furthermore, the sparsity pattern of $A$ and $M$ allows to simplify the resulting Least Squares problems to small problems in the form:

$$\min_{M_j(\mathcal{J})} \|A(\mathcal{I},\mathcal{J})M_j(\mathcal{J})-I_j(\mathcal{I})\|_2^2, \tag{6}$$

where $\mathcal{J}$ is again the pattern allowed in the $j$th column of $M$, and $\mathcal{I}$ is the so-called shadow of $\mathcal{J}$, indicating the nonzero rows in $A(:,\mathcal{J})$. See Fig. 1 for an illustration of $\mathcal{J}$ and its shadow $\mathcal{I}$.

### 2.4. Incomplete factorization preconditioners

Preconditioners based on incomplete factorizations differ from SAI preconditioners in that the preconditioner matrix $M$ is not formed explicitly. Instead, the preconditioner is considered in a factorized form such that the product of the incomplete factors $L$ and $U$ fulfill $L \cdot U \approx A$.

The approximation is exact on a ILU-specific sparsity pattern $\mathcal{S}$:

$$(LU)_{i,j} = A_{i,j} \quad \forall (i,j) \in \mathcal{S}. \tag{7}$$

The generation of an incomplete factorization can be seen as a truncated Gaussian elimination process where nonzero elements or fill-in are only permitted in specified locations defined by the sparsity pattern $\mathcal{S}$. For non-singularity of the incomplete factors, this sparsity pattern must include the diagonal. The choice of $S$ can be made either before the factorization, or dynamically, during the generation of the incomplete factors. The basic method for a static sparsity pattern is the ILU(0) factorization, where nonzero elements in $L$ and $U$ are only allowed in locations that are nonzero in $A$.

The approximation quality of an incomplete factorization is limited by the sparsity pattern of the incomplete factors. One way to enhance the quality of the preconditioner is to add more elements to $S$. In level-based ILU factorizations, the locations of additional elements are constructed from a structural analysis of the original matrix [8]. A threshold-based incomplete factorization does not use a preset sparsity pattern. Instead, the algorithm generating the preconditioner decides during the factorization process whether or not to include an element in the incomplete factors. The decision typically depends on the size of the element and a certain threshold [8].

For a static sparsity pattern, the incomplete factorization can be computed by a cropped Gaussian elimination process restricting the fill-in to the pre-defined pattern, see Algorithm 1.

**Algorithm 1** Conventional ILU algorithm.

```
for i = 2 : n do
  for k = 1 : i − 1 and (i, k) ∈ S do
    a_{i,k} = a_{i,k} / a_{k,k}
    for j = k + 1 : n and (i, j) ∈ S do
      a_{i,j} = a_{i,j} − a_{i,k} a_{k,j}
    end for
  end for
end for
```

In this form, the computation of the factors $L$ and $U$ is inherently sequential. Natural parallelism only exists if it is possible to find multiple rows that only depend on rows that have already been eliminated. This strategy is known as "level-scheduling" [21,22]: A "level" consists of the unknowns that can be computed in parallel, given the dependency graph implied by the sparse matrix. Unfortunately, the number of levels is, in most cases, much smaller than the parallelism level of the hardware. Also, the level-count usually suffers from allowing more fill-in. There exist efforts to increase the parallelism with strategies like multi-color ordering or domain decomposition [11–14,23]. However, the parallelism increase often comes at the cost of reduced approximation quality [12]. Ultimately, all these approaches have limited scalability, as they typically fail to match the parallelism level provided by the current HPC architectures [24].

Recently, a new way was suggested that can generate ILU factorizations with fine-grained parallelism [15]. Chow's algorithm approximates the incomplete factors using a fixed-point iteration. The idea is to consider (7) as a set of nonlinear equations:

$$(LU - A)_{i,j} = 0 \ \forall (i,j) \in \mathcal{S}, \tag{8}$$

respectively

$$\sum_{k=1}^{\min(i,j)} l_{i,k} u_{k,j} = a_{i,j} \ \forall (i,j) \in \mathcal{S}. \tag{9}$$

With the convention of the main diagonal of the lower triangular factor $L$ being fixed to one, a fixed-point iteration of the form $x = G(x)$ can be defined, where $x$ is the vector containing the unknowns $l_{ij}$ and $u_{ij}$, which are the entries in the sparse triangular factors $L$ and $U$. For $(i,j) \in S$, the fixed-point iteration becomes:

$$l_{i,j} = \frac{1}{u_{j,j}} \left( a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} u_{k,j} \right) \quad i > j, \tag{10}$$

$$u_{i,j} = \frac{1}{l_{i,i}} \left( a_{i,j} - \sum_{k=1}^{i-1} l_{i,k} u_{k,j} \right) \quad i \leq j. \tag{11}$$

Using this formulation, the generation of the incomplete factors heavily depends on the order in which the components of the triangular factors are computed. Choosing a sequential Gaussian elimination ordering results in the classical factorization. The advantage of the fixed-point formulation is that it allows for updating multiple components in parallel, turning the generation of the incomplete factors into an iterative process. Furthermore, it can be shown that, using a suitable initial guess, the fixed-point iterations (10) and (11) converge asymptotically toward an incomplete factorization, satisfying (8) on the chosen sparsity pattern [15]. The asymptotic convergence allows for considering the distinct components in parallel, and independently of the iteration stage of other components – a property making the algorithm attractive for parallel computing.

Particularly on manycore architectures providing a high level of parallelism, the fixed-point based ILU generation is often much faster than the classical factorization [15,16]. This partly comes from the fact that the fixed-point iterations are only required to provide a solution approximation, as also converged triangular factors are only an approximation of the exact factorization. Also, when considering a sequence of shifted linear systems, the algorithm can benefit from reusing a previously computed incomplete factorization as the initial guess the fixed-point iterations are started with [17].

### 2.5. Sparse triangular solves

A consequence of not forming the preconditioner explicitly but in factorized form is that every preconditioner application requires the solution of two triangular systems

$$Ly = b, \quad Ux = y$$

with $L$ and $U$ being the sparse lower and sparse upper triangular factors, respectively. This makes the application phase of ILU preconditioners typically more expensive compared to SAI preconditioners, where the preconditioner application boils down to (sparse) matrix vector multiplications.

The traditional ILU algorithm solves the triangular systems arising from the incomplete factors via forward and backward substitution. Unfortunately, the substitutions are of inherently sequential nature, and the same level scheduling techniques [25–28] used in the incomplete factorization process fail for the same reason: The available parallelism from level-scheduling is much smaller than the hardware concurrency, allowing for only limited scalability. Hence, the efficient acceleration of sparse triangular solves in a parallel environment remains the subject of current research, see [28–31].

An attractive alternative to a parallelized exact solve is the use of an approximate solve providing a much higher concurrency level. If a solution approximation of acceptable quality can be generated with few relaxation steps, by replacing the exact triangular solves with a basic Jacobi method, then this can result in a much faster solution process [15,18,32]. As derived in Section 2.1, the challenge in this context is that for ill-conditioned systems $Ax = b$, a good ILU preconditioner will come with ill-conditioned triangular systems. This is why few relaxation steps of a scalar Jacobi may be insufficient to generate a good solution approximation. If the triangular systems carry an inherent block-structure, like common for PDE discretization problems, a block-Jacobi method can be more successful in handling high condition numbers [19].

Another approach is to replace the exact sparse triangular solves by multiplications with a sparse matrix approximating the inverse of the triangular factor (SAI) [33–35]. Unfortunately, the inverse of a sparse matrix is not generally sparse, and approximating the inverse on a preset nonzero pattern can hamper the quality of the preconditioner.

## 3. Incomplete Sparse Approximate Inverses (ISAI)

In the previous section we introduced two principles for preconditioning: the incompleteness principle related to Jacobi, ILU, and SAI for symmetric positive definite (spd) matrices, and the Frobenius norm approximation in the form of the Least Squares problem used for the SAI in (6). Comparing these two approaches, we can mix them by simplifying the SAI condition:

$$\min_{M(\mathcal{J},j)} \|A(\mathcal{I},\mathcal{J})M(\mathcal{J},j) - I(\mathcal{I},j)\|_2,$$

to an incomplete version:

$$\min_{M(\mathcal{J},j)} \|A(\mathcal{J},\mathcal{J})M(\mathcal{J},j) - I(\mathcal{J},j)\|_2,$$

or

$$A(\mathcal{J},\mathcal{J})M(\mathcal{J},j) = I(\mathcal{J},j).$$

Combining all columns $M(\mathcal{J},j)$ we obtain

$$(I - AM)_{i,j} = 0 \ \forall (i,j) \in \mathcal{S} \tag{12}$$

for any chosen pattern $\mathcal{S}$. These linear systems induced by $\mathcal{S}$, and the resulting preconditioner, are well defined only for $j \in \mathcal{J} \subseteq \mathcal{I}$, and nonsingular $A(\mathcal{J},\mathcal{J})$. The property $j \in \mathcal{J} \subseteq \mathcal{I}$ is satisfied if the diagonal entries of $A$ are nonzero. $A(\mathcal{J},\mathcal{J})$ is nonsingular for, e.g.,

- $A$ being triangular;

**Table 2**
Iteration count of the stationary iterations using block-Jacobi, SAI, and ISAI preconditioners, respectively. and density nnz of block-Jacobi (block size 1,2,3,4,5), SAI and ISAI preconditioners (pattern $|L|^k, k = 1, \ldots, 5$) for the triangular 2D Laplace matrix. Note, that to keep the number of iterations constant for the growing problem size, we have to increase the allowed pattern in the ISAI preconditioner. Also note that the density of the different preconditioners (last column) is only an approximate value for large $N$.

| | Problem size $N$ | | | | | | *nnz* in |
|---|---|---|---|---|---|---|---|
| | 10 | 20 | 30 | 40 | 50 | 60 | preconditioner |
| Jacobi (BS 1) | 18 | 38 | 58 | 78 | 98 | 118 | $N$ |
| Jacobi (BS 2) | 14 | 29 | 44 | 59 | 74 | 89 | $2N$ |
| Jacobi (BS 3) | 16 | 33 | 39 | 60 | 76 | 79 | $3N$ |
| Jacobi (BS 4) | 14 | 24 | 42 | 49 | 69 | 74 | $4N$ |
| Jacobi (BS 5) | 11 | 23 | 35 | 47 | 59 | 71 | $5N$ |
| SAI ($L$) | 25 | 44 | 62 | 80 | 97 | 114 | $3N$ |
| SAI ($L^2$) | 18 | 31 | 44 | 56 | 68 | 79 | $5.7N$ |
| SAI ($L^3$) | 14 | 25 | 34 | 44 | 53 | 62 | $9.6N$ |
| SAI ($L^4$) | 12 | 21 | 29 | 37 | 44 | 52 | $14.1N$ |
| SAI ($L^5$) | 11 | 18 | 25 | 32 | 38 | 44 | $19.5N$ |
| ISAI ($L$) | 9 | 19 | 29 | 39 | 49 | 59 | $3N$ |
| ISAI ($L^2$) | 6 | 13 | 20 | 26 | 33 | 40 | $5.7N$ |
| ISAI ($L^3$) | 5 | 10 | 15 | 20 | 25 | 30 | $9.6N$ |
| ISAI ($L^4$) | 4 | 8 | 12 | 16 | 20 | 24 | $14.1N$ |
| ISAI ($L^5$) | 3 | 7 | 10 | 13 | 17 | 20 | $19.5N$ |

- $A$ being a nonsingular M-matrix;
- $A$ being symmetric positive definite (spd); and
- $A$ being an H-matrix with a nonsingular comparison matrix.

