An Algebraic Multilevel Preconditioner for Symmetric Positive Definite and Indefinite Problems

Arnold Reusken

Institut für Geometrie und Praktische Mathematik, RWTH Aachen,
Templergraben 55, D-52056 Aachen, Germany

Abstract. We present a preconditioning method for the iterative solution of large sparse systems of equations. The preconditioner is based on ideas both from ILU preconditioning and from multigrid. The resulting preconditioning technique requires the matrix only. A multilevel structure is obtained by constructing a maximal independent set of the graph of a reduced matrix. The computation of a Schur complement approximation is based on a Galerkin approach with a matrix dependent prolongation and restriction. The resulting preconditioner has a transparant modular structure similar to the algorithmic structure of a multigrid V-cycle. The method is applied to symmetric positive definite and indefinite Helmholtz problems. The multilevel preconditioner is compared with standard ILU preconditioning methods.

1 Introduction

Multigrid methods are very efficient iterative solvers for the large systems of equations resulting from discretization of partial differential equations (cf. [11,26] and the references therein). An important principle of multigrid is that a basic iterative method, which yields appropriate local corrections, is applied on a hierarchy of discretizations with different characteristic mesh sizes. This multilevel structure is of main importance for the efficiency of multigrid.
Another class of efficient iterative solvers consists of Krylov subspace methods combined with ILU preconditioning (cf. [8,20] and the references therein). These methods only need the matrix and are in general easier to implement than multigrid methods. Also the Krylov subspace methods are better suitable for a "black-box" approach. On the other hand, for discretized partial differential equations the Krylov methods with ILU preconditioning are often less efficient than multigrid methods.
In the multigrid field there have been developed methods which have a multilevel structure but require only the matrix of the linear system. These are called *algebraic* multigrid methods. Approaches towards algebraic multigrid are presented in, e.g. [7,9,19,23,25]. In all these methods one tries to mimic the multigrid principle. First one introduces a reasonable coarse "grid" space. Then a prolongation operator is chosen and for the restriction one usually takes the adjoint of the prolongation. The operator on the coarse grid space is defined by a Galerkin approach. With these components, a standard multigrid approach (smoothing + coarse grid correction) is applied. These algebraic multigrid methods can be used in situations where a grid (hierarchy) is not available. Also these methods can be used for developing black-box solvers.
Recently there have been developed ILU type of preconditioners with a multilevel structure, cf. [5,6,16,21,22]. The multilevel structure is induced by a level wise numbering of the unknowns. In [2,3,17,18], new hybrid methods have been presented, which use ideas both from ILU (incomplete Gaussian elimination) and from multigrid.

In the present paper we reconsider the approximate cyclic reduction preconditioner of [17,18]. This method is based on the recursive application of a two-level method, as in cyclic reduction or in a multigrid V-cycle method. For the definition of a two level structure we use two important concepts: a reduced graph and a maximal independent set. The partitioning of the set of unknowns, denoted by the labels "red " and "black", yields a corresponding block-representation of the given matrix $\mathbf{A}$:

$$\mathbf{PAP}^T = \begin{bmatrix} \mathbf{A}_{bb} & \mathbf{A}_{br} \\ & \\ \mathbf{A}_{rb} & \mathbf{A}_{rr} \end{bmatrix} , \tag{1}$$

with $\mathbf{P}$ a suitable permutation matrix. The construction of the red-black partitioning is such that, under reasonable assumptions on $\mathbf{A}$, the $\mathbf{A}_{rr}$ block is guaranteed to be strongly diagonally dominant. In [18] one can find a technique for constructing a sparse approximation $\tilde{\mathbf{S}}_{bb}$ of the Schur complement $\mathbf{S}_{bb} := \mathbf{A}_{bb} - \mathbf{A}_{br}\mathbf{A}_{rr}^{-1}\mathbf{A}_{rb}$. This approximation is obtained by replacing the *block* Gaussian elimination which results in the Schur complement (cf. (4)) by a sequence of *point* Gaussian elimination steps.