Let us assume that the pattern of the preconditioner is given by $|A|^k$, where $|A|$ is the matrix with the absolute values of $A$ and $k$ some integer $k \geq 0$. Furthermore, we assume that all $A(\mathcal{J}, \mathcal{J})$ are well-defined nonsingular. Then, using the notation ".*" for the Hadamard product,[1] we can describe the incomplete approximation by the condition:

$$(AM - I) .* \left(|A|^k\right) = 0 \text{ for } \mathcal{S}(M) = \mathcal{S}\left(|A|^k\right).$$

For $k = 0$ this results in $\mathcal{S}(M) = \mathcal{S}(diag(A))$ and $M = diag(A)^{-1}$. This is an obvious link to the Jacobi preconditioner.

Similarly, applying this approach to a (block) diagonal pattern results in the (block-) Jacobi preconditioner. Hence, the approach (12) is a generalization of the block- Jacobi method, allowing for general patterns and computing the preconditioner columnwise. The same idea was also used by Benson and Frederickson for matrices of block banded structure called Diagonal Block Approximate Inverse (DBAI), see [2,36]. In this form the DBAI preconditioner was also considered in [4]. Considering $\mathcal{S}\left(|A|^k\right)$ for $k > 0$, the preconditioner leads to zeros of $AM - I$ in the prescribed pattern of $|A|^k$.

A first advantage of this "Incomplete Sparse Approximate Inverse (ISAI)" over the SAI preconditioner is its cheaper generation, as solving a linear system in sparse $A(\mathcal{J}, \mathcal{J})$ will typically be much cheaper than solving a least squares problem in $A(\mathcal{I}, \mathcal{J})$. Furthermore, the ISAI preconditioner can result in faster convergence, see results in Table 2 in Section 3.2. Later we will show that – in contrast to the SAI preconditioner – the ISAI preconditioner results in guaranteed convergence of the preconditioned stationary iteration by forcing zeros on the main diagonal of the iteration matrix $I - ML$, see Theorem 2.

### 3.1. ISAI for sparse triangular solves

The aim of the paper is to test the efficiency of using ISAI in the context of ILU preconditioned iterative methods such as Krylov Subspace methods (e.g., GMRES, CG, BiCGStab [8]). At every iteration of the ILU preconditioned Krylov subspace method, the preconditioner has to be applied to a vector, which is equivalent to solving two sparse triangular systems. As previously elaborated in Section 2.5, significant attention is put on developing parallel methods for efficiently generating solution approximations to sparse triangular systems coming from incomplete factorization preconditioners. In this section, we derive the ISAI preconditioner for approximating these sparse triangular solves.

If we consider the derivation of an ISAI preconditioner $M_L$ for a lower triangular matrix $L$, (12) becomes:

$$(I - LM_L)_{i,j} = 0 \ \forall (i, j) \in \mathcal{S}. \tag{13}$$

Apparently, this is very similar to the ILU property (8). A difference is that both factors, $L$ and $M_L$, are lower triangular. As $L$ is fixed, (13) poses a linear problem that can be solved directly without need of iterations. In Algorithm 2, we outline how to compute the (right-) ISAI preconditioner for a lower triangular system. An equivalent algorithm can be derived for the

[1] The notation based on the Hadamard product can be helpful in testing whether the incompleteness condition is satisfied, e.g., in MATLAB.

**Algorithm 2** Algorithm computing (right-) ISAI preconditioner $M$ for lower triangular matrix $L$ using the sparsity pattern $\mathcal{S}(M) = \mathcal{S}(|L|)$. The same algorithm can be used for computing the ISAI preconditioner for any other choice of $\mathcal{S}(M)$ (including the main diagonal), or an upper triangular matrix.

```
Choose S(M) = S(|L|)
parallel
for j = 1 : n do
    m_{j,j} = 1/l_{j,j}
    for k = j + 1 : n and k ∈ S(L(:, j)) do
        m_{k,j} = 0
        for r = j : k − 1 and r ∈ S(L(:, j)) do
            m_{k,j} = m_{k,j} − l_{k,r} m_{r,j}
        end for
        m_{k,j} = m_{k,j} / l_{k,k}
    end for
end for
```

ISAI preconditioner for an upper triangular system. We notice that beside this right-side ISAI preconditioner, it is possible to derive a left-side ISAI preconditioner by modifying (13) to:

$$(I - M_L L)_{i,j} = 0 \ \forall (i,j) \in \mathcal{S}. \tag{14}$$

In the remainder of the paper, however, we mainly focus on the right-side ISAI preconditioner.

Like in the ILU case, the ISAI preconditioner quality depends on the chosen nonzero pattern $\mathcal{S}$. But it also determines the cost of the ISAI generation, and the communication pattern in the preconditioner application. A simple strategy for enhancing the quality of the ISAI preconditioner is to allow for a higher number of nonzeros in the preconditioner matrix $M_L$, e.g., by pre-setting the sparsity pattern to $\mathcal{S}\left(|L|^k\right)$ for some $k > 1$.

We emphasize that the flexibility in choosing the sparsity structure also allows for optimization in a larger context: optimizing the communication pattern of the preconditioner to a certain subproblem structure is of particular interest in distributed memory settings and communication-avoiding Krylov solvers based on the matrix powers kernel [37,38]. For the ISAI preconditioner, this can be realized by dropping locations at the intersection of subproblems, or replacing them with locations that are more attractive in terms of communication. An interesting approach is to use the sparsity pattern $\mathcal{S}\left(|L|^k\right)$ for some $k > 1$, and to drop all nonzeros on subproblem intersections that are not included in $\mathcal{S}(|L|)$. We refrain in this work from investigating this idea, but leave it for future research.

For convenience, we use the notation "ISAI ($L^k$)" to denote the ISAI preconditioner using the sparsity pattern $\mathcal{S}\left(|L|^k\right)$.

As previously discussed, a specific choice of the nonzero pattern $\mathcal{S}$ also links the ISAI strategy to the (block-) Jacobi methods:

**Remark 1.** For $\mathcal{S}$ being a (block) diagonal structure, the resulting ISAI matrix becomes the (block-) Jacobi matrix.

**Remark 2.** The preconditioner is well-defined if the sparsity structure $\mathcal{S}$ includes the main diagonal. This is the case when choosing $\mathcal{S}$ consistent with the structure of $|L|^k$ for some $k \geq 0$, or a (block-) Jacobi pattern.

**Remark 3.** Note that Algorithm 2 can be seen as a collection of incomplete approximate triangular solves of the form $LM_j = e_j$. Precisely, for a preconditioner matrix containing $n$ columns, it requires the solution of $n$ small triangular systems in the restricted space $\mathcal{S}$.

This aspect makes the ISAI preconditioner attractive for highly parallel computing architectures. In Section 4 we will elaborate on how the ISAI generation can be realized efficiently using batched routines on manycore accelerators.

Having computed the ISAI triangular preconditioner $M_L$ for $L$ and $M_U$ for $U$, we can replace the ILU left-preconditioned system

$$(L \cdot U)^{-1} A x = (L \cdot U)^{-1} b \Leftrightarrow U^{-1} L^{-1} A x = U^{-1} L^{-1} b$$

by

$$M_U M_L A x = M_U M_L b$$

since $M_L \approx L^{-1}$ and $M_U \approx U^{-1}$.

This way, we can realize the preconditioner application to a vector $z$, $v = U^{-1} L^{-1} z$ in terms of sparse matrix vector multiplications with $M_L$ and $M_U$, $v = M_U \cdot M_L \cdot z$. In the remainder of the paper, we use the notation "SpMV ISAI" to indicate that we approximate the solution of a linear system $LUv = z$ through multiplying the right-hand side with the ISAI preconditioner $v = M_U M_L z$.

A sparse matrix vector multiplication with the ISAI preconditioner matrix offers much more parallelism than an ILU preconditioner using exact triangular solves, but the loss in preconditioner quality may result in a need for more outer

solver iterations. Then an alternative is to keep the incomplete factors for $L$ and $U$ and solve the sparse triangular systems in the preconditioner application phase. Precisely, we can solve $Ly = z$, via stationary iterations with some initial guess $y^{(0)}$ in the form:

$$y^{(s+1)} = z + (I - LM_L)y^{(s)}. \tag{15}$$

In the remainder of the paper we call this strategy for applying an ISAI preconditioner matrix "ISAI relaxation steps", and use the right-hand side $z$ as initial guess for $y^{(0)}$.

It is also possible to form the product $M := M_U \cdot M_L$, and apply the preconditioner in terms of a single (sparse) matrix vector multiplication. This, however, makes using the ISAI preconditioner in terms of stationary iterations impossible, and so we refrain from investigating the potential of this strategy.

Choosing the sparsity pattern $\mathcal{S}$ to be $|L|^k$ for $k \geq 0$, or a (block-) Jacobi pattern, the ISAI preconditioner is well-defined, and the corresponding stationary iterations fulfill:

**Theorem 2.** *Let $L$ be a lower triangular matrix, $\mathcal{S} = |L|^k$ for some $k \geq 0$. Then, the stationary iterations (15) using the ISAI preconditioner matrix converge in the asymptotic sense.*

**Proof.** Obviously, since $L$ and $M_L$ are lower triangular, the iteration matrix $I - LM_L$ is lower triangular as well. Given a sparsity pattern $\mathcal{S}$ that includes the diagonal, we get from (12) that the ISAI left-sided preconditioner particularly fulfills:

$$(I - LM_L)_{i,i} = 0 \ \forall \ i \in 1 \ldots n.$$

This implies that the iteration matrix $I - LM_L$ is a strictly lower triangular matrix. For the spectral radius of the componentwise positive iteration matrix we get:

$$\rho(|I - LM_L|) = 0.$$

This is a sufficient condition for convergence in the asymptotic sense [39]. □

**Remark 4.** The convergence for the left-side ISAI preconditioner can be proven equivalently.

**Remark 5.** Applying the ISAI preconditioner to general problems, the iteration matrix $I - AM$ will also be zero on the main diagonal, but there is no direct connection to the spectral radius, or convergence in this case. Heuristically, one could argue, that choosing a thick sparsity pattern covering the most relevant locations, the entries not covered will be small, and thus the spectral radius will also be less than 1.

For choosing $\mathcal{S}$ consistent with the sparsity pattern of $|L|^k$ for some $k \geq 1$, we can expect fast convergence of the stationary iterations (15) solving $Lx = b$. In particular, choosing those patterns in the ISAI preconditioner may allow for faster convergence than using a (block-) Jacobi pattern. The reason is that the inverse relation (4) is enforced on the nonzero pattern on $L$, while (block-) Jacobi is concentrated on main diagonal blocks.

**Remark 6.** Note, that for a general sparse matrix $A$ here we assume a given factorized preconditioner, e.g. $A \approx LU$, and use the ISAI method for approximating and replacing the triangular factors $L^{-1}$ and $U^{-1}$ in the preconditioner for $A$ by triangular matrices $L_M$ and $U_M$, e.g. via $(L_M L = I)_S$. In contrast, factorized sparse approximate inverses like FSAI [3] aim directly for triangular factors $L_G$ and $U_G$ of $A^{-1}$, e.g. via $(L_G A = L_A)_S$ for exact LU factorization $A = L_A U_A$. Similarly, in SAI, a preconditioner for spd $A$ is determined directly e.g. via $(GA = I)_S$. Technically, we use the same condition, but for the triangular factors and not for $A$ itself.

In [40] it is shown that a block-Jacobi preconditioner can be superior to a scalar Jacobi preconditioner with respect to handling ill-conditioned problems, particularly if the problem itself contains some block-structure. Obviously, when choosing the sparsity pattern $\mathcal{S}(|L|)$, the ISAI matrix preserves these blocks in the nonzero structure. Consequently, as long as the block size used in block-Jacobi does not introduce a high number of additional nonzeros, the ISAI matrix can also be expected to be a superior preconditioner.

### 3.2. Illustrating the ISAI preconditioner using example problems

We now use two small examples for illustrating the ISAI preconditioner generation and its efficiency. The examples arise as the lower triangular matrices related to the constant coefficient 1D and 2D Laplacian to stencils $[-1, 2, -1]$ and

$$\begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix},$$

respectively. For the 1D case, we consider $L = tridiag(-1, 1, 0)$ with the exact inverse:

$$L^{-1} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 1 & \cdots & 1 & 1 \end{pmatrix}.$$

We denote the nonzero structure of the $j$th column of $L$, with $\mathcal{J}$. To compute the ISAI preconditioner with pattern $L$, we have to solve the linear systems:

$$L(\mathcal{J}, \mathcal{J})M_j(\mathcal{J}) = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} M_j(\mathcal{J}) = e_j(\mathcal{J}) \tag{16}$$

for the columns $j = 1 \ldots n-1$, and the system $M_n(\mathcal{J}) = e_n(\mathcal{J})$ for column $n$. The ISAI preconditioner $M_L$ for the sparsity pattern $\mathcal{S}(|L|)$ arises as

$$M_L = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 1 & \ddots & \ddots & & \vdots \\ 0 & \ddots & & \ddots & \vdots \\ \vdots & \ddots & \ddots & & 0 \\ 0 & \cdots & 0 & 1 & 1 \end{pmatrix}.$$

It is easy to see how (16) grows for a thicker sparsity pattern $\mathcal{S}\left(|L|^k\right)$, $k > 1$, and how the ISAI preconditioners for larger $k$ arise as lower triangular band matrices of ones with a band width of $k+1$:

$$M_L = \begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & \ddots & & & & \vdots \\ 1 & & \ddots & & & \vdots \\ 0 & \ddots & & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & \cdots & 1 \end{pmatrix}. \tag{17}$$

If we use the arising ISAI preconditioner for right-preconditioned stationary Richardson iterations, the iteration matrix becomes:

$$I - LM_L = \begin{pmatrix} 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & \ddots & & & & & \vdots \\ 0 & & \ddots & & & & \vdots \\ -1 & \ddots & & \ddots & & & \vdots \\ 0 & \ddots & \ddots & & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 0 & \cdots & 0 \end{pmatrix},$$

with a zero diagonal band of width $k+1$. Choosing a thicker sparsity pattern (larger values for $k$) will accelerate the convergence of the stationary iteration. Ultimately, for this problem, the ISAI preconditioner $M_L$ becomes the exact inverse of $L$ for $k = n-1$.

In comparison, each column of the SAI preconditioner for this problem is obtained by solving the least squares problem

$$min \left\| \begin{pmatrix} 1 & 0 \\ -1 & 1 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right\|^2.$$

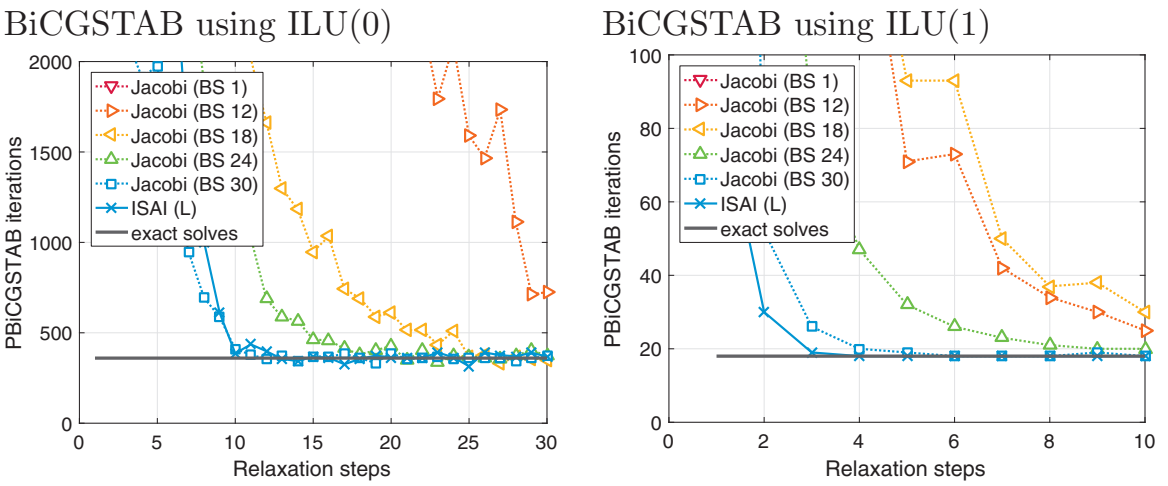


**Fig. 2.** BiCGSTAB convergence using ILU preconditioning in combination with exact triangular solves or relaxation steps. The test problem is BCS.

This results in the SAI preconditioner

$$M_{SAI} = \frac{1}{3}\begin{pmatrix} 2 & 0 & \cdots & \cdots & 0 \\ 1 & \ddots & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 & 2 \end{pmatrix}.$$

We observe that $M_{SAI}$ is well-conditioned, while the ISAI matrix given in (17) is ill-conditioned, and hence expected to be the more efficient preconditioner.

For the 2D case we consider the block matrix $L_2 = L \otimes I + I \otimes L$ with the 1D matrix $L$ from above and the first column given by $(2, -1, 0, \ldots, 0, -1, 0, \ldots, 0)^T$. To compute the ISAI preconditioner with pattern $\mathcal{S}(|L_2|)$, we have to solve the linear systems (or subsystems):

$$L(\mathcal{J}, \mathcal{J})M_j(\mathcal{J}) = \begin{pmatrix} 2 & 0 & 0 \\ -1 & 2 & 0 \\ -1 & 0 & 2 \end{pmatrix} M_j(\mathcal{J}) = e_j(\mathcal{J}).$$

The related SAI Least Squares problem has three additional rows related to the entries in $\mathcal{I}\backslash\mathcal{J}$ containing four times the entry $-1$. Again, the systems grow with the choice of $k$.

We report in Table 2 the number of relaxation steps needed by a stationary iteration to solve a linear system $Lx = b$ of size $N$. The right-hand side is chosen randomly; the iterations are started with $x^0 \equiv 0$; and the relative residual stopping criterion is chosen as $10^{-6}|b|$. The preconditioner used for the stationary iterations is either block-Jacobi, SAI, or ISAI, all considering different nonzero patterns. In Table 2, we also report the number of nonzeros ($nnz$) in the different preconditioner matrices. For block-Jacobi, we estimate the nonzero count as the product of block size and the number of blocks. Note that different strategies exist for realizing block-Jacobi preconditioning [40]: Instead of generating the explicit block-inverse in the preconditioner setup, it is also possible to only factorize the diagonal blocks, or even keep the (sparse) diagonal blocks of the original matrix and solve the small triangular block systems in each preconditioner application. The last strategy can help in keeping a low nonzero count. However, it is rarely considered in practice because it dramatically increases the computational cost of every preconditioner application.

The results in Table 2 reveal that for comparable nonzero counts, iterating with the ISAI preconditioner is always better than iterating with the SAI preconditioner or a block-Jacobi matrix. For higher nonzero counts, the ISAI preconditioner becomes very accurate, quickly reducing the number of necessary relaxation steps.

The quick convergence of the stationary iterations using an ISAI matrix can also be observed in Fig. 2. Here, we analyze the iteration count of an ILU-preconditioned BiCGSTAB solver using different kinds of triangular solves: exact triangular solves vs. a fixed number of relaxation steps of a stationary method. For the latter, we consider the block-Jacobi and the ISAI preconditioner. The left side shows the results for a level-0 ILU factorization ( ILU(0) ); the right side shows the results for a level-1 ILU ( ILU(1) ). The BCS test matrix carries an inherent block structure with blocks of size 8. It has a condition number of $\text{cond}_2(\text{BCS}) = 1.32 \cdot 10^6$. More details about BCS can be found in Table 3.

**Table 3**
Test matrices.

| Name | Abbr. | Nonzeros $n_z$ | Size $n$ | Description |
|---|---|---|---|---|
| AF_SHELL3 | AF3 | 17,562,051 | 504,855 | 3D FEM discretization, struct. |
| BCSSTK10 | BCS | 22,070 | 1086 | Stiffness matrix, structural |
| ECOLOGY2 | ECO | 4,995,991 | 999,999 | Circuit theory, animal/gene flow |
| OFFSHORE | OFF | 4,242,673 | 259,789 | 3D FEM, transient electric diff. |
| PARABOLIC_FEM | PAR | 3,674,625 | 525,825 | CFD problem |
| TMT_UNSYM | TMT | 4,584,801 | 917,825 | Electromagnetics problem |

ILU preconditioning is very efficient for this problem. The condition numbers of the incomplete factors are $\text{cond}_2(L) = 8.68 \cdot 10^4$ and $\text{cond}_2(U) = 1.09 \cdot 10^6$ for the ILU(0) preconditioner, and $\text{cond}_2(L) = 2.68 \cdot 10^2$ and $\text{cond}_2(U) = 2.56 \cdot 10^4$ for the ILU(1) preconditioner.