In [18] the approximate cyclic reduction preconditioner is presented and analyzed in a general framework and applied to convection-diffusion and anisotropic diffusion problems. In the present paper we explain a simple variant of the cyclic reduction preconditioner which is then applied to a discretization of the Helmholtz equation $-\Delta u - \lambda u = f$ ($\lambda \geq 0$ a constant) on the unit square. In Sect. 4 we consider $\lambda = 0$ (Poisson equation) and $\lambda = 19.73$ (SPD, close to singular) and compare CG + approximate cyclic reduction preconditioning with the standard ICCG method. In Sect. 5 we consider the indefinite case ($\lambda = 100$, $\lambda = 200$) and compare the GMRES(5)+ ILU preconditioning (using droptolerances) with GMRES(5) + approximate cyclic reduction preconditioning.

2 The Cyclic Reduction Principle

We recall the classical method of cyclic reduction. This method can be used, for example, for solving a linear system with a tridiagonal matrix or with a special block tridiagonal matrix (cf. [10,13,24]). We explain the cyclic reduction principle by considering an $n \times n$ linear system with a tridiagonal matrix:

$$\mathbf{Ax} = \mathbf{b}, \quad \mathbf{A} = \begin{bmatrix} a_1 & b_1 & & & \\ c_1 & a_2 & b_2 & & \emptyset \\ & \ddots & \ddots & \ddots & \\ \emptyset & & \ddots & \ddots & b_{n-1} \\ & & & c_{n-1} & a_n \end{bmatrix} , \quad a_i \neq 0 \quad \text{for all} \quad i \ . \tag{2}$$

Reordering the unknowns based on an obvious red-black (or "odd-even") structure results in a permuted system with a matrix of the form

$$\mathbf{PAP}^T = \begin{bmatrix} \mathbf{A}_{bb} & \mathbf{A}_{br} \\ & \\ \mathbf{A}_{rb} & \mathbf{A}_{rr} \end{bmatrix} , \tag{3}$$

in which $[\mathbf{A}_{bb} \;\; \mathbf{A}_{br}]$ represents the equations in the unknowns with a black label and $[\mathbf{A}_{rb} \;\; \mathbf{A}_{rr}]$ represents the equations in the unknowns with a red label. Note that, because $\mathbf{A}$ is tridiagonal, the diagonal blocks $\mathbf{A}_{bb}, \mathbf{A}_{rr}$ are diagonal matrices. Gaussian elimination in the red points results in a reduced system with dimension (approximately) $\frac{1}{2}n$. In matrix notation this corresponds to block UL-decomposition:

$$\mathbf{PAP}^T = \begin{bmatrix} \mathbf{I} & \mathbf{A}_{br}\mathbf{A}_{rr}^{-1} \\ & \\ \emptyset & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{S}_{bb} & \emptyset \\ & \\ \mathbf{A}_{rb} & \mathbf{A}_{rr} \end{bmatrix} , \quad \mathbf{S}_{bb} := \mathbf{A}_{bb} - \mathbf{A}_{br}\mathbf{A}_{rr}^{-1}\mathbf{A}_{rb} \ . \tag{4}$$

The reduced system has a matrix $\mathbf{S}_{bb}$ (Schur complement) which is tridiagonal, and thus the same approach can be applied to $\mathbf{S}_{bb}$. So the basic cyclic reduction idea is to reduce significantly the dimension of the problem repeatedly until one has a relatively small problem that can be solved easily. After this *decomposition phase* a block UL-decomposition of the matrix $\mathbf{A}$ is available and the linear system in (2) can be solved using a simple backward–forward substitution process. In this process the systems with matrix $\mathbf{A}_{rr}$ are trivial to solve, because $\mathbf{A}_{rr}$ is diagonal.

In Sect. 3 we will modify this simple direct method, resulting in a *preconditioner* for sparse matrices which are not necessarily tridiagonal. For a better understanding of this preconditioner we first give a rather detailed description of a particular implementation of the cyclic reduction method for a tridiagonal matrix, which consists of a decomposition phase and a solution phase.

Decomposition phase. We assume a tridiagonal matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$. *Dimbound*, with $1 <$ *Dimbound* $\ll n$ is a given integer (used in D5 below). Set $i := 1$, $\mathbf{A}_1 := \mathbf{A}$, $m_0 := n$.

D1. *Red-black partitioning* . Given the tridiagonal matrix $\mathbf{A}_i$ we construct a red-black (odd-even) partitioning of the unknowns. This results in n_i vertices with label red and m_i vertices with label black. Note: $m_i + n_i = m_{i-1}$.