A plain Jacobi fails to handle the ill-conditioned factors within a reasonable range of relaxation steps. The block-Jacobi works better, and larger block sizes are more successful. For the ISAI preconditioner, 10 relaxation steps are sufficient to match the exact solves in the ILU(0) case. For ILU(1), only 3 ISAI relaxation steps are needed. In comparison, the same preconditioner quality requires a block size of 30 for the Jacobi solver.

## 4. Parallel ISAI preconditioner generation

As previously discussed, the ISAI preconditioner generation via Algorithm 2 breaks down into the solution of a set of small triangular linear problems. Precisely, each (nonzero) column of the preconditioner matrix $M$ arises as the solution of one (typically small) triangular system. Consequently, for a well-defined preconditioner, $n$ linear systems need to be solved, with $n$ being the size of the original system matrix $A$. The sizes of the small linear systems correlate with the number of nonzeros in the distinct columns of the chosen nonzero pattern $\mathcal{S}$. This implies that choosing a denser sparsity pattern $\mathcal{S}$ results in larger systems that have to be solved in the preconditioner generation.

For triangular systems $L$ and $U$, the arising linear systems also have triangular structure. For problems coming from PDE discretizations, the size of these systems is typically much smaller than the original matrix. Hence, their solution can efficiently be realized via forward and backward substitution, respectively.

For a lower triangular matrix $L$, the computation of the entries in each column $i$ of $M$ requires the following steps:

1. $\mathcal{J} = find(M(:, i))$.
   Collect all nonzero locations in the $i$th column of the pre-defined sparsity structure $\mathcal{S}(M)$.
2. **Generate** $L(\mathcal{J}, \mathcal{J})$.
   Generate the small system matrix by extracting the respective entries of the target matrix, in this case, the lower incomplete factor $L$.
3. **Solve** $L(\mathcal{J}, \mathcal{J}) \cdot M(\mathcal{J}, i) = I(\mathcal{J}, i)$.
   Solve the arising small (lower) triangular system.
3. **Insert solution** $M(\mathcal{J}, i)$ **into preconditioner matrix** $M$.
   Back-insert the computed solution into the sparse structure of the preconditioner matrix $M$.

All columns of the preconditioner matrix $M$ can be computed in parallel. Furthermore, the ISAI generation can be realized in terms of "batched routines." These types of routines are attractive when addressing a large set of small problems on architectures with a high concurrency level [20].

Fig. 3 visualizes the generation of the ISAI preconditioner $M$ for a lower triangular factor.

In Section 5, we evaluate the numerical properties and the performance of the ISAI preconditioner on a GPU architecture. For this architecture, we design batched routines for the distinct building blocks listed above: the nonzero locator, the small-system generator, the triangular solves, and the back-insertion. To enable the efficient parallel execution, we preset the memory layout such that the distinct systems – although of different size – are all generated in uniform memory layout with consistent stride, see Fig. 3. This approach implies that none of the arising small systems may exceed a certain upper bound. In our implementation, this upper bound is chosen consistent with the warp size of 32. As a result, no ISAI preconditioner can be generated for a sparsity pattern containing more than 32 elements in one column. Although this implementation-specific upper bound may appear very restrictive, the experiments reveal that this setting allows for covering a good portion of the matrices available at the SuiteSparse Matrix Collection [41].

The efficient solution of the triangular systems is realized by using the batched trsv routine presented in [42].

## 5. Numerical experiments

In this section, we experimentally evaluate the numerical properties of the ISAI preconditioner in the context of approximate ILU preconditioning, and assess the efficiency of the ISAI generation based on algorithm-specific batched routines for a GPU hardware setting. First, we introduce the hardware architecture, then analyze numerical and performance-related

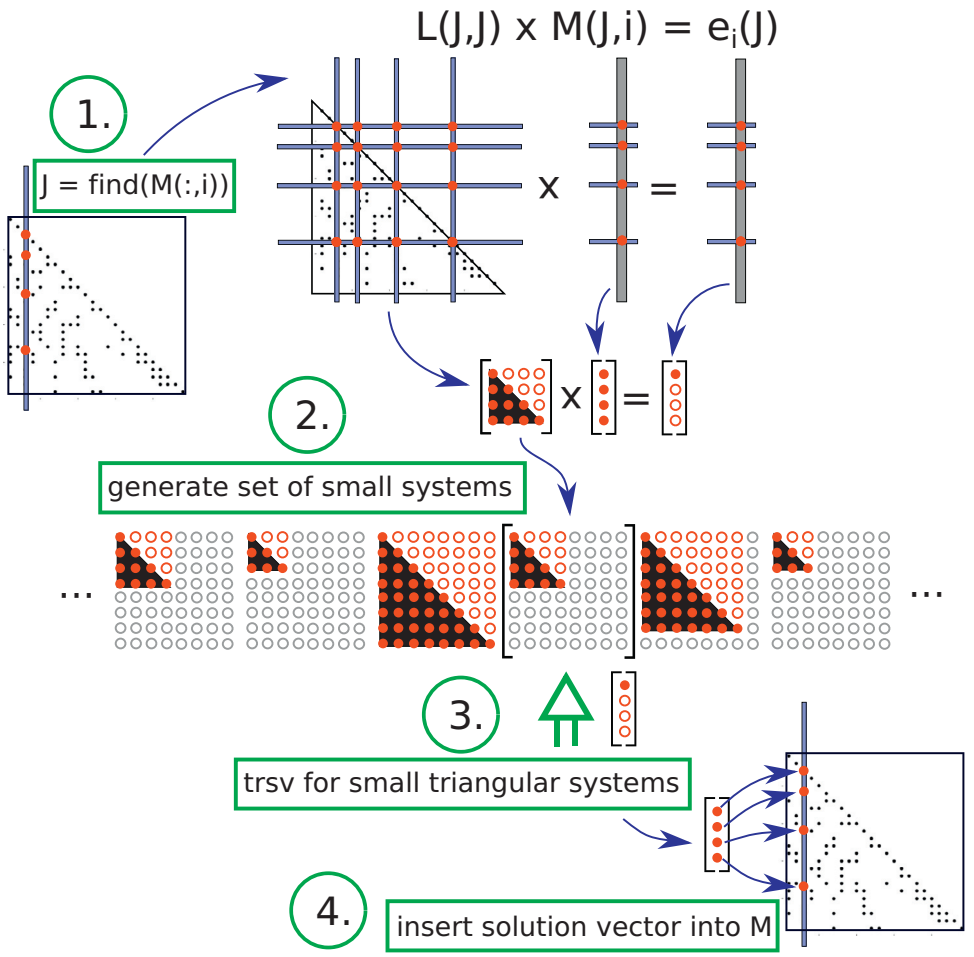


**Fig. 3.** Generation of the ISAI preconditioner matrix $M$ for a lower sparse triangular system $L$ via batched routines. The sparsity structure of the ISAI matrix is chosen to be consistent with the sparsity structure of the lower triangular system $L$.

aspects of the ISAI implementation for selected test matrices, and finally assess the preconditioner's efficiency in a study covering a large number of test matrices.

### 5.1. Experiment setup

Our experimental setup is an NVIDIA Tesla K80 GPU, which is composed of two Tesla GK210 processors [43]. Combined, the two processors have a memory bandwidth of 480GB/s, 24GB of main memory, and a theoretical peak performance of 2.91 TFLOPS (double precision). The kernels generating the ISAI preconditioner are implemented in CUDA version 7.5 [44] and use a default thread block size of 32. All other functionalities, including the outer BiCGSTAB solver and the generation of the ILU factors, are taken from the MAGMA-sparse open-source software library [45,46]. The exact sparse triangular solve routines are from the NVIDIA cuSPARSE library [47]. If we use block-Jacobi relaxation steps to generate approximations to the sparse triangular systems coming from the incomplete LU factors, we generate the block-inverse matrix explicitly using the batched Gauss-Jordan Elimination presented in [40]. The relaxation steps based on block-Jacobi are then composed of matrix-vector multiplications. All computations use double-precision arithmetic.

The test matrices listed in Table 3 are taken from the SuiteSparse Matrix Collection [41]. We use Reverse Cuthill-McKee (RCM) ordering for all test matrices, as this ordering helps in producing accurate incomplete factorization preconditioners [48,49]. After reordering, we symmetrically scale the matrices to have a unit diagonal. In Fig. 4 we visualize the nonzero pattern of selected test cases. We note that some of the matrices are symmetric and positive definite (spd). This would allow us to replace the ILU with an incomplete Cholesky and to choose a Conjugate Gradient (CG [8]) as the outer solver. In Section 5.3 we evaluate the efficiency of the ISAI preconditioner for a large set of general matrices. To be consistent, we decide to ignore the spd information and handle all systems with a robust combination of BiCGSTAB and ILU.