D2. *Determine permutation.* We determine a symmetric permutation $p_i : \{1, 2, ..., m_{i-1}\} \rightarrow \{1, 2, ..., m_{i-1}\}$ such that applying this permutation to the index set of the unknowns results in an ordering in which all unknowns with label red have index $j \in (m_i, m_{i-1}]$ and all unknowns with label black have index $j \in [1, m_i]$. Note that since we only have to permute between the sets $\{j \mid j > m_i$ and $\text{label}(j) = \text{black}\}$ and $\{j \mid j \leq m_i$ and $\text{label}(j) = \text{red}\}$, such a permutation can be fully characterized by a permutation $\hat{p}_i : \{m_i + 1, m_i + 2, ..., m_{i-1}\} \rightarrow \{1, 2, ..., m_i\}$.

D3. *Determine permuted matrix.* The symmetric matrix corresponding to the permutation p_i of D2 is denoted by $\mathbf{P}_i$. We determine $\mathbf{P}_i\mathbf{A}_i\mathbf{P}_i$. This matrix has a 2×2-block representation:

$$\mathbf{P}_i\mathbf{A}_i\mathbf{P}_i = \begin{bmatrix} \mathbf{A}_i^{bb} & \mathbf{A}_i^{br} \\ \mathbf{A}_i^{rb} & \mathbf{A}_i^{rr} \end{bmatrix} , \tag{5}$$

with $\mathbf{A}_i^{rr} \in \mathbb{R}^{n_i\times n_i}$, $\mathbf{A}_i^{bb} \in \mathbb{R}^{m_i\times m_i}$, $\mathbf{A}_i^{rb} \in \mathbb{R}^{n_i\times m_i}$, $\mathbf{A}_i^{br} \in \mathbb{R}^{m_i\times n_i}$.

D4. *Compute Schur complement.* Compute the Schur complement $\mathbf{A}_{i+1} := \mathbf{P}_i\mathbf{A}_i\mathbf{P}_i/\mathbf{A}_i^{rr} := \mathbf{A}_i^{bb} - \mathbf{A}_i^{br}(\mathbf{A}_i^{rr})^{-1}\mathbf{A}_i^{rb}$.

D5. *Store.* Save $m_i, \hat{p}_i, \mathbf{A}_i^{rr}, \mathbf{A}_i^{rb}, \mathbf{A}_i^{br}$. If $m_i <$ *Dimbound* then save $\mathbf{A}_{i+1}$ (stop the reduction process) else $i := i + 1$ and goto D1.

Remark 1. The algorithm used in the decomposition phase is well-defined iff the diagonal matrices $\mathbf{A}_i^{rr}$ are nonsingular. The latter property holds if the matrix $\mathbf{A}$ is symmetric positive definite or an M-matrix. This follows from the fact that the Schur complement of an SPD-matrix (M-matrix) is an SPD-matrix (M-matrix), cf. [12].

If the above decomposition process stops with $i = i_{\max}$, we obtain integers $m_1 > m_2 > ... > m_{i_{\max}}$, permutation vectors $\hat{p}_i$ $(1 \leq i \leq i_{\max})$, sparse matrices $\mathbf{A}_i^{rr}, \mathbf{A}_i^{rb}, \mathbf{A}_i^{br}$ $(1 \leq i \leq i_{\max})$ and the approximate Schur complement on the highest level $\mathbf{A}_{i_{\max}+1}$. We use the following terminology: $\hat{p}_i$ is called the *permutation* operator on level i, $\mathbf{A}_i^{rr}$ is called the *solve* operator on level i, $\mathbf{A}_i^{rb}$ is called the *collect* operator on level i, $\mathbf{A}_i^{br}$ is called the *distribute* operator on level i.

The red unknowns on all levels, together with the black unknowns on the final level induce a direct sum decomposition $\mathrm{I\!R}^n = \mathrm{I\!R}^{n_1} \oplus \mathrm{I\!R}^{n_2} \oplus \ldots \oplus \mathrm{I\!R}^{n_{i_{\max}}} \oplus \mathrm{I\!R}^{m_{i_{\max}}}$. The unknowns on level i with label red are assigned the level number i, and the vertices on level $i_{\max}$ with label black are assigned level number $i_{\max}+1$. The unknowns with level number j are called the level j unknowns. Note that every unknown has a unique level number.

Solution phase. For a clear description of the solution phase we introduce permute, collect, distribute and solve operations. These operations use the corresponding operators which are available from the decomposition phase. We give a description in a pseudo-programming language.

```
procedure permuteoperation($i$: int; var $\mathbf{x} \in \mathrm{I\!R}^{m_{i-1}}$)     (* uses $\hat{p}_i$*)
    for $j := m_i + 1$ to $m_{i-1}$ do
    if $j \neq \hat{p}_i(j)$ then interchange $x_j$ and $x_{\hat{p}_i(j)}$;

procedure collectoperation($i$: int;var $\mathbf{x} \in \mathrm{I\!R}^{n_i}$; $\mathbf{g} \in \mathrm{I\!R}^{m_i}$)   (* uses $\mathbf{A}_i^{rb}$*)
    compute $\mathbf{x} := \mathbf{x} - \mathbf{A}_i^{rb}\mathbf{g}$;

procedure distributeoperation($i$: int;var $\mathbf{x} \in \mathrm{I\!R}^{m_i}$;$\mathbf{g} \in \mathrm{I\!R}^{n_i}$) (* uses $\mathbf{A}_i^{br}$*)
    compute $\mathbf{x} := \mathbf{x} - \mathbf{A}_i^{br}\mathbf{g}$;

procedure solveoperation($i$: int; var $\mathbf{x} \in \mathrm{I\!R}^{n_i}$)   (* uses $\mathbf{A}_i^{rr}$*)
    solve $\mathbf{A}_i^{rr}\mathbf{w} = \mathbf{x}$. The result is written in $\mathbf{x}$.

procedure highestlevelsolve(var $\mathbf{x} \in \mathrm{I\!R}^{m_{i_{\max}}}$)   (* uses $\mathbf{A}_{i_{\max}+1}$*)
    solve $\mathbf{A}_{i_{\max}+1}\mathbf{w} = \mathbf{x}$; $\mathbf{x} := \mathbf{w}$;
```

Using these procedures it is easy to formulate the backward and forward substitution process, i.e. the solution phase, of the approximate cyclic reduction preconditioner. On each level i $(1 \leq i \leq i_{\max}+1)$ we define ULsolve as follows:

```
  procedure ULsolve($i$: int; var $\mathbf{f} \in \mathrm{I\!R}^{m_{i-1}}$);
    var $\mathbf{f}_{\text{red}} \in \mathrm{I\!R}^{n_i}$;
    begin
      if $i = i_{\max} + 1$ then highestlevelsolve($\mathbf{f}$) else
      begin
        permuteoperation($i, \mathbf{f}$);
        partition $\mathbf{f} = \begin{pmatrix} \mathbf{f}_b \\ \mathbf{f}_r \end{pmatrix}$ with $\mathbf{f}_r \in \mathrm{I\!R}^{n_i}$, $\mathbf{f}_b \in \mathrm{I\!R}^{m_i}$;
        make a copy $\mathbf{f}_{\text{red}} := \mathbf{f}_r$;
        solveoperation($i, \mathbf{f}_{\text{red}}$);
        distributeoperation($i, \mathbf{f}_b, \mathbf{f}_{\text{red}}$);
        ULsolve($i + 1, \mathbf{f}_b$);
        collectoperation($i, \mathbf{f}_r, \mathbf{f}_b$);
        solveoperation($i, \mathbf{f}_r$);
        permuteoperation($i, \mathbf{f}$);
      end
    end;
```

The solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ results from the call ULsolve$(1, \mathbf{b})$. The structure of ULsolve is similar to the structure of the multigrid V-cycle algorithm as presented in [11]. The distribute and collect operations correspond to the multigrid restriction and prolongation, respectively. The solve operation corresponds to the smoother in multigrid. Note, however, that in `ULsolve` we do not use any grid information and that every unknown is involved in the solve operations of precisely one level (as in hierarchical basis multigrid, cf. [1]).

3 Approximate Cyclic Reduction Preconditioning

In this section we introduce an approximate cyclic reduction preconditioner. For this we recall a few notions from graph theory.

A matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ induces an *ordered directed graph* $G_A(V,E)$, consisting of an ordered set of vertices $V = \{1,2,\dots,n\}$ and a set E of ordered pairs of vertices called *edges* . This set E consists of all pairs (i,j) for which $a_{ij} \neq 0$. A directed graph will also be called a *digraph.* If (i,j) is an element of E then i is said to be *adjacent to* j and j is said to be *adjacent from* i. Two vertices $i \neq j$ are said to be *independent* if $(i,j) \notin E$ and $(j,i) \notin E$. A subset M of V is called an independent set if every two vertices in M are independent. M is called a *maximal independent set* of vertices if M is independent but no proper superset of M in V is independent. Note that a maximal independent set is in general not unique. For a vertex $i \in V$, its *neighbourhood* $N(i)$ is defined by $N(i) = \{j \in V \mid j \neq i \text{ and } (i,j) \in E\}$. For $i \in V$ its *degree* , $deg(i)$, is the number of elements in the neighbourhood of i, that is, $deg(i) = |N(i)|$. A vertex i is called an *isolated* vertex if $deg(i) = 0$. Note that an isolated vertex can be adjacent *from* other vertices in V.

In the classical cyclic reduction method, as described in Sect. 2, a red-black partitioning $V = V_r \cup V_b,\;\; V_r \cap V_b = \{\emptyset\}$ of the vertex set V corresponding to a tridiagonal matrix is constructed such that V_r is a maximal independent subset of V. Since the vertices in V_r are independent, the matrix $\mathbf{A}_{rr}$ is diagonal. Moreover, the red-black partitioning yields a maximal independent set V_r for which the corresponding Schur complement $\mathbf{A}/\mathbf{A}_{rr}$ is tridiagonal and has much smaller dimension than the original matrix. For a general sparse matrix $\mathbf{A}$ it is easy to construct a partitioning $V = V_r \cup V_b,\;\; V_r \cap V_b = \{\emptyset\}$ of the vertex set of the graph $G_A(V,E)$ such that V_r is a maximal independent subset of V. Hence the same approach as in cyclic reduction, i.e. compute the Schur complement and apply the same technique recursively, can be applied. However, it is well-known that almost always one gets an unacceptable amount of fill-in in the Schur complement after only a few recursive steps. Thus this direct (!) method is not satisfactory. In the approximate cyclic reduction preconditioner we construct a partitioning $V = V_r \cup V_b,\;\; V_r \cap V_b = \{\emptyset\}$ such that $\mathbf{A}_{rr}$ is *strongly diagonally dominant*. Furthermore an *approximate* Schur complement is used and systems with matrix $\mathbf{A}_{rr}$ are solved only *approximately*. The preconditioner has the following structure, which is very similar to the cyclic reduction method of Sect. 2.

Decomposition phase. We assume a sparse matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$. *Dimbound*, with $1 < \textit{Dimbound} \ll n$ is a given integer (used in $\overline{\text{D5}}$ below). Set $i := 1$, $\mathbf{A}_1 := \mathbf{A}$, $m_0 := n$.

$\overline{\text{D1}}$. *Partitioning of the vertex set.* Compute a partitioning $V_i = V_i^r \cup V_i^b,\;\; V_i^r \cap V_i^b = \{\emptyset\}$ of the vertex set V_i of the graph corresponding to $\mathbf{A}_i$ such that the matrix $\mathbf{A}_i^{rr}$ is strongly diagonally dominant. This results in n_i vertices with label red and m_i vertices with label black. Note: $m_i + n_i = m_{i-1}$.
$\overline{\text{D2}}$ = D2. (as in Sect. 2)
$\overline{\text{D3}}$ = D3.
$\overline{\text{D4}}$. *Compute approximate Schur complement.* Compute a sparse approximation $\mathbf{A}_{i+1} \in \mathbb{R}^{m_i\times m_i}$ of the Schur complement $\mathbf{P}_i\mathbf{A}_i\mathbf{P}_i/\mathbf{A}_i^{rr}$.
$\overline{\text{D5}}$. *Store.* Save $m_i, \hat{p}_i, \mathbf{A}_i^{rr}, \mathbf{A}_i^{rb}, \mathbf{A}_i^{br}$. If $m_i < \textit{Dimbound}$ or $m_i > 0.8m_{i-1}$ then save $\mathbf{A}_{i+1}$ (stop the reduction process) else $i := i + 1$ and goto D1.