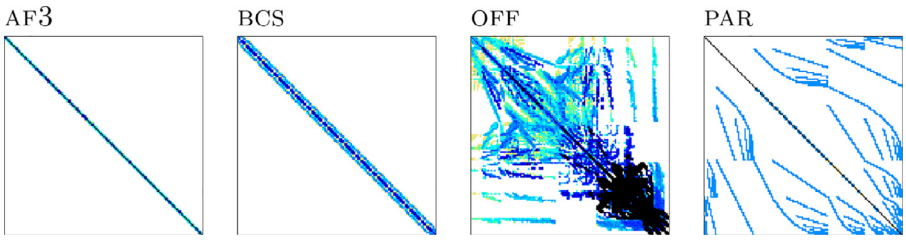


**Fig. 4.** Sparsity structure of selected test matrices before reordering.

**Table 4**
Comparing BiCGSTAB for different preconditioners based on ILU(0). We report outer iterations (ITER), preconditioner setup time (PST), iterative solver runtime (IST), and overall execution time (OET). "0 relaxation steps" corresponds to the use of the preconditioner matrix as sparse approximate inverse without stationary iterations. The results are for the TMT problem.

| | ITER | PST | IST | OET |
|---|---|---|---|---|
| No preconditioner | 17,145 | 0.00 | 42.17 | 42.17 |
| ILU + exact solves | 1490 | 0.29 | 98.23 | 98.53 |
| ILU + Jacobi (BS 1) | | | | |
| 0 relaxation steps | 10,541 | 0.35 | 44.77 | 45.12 |
| 1 relaxation steps | 3223 | 0.35 | 33.65 | 34.00 |
| 2 relaxation steps | 2165 | 0.35 | 32.55 | 32.90 |
| 3 relaxation steps | 1655 | 0.35 | 32.47 | 32.82 |
| ILU + Jacobi (BS 4) | | | | |
| 0 relaxation steps | 8756 | 0.50 | 48.13 | 48.63 |
| 1 relaxation steps | 3033 | 0.50 | 42.94 | 43.43 |
| 2 relaxation steps | 2113 | 0.50 | 42.21 | 42.71 |
| 3 relaxation steps | 1965 | 0.50 | 50.65 | 51.16 |
| ILU + Jacobi (BS 8) | | | | |
| 0 relaxation steps | 7494 | 0.67 | 45.43 | 46.09 |
| 1 relaxation steps | 3077 | 0.67 | 48.72 | 49.39 |
| 2 relaxation steps | 2143 | 0.67 | 47.64 | 48.31 |
| 3 relaxation steps | 1844 | 0.67 | 52.75 | 53.43 |
| ILU + ISAI ($L$) | | | | |
| 0 relaxation steps | 2712 | 0.33 | 12.51 | 12.84 |
| 1 relaxation steps | 1750 | 0.33 | 20.04 | 20.37 |
| 2 relaxation steps | 1501 | 0.33 | 24.64 | 24.97 |
| 3 relaxation steps | 1544 | 0.33 | 33.04 | 33.37 |
| ILU + ISAI ($L^2$) | | | | |
| 0 relaxation steps | 2673 | 0.46 | 15.43 | 15.88 |
| 1 relaxation steps | 1871 | 0.46 | 27.94 | 28.40 |
| 2 relaxation steps | 1543 | 0.46 | 32.42 | 32.88 |
| 3 relaxation steps | 1513 | 0.46 | 40.96 | 41.41 |
| ILU + ISAI ($L^3$) | | | | |
| 0 relaxation steps | 1719 | 0.66 | 12.13 | 12.79 |
| 1 relaxation steps | 1589 | 0.66 | 30.22 | 30.88 |
| 2 relaxation steps | 1504 | 0.66 | 39.83 | 40.49 |
| 3 relaxation steps | 1657 | 0.66 | 56.25 | 56.92 |

### *5.2. Detailed evaluation for selected problems*

First, we perform a very detailed evaluation of the unsymmetric electromagnetics problem TMT, see Table 3 and Fig. 4 for details.

In Table 4 we compare the convergence and performance of a BiCGSTAB iterative solver preconditioned with different versions of an ILU(0) preconditioning: exact triangular solves (using NVIDIA's cuSPARSE routines exploiting level-scheduling); approximate triangular solves based on (block-) Jacobi; and approximate triangular solves based on the ISAI preconditioner. All approximate triangular solves are either realized in the form of relaxation steps according to (15), or by considering the preconditioner matrix as sparse approximate inverse (labeled with "0 relaxation steps").

We note that ILU(0) preconditioning combined with exact triangular solves is very efficient for this problem because it significantly reduces the iteration count of the BiCGSTAB solver. However, despite the significantly higher iteration count, the non-preconditioned BiCGSTAB converges faster (w.r.t runtime) than the ILU(0)-preconditioned (exact trsv) configuration.

As expected, combining the ILU(0) factorization with approximate triangular solves, we typically need more BiCGSTAB iterations. At the same time, the approximate triangular solves are much faster than the exact triangular solves. For (block-) Jacobi, more relaxation steps in each triangular solve reduce the outer iteration count, and increasing the size of the Jacobi blocks can also help in this respect. In the end, all considered configurations using Jacobi-based triangular solves are faster than the traditionally used exact solves.

The Incomplete Sparse Approximate Inverse preconditioner is already capable of providing a competitive preconditioner when handling the triangular systems with a single ISAI preconditioner multiplication. As one sparse matrix vector product is much faster than forward and backward substitutions, the overall iterative solver runtime(IST) is only a fraction of the iterative solver runtime using exact triangular solves, see the fourth column of Table 4. The generation of the ISAI preconditioner introduces some overhead, (preconditioner setup time PST, see the third column in Table 4). But the implementation introduced in Section 4 proves to be very efficient: the ISAI overhead over a plain ILU(0) is only about 14% when considering the sparsity structure $\mathcal{S}(|L|)$. Naturally, this overhead grows with an increasing nonzero count in the ISAI matrix, but a denser ISAI preconditioner also improves the preconditioner quality. Using the sparsity structure $\mathcal{S}\left(|L|^3\right)$, the quality of the ISAI preconditioner matrix is competitive to an exact triangular solve. Realizing the ISAI preconditioner for stationary iterations (according to (15)) reduces the outer iteration count further, but it also makes the preconditioner application more expensive. Ultimately, the overall execution time (OET) accumulates the preconditioner setup time and the iterative solver runtime. For the TMT problem, the best choice is to generate an ISAI preconditioner matrix based on the sparsity pattern of $\mathcal{S}\left(|L|^3\right)$ and to handle the sparse triangular solves in terms of a single sparse approximate inverse multiplication. This configuration is about 7.7 $\times$ faster than the ILU(0) preconditioner using exact triangular solves, and 3.3 $\times$ faster than the non-preconditioned BiCGSTAB.

Next, we consider problems coming from different scientific applications, see Table 3. In Fig. 5 we analyze the iteration count (on the left side) and the overall solver execution time, accumulating preconditioner setup time, and iterative solver execution time (on the right side). We compare with exact and approximate triangular solves in an ILU(0)-preconditioned BiCGSTAB.

Ignoring some rounding-related outliers, using approximate triangular solves usually results in a need for additional iterations of the outer BiCGSTAB solver. Independent of how the approximate triangular solves are realized, this iteration overhead decreases with increasing relaxation step count. Using block-Jacobi steps, larger block sizes are not always beneficial. Also, using a block-Jacobi preconditioner in the form of a sparse approximate inverse multiplication (see results for "0 relaxation steps") can destroy the outer solver convergence. ISAI is in most cases the better preconditioner. Using the ISAI preconditioner for relaxation steps, the outer iteration count quickly approaches the setting based on exact triangular solves. At the same time, the ISAI preconditioner application is much faster than the sparse triangular solves, which in most cases compensates for the outer iteration overhead — compare the left and the right side of Fig. 5. In the performance metric, the ISAI preconditioner strategy is always better than the Jacobi-based triangular solves.

For the AF3 and OFF problems, the ISAI implementation presented in Section 4 fails for the sparsity structure $\mathcal{S}\left(|L|^2|\right)$. The reason is that the small triangular systems that need to be solved in the preconditioner generation exceed the implementation-specific upper bound of 32 elements per column in the nonzero pattern $\mathcal{S}$. Besides using a different nonzero pattern, an obvious work-around would be to modify the implementation such that larger systems can be handled. This, however, increases the memory requirement, and using larger thread block sizes in the GPU kernels would result in a performance loss. Also, increasing the upper bound to a limit that allows for handling these problems does not ensure success for other problems. Ultimately, it is important to realize that limitations exist in the capability of the ISAI strategy: generating the ISAI preconditioner for the sparsity pattern of matrices containing a dense column, which is not uncommon in circuit simulation problems [50], would require solving a triangular systems of full size. Obviously, this is not practical. Hence, these types of systems remain outside the scope of ISAI preconditioning. Fortunately, most problems coming from discretized PDEs have a balanced nonzero pattern, and incomplete factorizations with moderate fill-in typically have fewer than 32 elements in the columns of the triangular factors.

### 5.3. ISAI efficiency study

To assess the practicability of the proposed implementation and the efficiency using ISAI preconditioning for the sparse triangular solves in ILU(0) preconditioning, we consider a set of test matrices from the SuiteSparse Matrix Collection. The goal of this effort is also to identify a preconditioner configuration that works well for a large number of problems. Precisely, we test different preconditioning techniques for all matrices available at the UFMC that fulfill:

1. The matrix is real-valued;
2. The matrix is square;
3. The matrix contains less than 100,000,000 nonzeros;
4. A BiCGSTAB solver preconditioned with an ILU(0) with exact triangular solves converges within 10,000 iterations.

We obtain a test suite containing 316 matrices, and in the following analysis, we always compare 2 preconditioner configurations in the time-to-solution metric. We quantify which configuration is faster and for how many problems, and how many problems can only be handled by one and not the other configuration. We note that in this experiment we consider

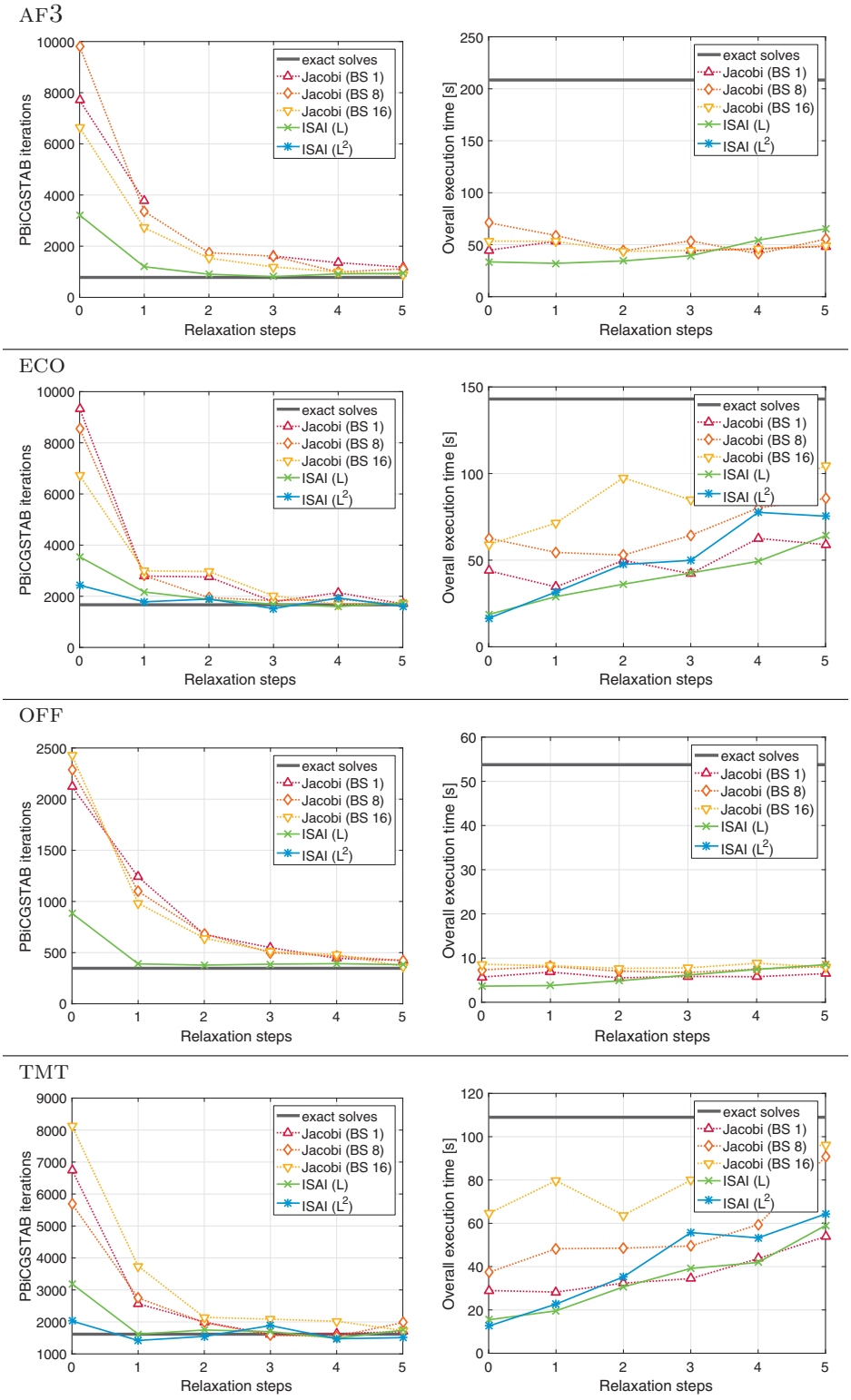


**Fig. 5.** BiCGSTAB convergence using ILU(0) preconditioning in combination with exact triangular solves or relaxation steps.

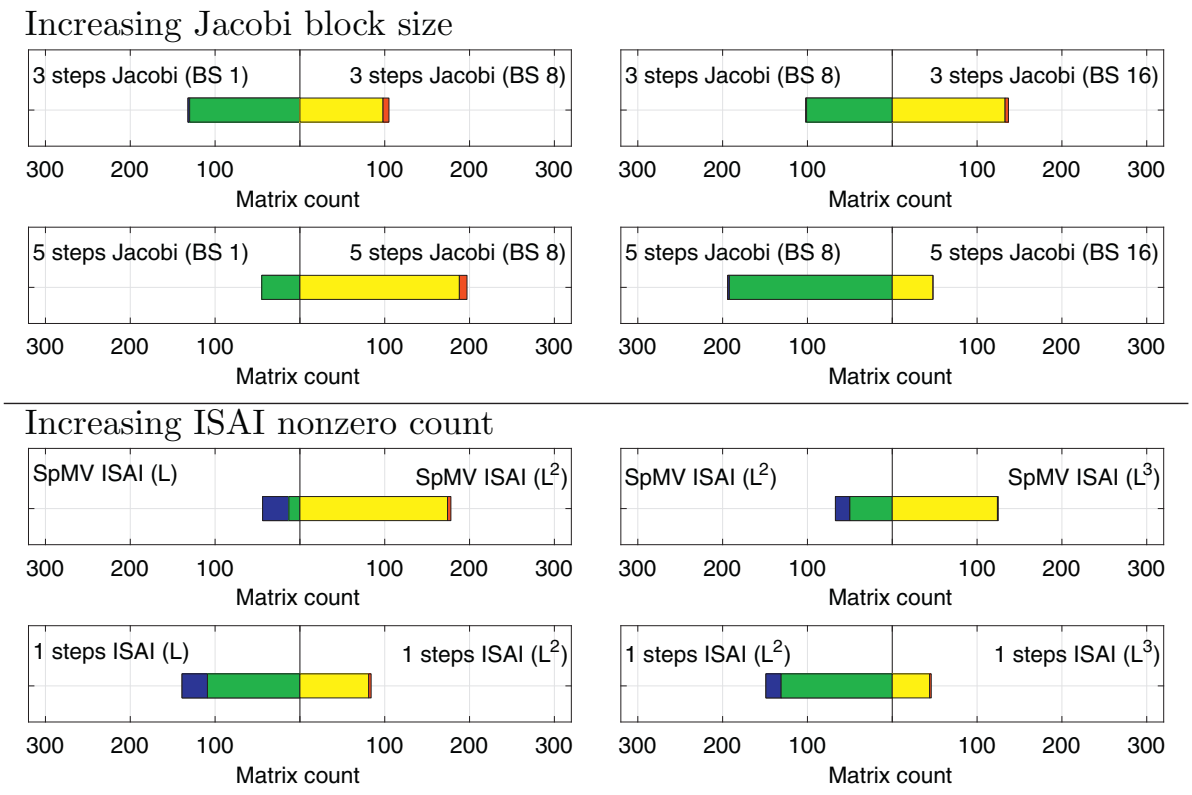


**Fig. 6.** Analysis: increasing the block size in Jacobi and nonzero count in ISAI, respectively. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

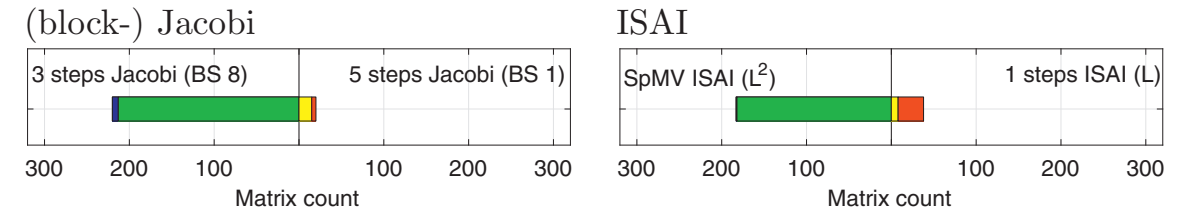


**Fig. 7.** Analysis: thinner sparsity pattern with more relaxation steps vs. thicker sparsity pattern and fewer relaxation steps. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

all matrices in the ordering in which they are available at the UFMC webpage [41]. The analysis presented here tries to include the most important aspects of the more comprehensive interactive comparison available at:
<http://www.icl.utk.edu/~hanzt/precond_comparison/>.
In Figs. 6–10 we use bar-plots to visualize this evaluation:

- The (left-most) blue part indicates the number of problems that can be handled by the left configuration only, the right configuration fails.
- Conversely, the (right-most) red part indicates the number of problems that can be handled by the right configuration only, the left configuration fails
- The green part indicates the number of problems that can be handled by both configurations, but the left configuration is faster.
- The yellow part indicates the number of problems that can be handled by both configurations, but the right configuration is faster.

We always use a BiCGSTAB outer solver, and the incomplete factors are generated using the ILU(0) factorization of NVIDIA's cuSPARSE library. The evaluation is based on total solver execution time, including the preconditioner setup time.

Obviously, identifying a good preconditioner configuration is a multi-parameter optimization problem. We first look into the sparsity pattern. In the upper part of Fig. 6, we increase the Jacobi block size from 1 to 8 and from 8 to 16. The results

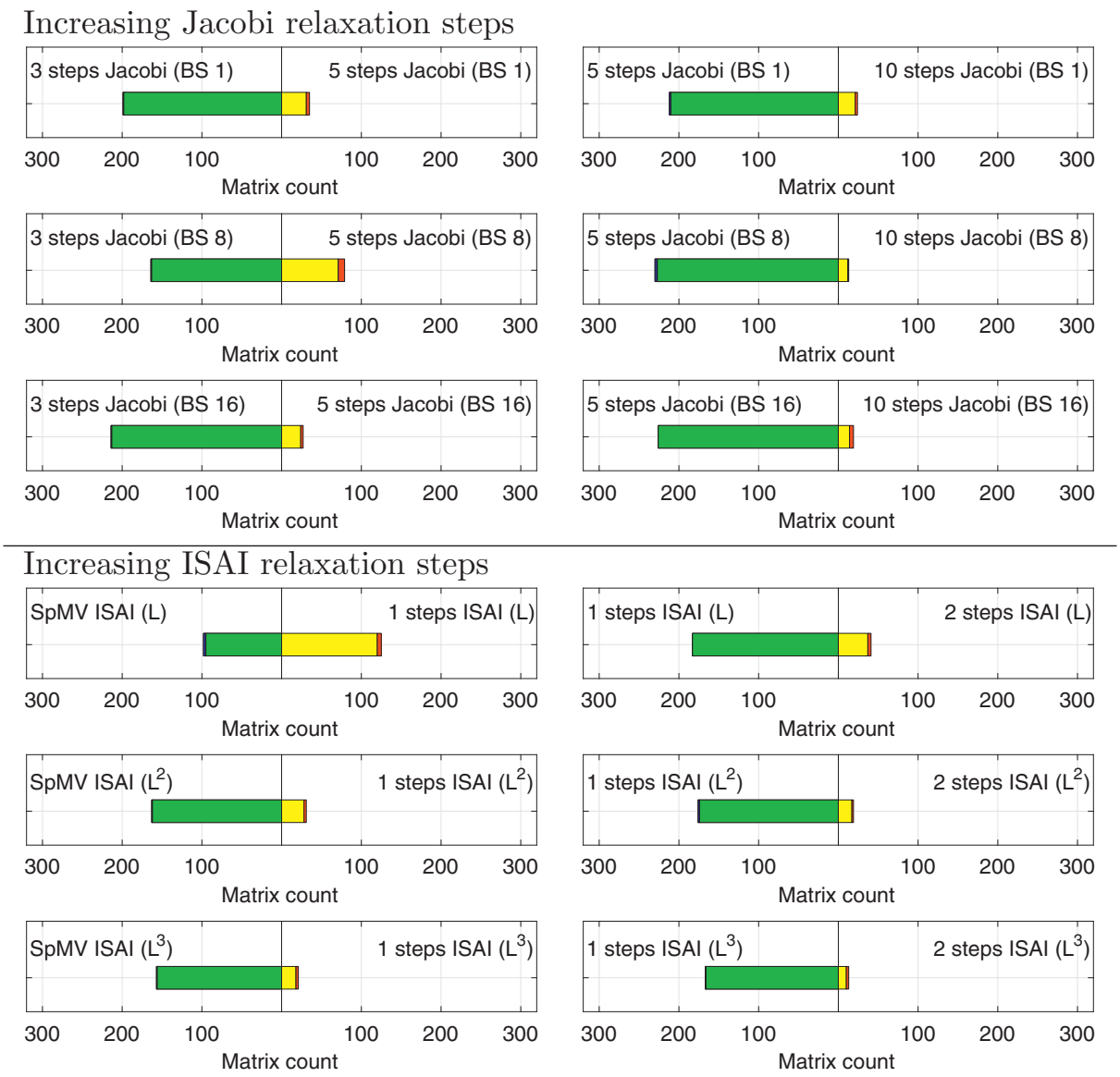


**Fig. 8.** Analysis: increasing the number of relaxation steps of (block-) Jacobi and ISAI, respectively. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

in the first row are for using 3 relaxation steps in every preconditioner application, the results in the second row are for 5 relaxation steps.