The methods used in $\overline{\text{D1}}$, $\overline{\text{D4}}$ will be explained below. In $\overline{\text{D5}}$ we introduced an additional stopping criterion to avoid a stagnation in the problem size reduction.

Solution phase. We apply the procedure `ULsolve` of Sect. 2 in which all procedures are *unchanged* except for the procedure `solveoperation` which is replaced by

```
procedure solveoperation($i$: int; var $\mathbf{x} \in \mathbb{R}^{n_i}$)
    solve $\mathbf{A}_i^{rr}\mathbf{w} = \mathbf{x}$ approximately, using a few iterations of a basic
    iterative method. The result is written in $\mathbf{x}$.
```

For the choice of the basic iterative method in `solveoperation` there are obvious possibilities: Jacobi, Gauss-Seidel, ILU. Note that the construction in the decomposition phase is such that the matrices $\mathbf{A}_i^{rr}$ are strongly diagonally dominant, hence these basic iterative methods have a high convergence rate for these systems. In this paper we use a method of Jacobi type. The solution of the system $\mathbf{A}_i^{rr}\mathbf{w} = \mathbf{x}$ is approximated by $\mathbf{w}^3$, which results from:

$$\begin{aligned} &\mathbf{w}^1 = (\mathbf{D}_i^{rr})^{-1}\mathbf{x} \qquad \text{(start)} \\ &\mathbf{w}^2 = \mathbf{w}^1 - (\tilde{\mathbf{D}}_i^{rr})^{-1}(\mathbf{A}_i^{rr}\mathbf{w}^1 - \mathbf{x}) \quad \text{(modified Jacobi)} \\ &\mathbf{w}^3 = \mathbf{w}^2 - (\mathbf{D}_i^{rr})^{-1}(\mathbf{A}_i^{rr}\mathbf{w}^2 - \mathbf{x}) \quad \text{(Jacobi)}, \end{aligned} \tag{6}$$

with $\mathbf{D}_i^{rr} = \text{diag}(\mathbf{A}_i^{rr})$, $\tilde{\mathbf{D}}_i^{rr}$ the diagonal matrix which satisfies $\tilde{\mathbf{D}}_i^{rr}\mathbb{1} = \mathbf{A}_i^{rr}\mathbb{1}$, $\mathbb{1} := (1, 1, \ldots, 1)^T$. Using $\mathbf{M} = \mathbf{I} - (\mathbf{D}_i^{rr})^{-1}\mathbf{A}_i^{rr}$, $\tilde{\mathbf{M}} = \mathbf{I} - (\tilde{\mathbf{D}}_i^{rr})^{-1}\mathbf{A}_i^{rr}$, a simple computation yields that $\mathbf{w}^3 = (\mathbf{I} - \mathbf{M}\tilde{\mathbf{M}}\mathbf{M})(\mathbf{A}_i^{rr})^{-1}\mathbf{x}$. If $\mathbf{A}_i^{rr}$ is symmetric positive definite then the matrix $(\mathbf{I} - \mathbf{M}\tilde{\mathbf{M}}\mathbf{M})(\mathbf{A}_i^{rr})^{-1}$ is symmetric. This conservation of symmetry is important for the CG method in Sect. 4. In general the rate of convergence of the Jacobi method (iteration matrix $\mathbf{M}$) is higher than the rate of convergence of the modified Jacobi method (iteration matrix $\tilde{\mathbf{M}}$). However, for the modified Jacobi method the (consistency) property $\tilde{\mathbf{M}}\mathbb{1} = 0$ holds, which is favourable for Poisson type of problems (cf. [15]). This motivates our choice for a symmetric combination of the Jacobi and the modified Jacobi method.

This procedure `solveoperation` is used both in Sect. 4 and in Sect. 5.

We now explain methods which can be used in in $\overline{\text{D1}}$ and $\overline{\text{D4}}$. Numerical experiments based on these methods are presented in Sect. 4 and Sect. 5.

Partitioning of the vertex set. The partitioning method consists of three steps, where the third one is optional (cf. Sect. 4 and 5).
P1. <u>Compute a *reduced* digraph</u>. As in algebraic multigrid methods (cf. [19,25]), for the graph coarsening we distinguish "strong" and "weak" edges in the digraph. The underlying multigrid heuristic is that if one wants to use simple (point) smoothers then one should coarsen in the direction of the "strong" connections.
Every loop in E, i.e. an edge of the form (i, i), is labeled strong. For every nonisolated vertex $i \in V$ an edge $(i, j) \in E$ with $j \neq i$ is labeled strong if for the corresponding matrix entry a_{ij} we have:

$$|a_{ij}| \geq \beta \max_{j \in N(i)} |a_{ij}| \quad , \tag{7}$$

with $0 \leq \beta < 1$ a given parameter (typically $0.4 \leq \beta \leq 0.6$). An edge is labeled weak if it is not strong. Note that for every nonisolated vertex i there is at least one strong edge (i, j) with $j \neq i$. Thus we obtain a partitioning $E = E_s \cup E_w$ of the edges into strong (E_s) and weak (E_w) edges. The directed graph consisting of the vertex set V and the set of strong edges E_s is called the *reduced* digraph and is denoted by $G_A(V, E_s)$.

P2. Compute a maximal independent set of the reduced digraph. We compute a maximal independent set M of the reduced digraph $G_A(V, E_s)$. This can be realized with low computational costs using a simple breath first search technique (cf. [14,18]). A vertex $i \in V$ is assigned a red (black) label if $i \in M$ ($i \notin M$).
P3.(optional) Check for diagonal dominance. The vertex set partitioning constructed in P1, P2 results in a corresponding block partitioning of the matrix $\mathbf{A}$ as in (1). We now check diagonal dominance of the $\mathbf{A}_{rr}$ block. If for a given parameter value κ

$$\sum_j |(\mathbf{A}_{rr})_{i,j}| > \kappa\, |(\mathbf{A}_{rr})_{i,i}| \tag{8}$$

for some red vertex i, then the red label of this vertex is changed to black. In our applications we use $\kappa = 1.5$.

Approximate Schur complement. In [18] on can find a technique for approximating the Schur complement which is based on replacing the *block* Gaussian elimination (cf. (4)) by a sequence of *point* Gaussian elimination steps. In the present paper we use a simple variant of this technique that can be interpreted as a Galerkin approach with matrix dependent prolongation and restriction. Note that for a matrix of the form

$$\mathbf{A}^{(0)} = \begin{bmatrix} \mathbf{A}_{bb} & \mathbf{A}_{br} \\ \mathbf{A}_{rb} & \mathbf{A}_{rr} \end{bmatrix} ,$$

the Schur complement $\mathbf{S}_{bb} = \mathbf{A}^{(0)}/\mathbf{A}_{rr}$ can be represented as

$$\mathbf{S}_{bb} = \begin{bmatrix} \mathbf{I}_b & -\mathbf{A}_{br}\mathbf{A}_{rr}^{-1} \end{bmatrix} \mathbf{A}^{(0)} \begin{bmatrix} \mathbf{I}_b \\ * \end{bmatrix} = \begin{bmatrix} \mathbf{I}_b & * \end{bmatrix} \mathbf{A}^{(0)} \begin{bmatrix} \mathbf{I}_b \\ -\mathbf{A}_{rr}^{-1}\mathbf{A}_{rb} \end{bmatrix} , \tag{9}$$

with $*$ arbitrary. Let $\mathbf{D}_{rr} = \mathrm{diag}(\mathbf{A}_{rr})$ and $\tilde{\mathbf{D}}_{rr}$ be the diagonal matrix which satisfies $\tilde{\mathbf{D}}_{rr}\mathbb{1} = \mathbf{A}_{rr}\mathbb{1}$ and let

$$\tilde{\mathbf{p}}_A = \begin{bmatrix} \mathbf{I}_b \\ -\tilde{\mathbf{D}}_{rr}^{-1}\mathbf{A}_{rb} \end{bmatrix}, \quad \tilde{\mathbf{r}}_A = \begin{bmatrix} \mathbf{I}_b & -\mathbf{A}_{br}\tilde{\mathbf{D}}_{rr}^{-1} \end{bmatrix},$$
$$\mathbf{r}_{\mathbf{A}} = \begin{bmatrix} \mathbf{I}_b & -\mathbf{A}_{br}\mathbf{D}_{rr}^{-1} \end{bmatrix}, \quad \mathbf{r}_{\mathrm{inj}} = \begin{bmatrix} \mathbf{I}_b & \emptyset \end{bmatrix}.$$