As expected, there are matrices where triangular solves based on a plain Jacobi fail to provide an efficient preconditioner. For problems that can be handled with either configuration, larger Jacobi blocks require a more expensive preconditioner setup phase. Nevertheless, larger Jacobi blocks can improve the time-to-solution performance. The results in the second row of Fig. 6 indicate that a block size of 8 might be a good choice when using 5 Jacobi steps.

The lower part of Fig. 6 visualizes a similar analysis for the ISAI preconditioner. Instead of increasing the Jacobi block size, the nonzero count in the ISAI matrix is increased by considering the sparsity pattern $\mathcal{S}(|L|)$, $\mathcal{S}\left(|L|^2\right)$, and $\mathcal{S}\left(|L|^3\right)$. The third row of Fig. 6 shows the results for using the ISAI preconditioner in the SpMV ISAI setting, the fourth row shows the results for one ISAI relaxation step. As previously observed for the AF3 problem, using a denser sparsity pattern can make the ISAI generation fail. This is reflected in the blue bars on the left, representing problems where only using the thinner sparsity pattern allows for the ISAI generation. The SpMV ISAI utilization suggests that using a thicker sparsity pattern could be beneficial (see the third row of Fig. 6). Conversely, the stationary usage suggests a thinner sparsity pattern would be better (see the fourth row of Fig. 6). This shows that the preconditioner quality may either come from a more accurate sparse inverse approximation, or the stationary iterations considering the residuum contributions.

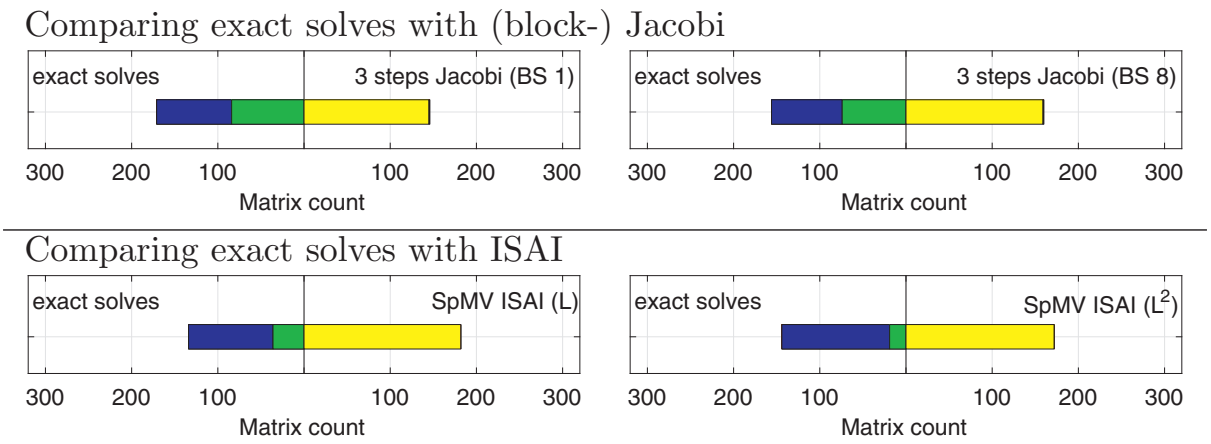


**Fig. 9.** Analysis: comparing exact triangular solves with approximate triangular solves. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

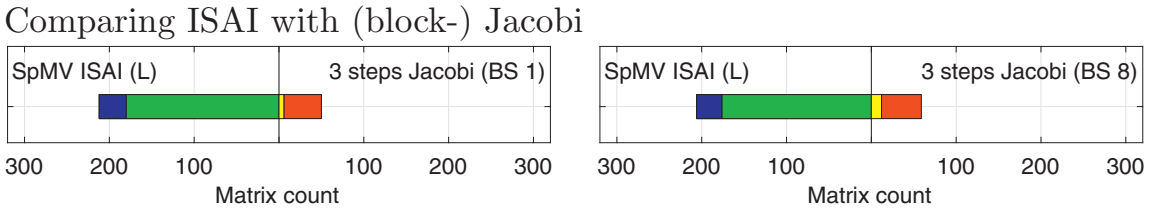


**Fig. 10.** Analysis: comparing the ISAI preconditioner with (block-) Jacobi. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

The remaining question of whether the combination of a thicker sparsity pattern with fewer relaxation steps should be preferred over a thinner sparsity pattern with more relaxation steps is answered in Fig. 7: for both preconditioner types, fewer steps and a thicker preconditioner matrix turn out to be faster.

In Fig. 8, we now fix the nonzero count in the preconditioner matrices, but analyze the effect of increasing the number of relaxation steps, instead. Using Jacobi blocks of size 8, some problems that can be handled only when using more than 3 relaxation steps. For the sparsity pattern $\mathcal{S}(|L|)$, one fixed point sweep is, for most problems, faster than the SpMV ISAI preconditioner. All other results suggest that fewer relaxation steps are faster.

We now turn to a comparison against exact triangular solves. Note that the success of exact triangular solves was used as a criterion to identify the matrices contained in the test suite. In Fig. 9, we compare the exact solves with Jacobi (first row) and ISAI (second row), respectively. Using 3 relaxation steps of a (block-) Jacobi fails in about one-fourth of the problems (see results in first row of Fig. 9). For the problems where a (block-) Jacobi works, it is often faster.

Using the sparsity pattern $\mathcal{S}(|L|)$ and $\mathcal{S}\left(|L^2|\right)$, the ISAI preconditioner fails for 25% and 30%, respectively. This can have two causes: the implementation-specific limitations do not allow for the preconditioner generation (see Section 4); or the preconditioner quality is not sufficient. A detailed analysis on this issue reveals that it is typically the generation of the preconditioner that fails, see Table 5. If the ISAI preconditioner generation succeeds, the ISAI-based triangular solves are for almost all problems the better choice, see the results in the second row of Fig. 9.

Having identified approximate triangular solves being faster for many problems, in Fig. 10 we finally compare the Jacobi-based approach with the ISAI-based approach. Again, the implementation-specific limitations thwart the ISAI utilization for some problems. For other problems, the preconditioner quality of the (block-) Jacobi turns out to be insufficient. Considering the cases where both configurations work, the ISAI preconditioner is almost exclusively better.

Finally, we also include a BiCGSTAB preconditioned with a simple Jacobi diagonal scaling preconditioner in the comparison to justify the use of an incomplete-factorization-based preconditioner. Jacobi can be realized very efficiently on manycore architectures like GPUs. The statistics in Fig. 11 visualize for how many problems a certain preconditioner is the fastest choice. Aside from the Jacobi-preconditioned BiCGSTAB (green bar on the left), we include an ILU(0)-preconditioned version using exact triangular solves, and different combinations where the ILU(0) factors are solved in approximate fashion: Jacobi with different block sizes using 3 (blue), 5 (green), or 10 (yellow) relaxation steps; SpMV ISAI (blue), or ISAI using 1 (green) or 2 (yellow) relaxation steps.

Obviously, there are problems where BiCGSTAB with a Jacobi diagonal preconditioner is the overall winner. For the majority of the problems however, incomplete factorization preconditioning is better. Some problems require combining the ILU(0) with exact triangular solves. For those problems, the iterations using level-based exact triangular solves are faster, or

**Table 5**
Number of problems where ISAI succeeds, fails due to implementation-specific limitations, or fails due to insufficient preconditioner quality.

| Configuration | Success | Generation fails | Insufficient quality |
|---|---|---|---|
| $\mathcal{S}(\|L\|)$ | | | |
| 0 relaxation steps | 218 | 67 | 31 |
| 1 relaxation steps | 220 | 67 | 29 |
| 2 relaxation steps | 224 | 67 | 25 |
| $\mathcal{S}(\|L^2\|)$ | | | |
| 0 relaxation steps | 191 | 106 | 19 |
| 1 relaxation steps | 193 | 106 | 17 |
| 2 relaxation steps | 193 | 106 | 17 |
| $\mathcal{S}(\|L^3\|)$ | | | |
| 0 relaxation steps | 175 | 125 | 16 |
| 1 relaxation steps | 177 | 125 | 14 |
| 2 relaxation steps | 179 | 125 | 12 |

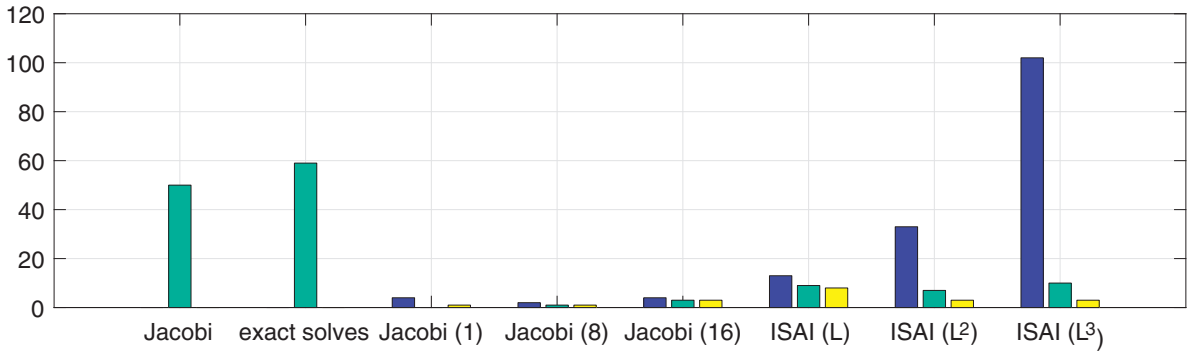


**Fig. 11.** Performance comparison between a Jacobi-preconditioned BiCGSTAB (left) and ILU(0)-preconditioned versions using different preconditioner configurations: 3,5,10 relaxation steps of Jacobi (Block); or the ISAI preconditioner as sparse approximate inverse, or 1,2 relaxation steps. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

the approximate triangular solves are unable to handle the sparse triangular systems. This can be due to ill-conditioning of the triangular factors, or the failing ISAI generation. We previously observed that the ISAI preconditioner is typically superior to the (block-) Jacobi approach. Fig. 11 reveals that a thick sparsity pattern in combination with the SpMV ISAI concept is the performance winner for most of the problems.

We conclude from this analysis, that the ISAI-based approximate triangular solves work well for many problems. In particular, if the matrix characteristics allow, it should always be preferred to Jacobi-based approach. For production code, a preliminary analysis could be used to identify the thickest sparsity pattern possible, as this can be expected to provide the best time-to-solution performance. A preliminary analysis is cheap as it boils down to identifying the largest number of nonzeros accumulated in one column.

## 6. Summary and future research

In this paper, we have proposed a new Incomplete Sparse Approximate Inverse (ISAI) preconditioner, arising as a generalization of (block-) Jacobi methods. We have shown that this preconditioner can be generated with a parallel algorithm, efficiently exploiting the manycore technology. We have shown that the ISAI preconditioner is an attractive alternative to exact triangular solves in the context of incomplete factorization preconditioning. In particular for problems coming from PDE discretization with a balanced nonzero distribution, the ISAI preconditioner can be generated quickly for the incomplete factors, and allows for efficient preconditioner generation. Compared with (block-) Jacobi preconditioners, the ISAI preconditioners do not require matrix reordering and block size optimization as they inherently comprise the nonzero pattern of the system matrix. This allows the generic handling of ill-conditioned triangular systems that are expected when addressing hard problems with ILU preconditioning.

In the future, we will investigate the ISAI efficiency for general problems outside the ILU setting. For those, optimizing the ISAI nonzero pattern for fast convergence may be required. The flexibility in choosing the sparsity structure will also be explored in the context of communication-avoiding Krylov methods that require a certain communication pattern for efficient preconditioner utilization.

## Acknowledgments

This material is based upon work supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Number DE-SC0016513. This research was also supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. The authors would also like to acknowledge the Swiss National Computing Centre (CSCS) for granting computing resources in the Small Development Project entitled "Energy-Efficient preconditioning for iterative linear solvers" (#d65). The authors would like to thank Edmond Chow from Georgia Tech for comments on an earlier version of the manuscript. Also the unknown reviewers provided us in a very diligent revision process with a comprehensive list constructive comments for which we are in particular thankful.

## References

[1] M.J. Grote, T. Huckle, Parallel preconditioning with sparse approximate inverses, SIAM J. Sci. Comput. 18 (3) (1997) 838–853.
[2] M.W. Benson, P.O. Frederickson, Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems, Utilitas Math. 22 (1982) 127–140.
[3] L.Y. Kolotilina, A.Y. Yeremin, Factorized sparse approximate inverse preconditionings i. theory, SIAM J. Matrix Anal. Appl. 14 (1) (1993) 45–58.
[4] E. Chow, Y. Saad, Approximate inverse techniques for block-partitioned matrices, SIAM J. Sci. Comput. 18 (6) (1997) 1657–1675.
[5] E. Chow, Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns, Int. J. High Perform. Comput. Appl. 15 (1) (2001) 56–74.
[6] T. Huckle, A. Kallischko, Frobenius norm minimization and probing for preconditioning, Int. J. Comput. Math. 84 (8) (2007) 1225–1248.
[7] T. Huckle, M. Sedlacek, Smoothing and regularization with modified sparse approximate inverses, J. Image Video Process. 2010 (2010) 3:1–3:16.
[8] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, Philadelphia, PA, USA, 2003.
[9] O. Axelsson, Iterative Solution Methods, Cambridge University Press, New York, NY, USA, 1994.
[10] J.A. Meijerink, H.A.V. der Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric *m*-matrix, Math. Comput. 31 (1977) 148–622.
[11] E.L. Poole, J.M. Ortega, Multicolor ICCG methods for vector computers, SIAM J. Numer. Anal. 24 (1987) 1394–1417.
[12] D. Lukarski, Parallel Sparse Linear Algebra for Multi-Core and Many-Core Platforms – Parallel Solvers and Preconditioners, Karlsruhe Institute of Technology, 2012 Ph.D. thesis.
[13] M. Benzi, W.D. Joubert, G. Mateescu, Numerical experiments with parallel orderings for ILU preconditioners, Electron. Trans. Numer. Anal. 8 (1999) 88–114.
[14] S. Doi, On parallelism and convergence of incomplete LU factorizations, Appl. Numer. Math. 7 (5) (1991) 417–436.
[15] E. Chow, A. Patel, Fine-grained parallel incomplete LU factorization, SIAM J. Sci. Comput. 37 (2015) C169–C193.
[16] E. Chow, H. Anzt, J. Dongarra, Asynchronous iterative algorithm for computing incomplete factorizations on GPUs, in: Lecture Notes in Computer Science, vol. 9137, 2015, pp. 1–16.
[17] H. Anzt, E. Chow, J. Saak, J. Dongarra, Updating incomplete factorization preconditioners for model order reduction, Numer. Algorithms 73 (3) (2016) 611–630.
[18] H. Anzt, E. Chow, J. Dongarra, Iterative sparse triangular solves for preconditioning, in: J.L. Träff, S. Hunold, F. Versaci (Eds.), Euro-Par 2015: Parallel Processing, Lecture Notes in Computer Science, vol. 9233, Springer Berlin Heidelberg, 2015, pp. 650–661.
[19] E. Chow, J. Scott, On the Use of Iterative Methods and Blocking for Solving Sparse Triangular Systems in Incomplete Factorization Preconditioning, Technical Report RAL-P-2016-006, Rutherford Appleton Laboratory, 2016.
[20] A. Abdelfattah, A. Haidar, S. Tomov, J. Dongarra, Performance tuning and optimization techniques of fixed and variable size batched cholesky factorization on GPUs, Procedia Comput. Sci. 80 (2016) 119–130. International Conference on Computational Science 2016, ICCS 2016, 6–8 June 2016, San Diego, California, USA.
[21] E.R. Bank, C. Wagner, Multilevel ILU decomposition, Numer. Math. 82 (4) (1999) 543–576.
[22] Y. Saad, Multilevel ILU with reorderings for diagonal dominance, SIAM J. Sci. Comput. 27 (3) (2005) 1032–1057.
[23] D. Hysom, A. Pothen, A scalable parallel algorithm for incomplete factor preconditioning, SIAM J. Sci. Comput. 22 (6) (2001) 2194–2215.
[24] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R.S. Williams, K. Yelick, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Keckler, D. Klein, P. Kogge, R.S. Williams, K. Yelick, Exascale computing study: technology challenges in achieving exascale systems, 2008, (DARPA IPTO ExaScale Computing Study).
[25] E.C. Anderson, Y. Saad, Solving sparse triangular systems on parallel computers, Int. J. High Speed Comput. 1 (1989) 73–96.
[26] J.H. Saltz, Aggregation methods for solving sparse triangular systems on multiprocessors, SIAM J. Sci. Stat. Comput. 11 (1990) 123–144.
[27] S.W. Hammond, R. Schreiber, Efficient ICCG on a shared memory multiprocessor, Intl. J. High Speed Comput. 4 (1992) 1–21.
[28] M.M. Wolf, M.A. Heroux, E.G. Boman, Factors impacting performance of multithreaded sparse triangular solve, in: Proceedings of the 9th International Conference on High Performance Computing for Computational Science, VECPAR'10, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 32–44.
[29] J. Park, M. Smelyanskiy, N. Sundaram, P. Dubey, Sparsifying synchronization for high-performance shared-memory sparse triangular solver, in: Supercomputing, in: Lecture Notes in Computer Science, vol. 8488, 2014, pp. 124–140.
[30] J. Mayer, Parallel algorithms for solving linear systems with sparse triangular matrices, Computing 86 (4) (2009) 291–312.
[31] M. Naumov, Parallel Solution of Sparse Triangular Linear Systems in the Preconditioned Iterative Methods on the GPU, Technical Report NVR-2011-001, NVIDIA, 2011.
[32] H. Anzt, E. Chow, D.B. Szyld, J. Dongarra, Domain Overlap for Iterative Sparse Triangular Solves on GPUs, Springer International Publishing, Cham, pp. 527–545.
[33] A.C.N.V. Duin, Scalable parallel preconditioning with the sparse approximate inverse of triangular matrices, SIAM J. Matrix Anal. Appl. 20 (1996) 987–1006.
[34] M. Benzi, M. Tůma, A comparative study of sparse approximate inverse preconditioners, Appl. Numer. Math. 30 (2–3) (1999) 305–340.
[35] A.C.N. van Duin, Scalable parallel preconditioning with the sparse approximate inverse of triangular matrices, SIAM J. Matrix Anal. Appl. 20 (4) (1999) 987–1006.
[36] M.W. Benson, Iterative Solution of Large Scale Linear Systems, Lakehead University, Thunder Bay, 1973 Master's thesis.
[37] I. Yamazaki, H. Anzt, S. Tomov, M. Hoemmen, J. Dongarra, Improving the performance of CA-GMRES on multicores with multiple GPUs, 28th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2014), 2014.
[38] L. Grigori, S. Moufawad, Communication avoiding ILU0 preconditioner, SIAM J. Sci. Comput. 37 (2) (2015).
[39] A. Frommer, D.B. Szyld, On asynchronous iterations, J. Comput. Appl. Math. 123 (1–2) (2000) 201–216.
[40] H. Anzt, J. Dongarra, G. Flegar, E.S. Quintana-Ortí, Batched gauss-jordan elimination for block-jacobi preconditioner generation on gpus, in: Proceedings of the 8th International Workshop on Programming Models and Applications for Multicores and Manycores, PMAM'17, ACM, New York, NY, USA, 2017, pp. 1–10.

[41] T.A. Davis, University of Florida sparse matrix collection, NA-Digest 92 (1994).
[42] J. Kurzak, H. Anzt, M. Gates, J. Dongarra, Implementation and tuning of batched cholesky factorization and solve for NVIDIA GPUs, IEEE Trans. Parallel Distrib. Syst. (2015). 1045-9219.
[43] TESLA K80 GPU Active Accelerator, NVIDIA Corporation, BD-07317-001_v05 edition, 2015.
[44] NVIDIA Corp., CUDA C Programming Guide, v7.5, 2015.
[45] H. Anzt, J. Dongarra, M. Kreutzer, M. Koehler, Efficiency of general Krylov methods on GPUs – an experimental study, The Sixth International Workshop on Accelerators and Hybrid Exascale Systems (AsHES), 2016.
[46] Innovative Computing Lab, Software distribution of MAGMA version 2.0, 2016, (http://icl.cs.utk.edu/magma/).
[47] NVIDIA Corp., cuSPARSE Library, v7.5, 2015.
[48] I.S. Duff, G.A. Meurant, The effect of ordering on preconditioned conjugate gradients, BIT 29 (4) (1989) 635–657.
[49] M. Benzi, D.B. Szyld, A. van Duin, Orderings for incomplete factorization preconditionings of nonsymmetric problems, SIAM J. Sci. Comput. 20 (1999) 1652–1670.
[50] T.A. Davis, E.P. Natarajan, Algorithm 907: Klu, a direct sparse solver for circuit simulation problems, ACM Trans. Math. Softw. 37 (3) (2010).