For the approximation of the Schur complement in (9) we will use one of the following two Galerkin operators:

S1. Galerkin approximation $\quad \hat{\mathbf{S}}_{bb}^{(1)} = \tilde{\mathbf{r}}_A \mathbf{A}^{(0)} \tilde{\mathbf{p}}_A, \quad \hat{\mathbf{S}}_{bb}^{(2)} = \mathbf{r}_A \mathbf{A}^{(0)} \tilde{\mathbf{p}}_A.$ (10)

The approximation $\hat{\mathbf{S}}_{bb}^{(1)}$ is spd if $\mathbf{A}^{(0)}$ is spd. From the analysis in [18] it follows that the approximation $\hat{\mathbf{S}}_{bb}^{(1)}$ has better consistency properties than $\hat{\mathbf{S}}_{bb}^{(2)}$, whereas the latter has better stability properties.

To reduce the amount of fill-in in the approximate Schur complement $\hat{\mathbf{S}}_{bb}^{(k)}$ we use standard techniques: Restriction to a prescribed pattern (S2) and thresholding (S3). In the applications in Sect. 4, 5 we always use S2, whereas S3 is optional.
S2. Restriction to prescribed pattern

$$\tilde{\mathbf{S}}_{bb}^{(k)} := \left(\hat{\mathbf{S}}_{bb}^{(k)}\right)_{|\mathrm{graph}(\mathbf{r}_{\mathrm{inj}}\mathbf{A}^{(0)}\tilde{\mathbf{p}}_A)}, \quad k = 1, 2. \tag{11}$$

Here we use (with $\mathbf{C}, \mathbf{B} \in \mathbb{R}^{n \times n}$) the notation $(\mathbf{C}_{|_{\mathrm{graph}(\mathbf{B})}})_{i,j} = C_{i,j}$ if $i \neq j$ and $(i,j) \in \mathrm{graph}(\mathbf{B})$, $(\mathbf{C}_{|_{\mathrm{graph}(\mathbf{B})}})_{i,j} = 0$ if $i \neq j$ and $(i,j) \notin \mathrm{graph}(\mathbf{B})$ and $\mathrm{diag}(\mathbf{C}_{|_{\mathrm{graph}(\mathbf{B})}})$ such that $\mathbf{C}_{|_{\mathrm{graph}(\mathbf{B})}} \mathbb{1} = \mathbf{C}\mathbb{1}$ (i.e. entries outside the pattern are added to the diagonal).
S3. Thresholding. We use a threshold paremeter $\varepsilon_t > 0$. Let $\tilde{\mathbf{S}}_{bb} \in \mathbb{R}^{m \times m}$ be an approximate Schur complement. For $1 \leq i \leq m$, let $k_i > 0$ be the number of nonzero entries in the ith row of $\tilde{\mathbf{S}}_{bb}$. In row i every entry $(\tilde{\mathbf{S}}_{bb})_{i,j}$ with

$$|(\tilde{\mathbf{S}}_{bb})_{i,j}| < \varepsilon_t \frac{1}{k_i} \sum_j |(\tilde{\mathbf{S}}_{bb})_{i,j}| \tag{12}$$

is replaced by zero. This is done for all rows $i = 1, 2, \dots, m$.

4 Application to a SPD Helmholtz Problem

In this section we show results of a few numerical experiments with the approximate cyclic reduction (CR) preconditioner. We consider the standard 5-point finite difference discretization of the Helmholtz problem $-\Delta u - \lambda u = f$ on $(0,1)^2$ with zero Dirichlet boundary conditions on a uniform grid with mesh size h. The smallest eigenvalue of the discrete operator is $\lambda_{\min} = 8h^{-2}\sin^2(\frac{1}{2}\pi h) - \lambda = 2\pi^2 - \lambda + \mathcal{O}(h^2) = 19.73921 - \lambda + \mathcal{O}(h^2)$. We take $\lambda = 0$, i.e. the Poisson equation, and $\lambda = 19.73$. In both cases the discrete problem is symmetric positive definite. We will compare the CG method with CR preconditioning (CR-CG) with the standard ICCG method. The implementation is done in MATLAB and we used the MATLAB function CHOLINC. We consider mesh sizes $h = \frac{1}{60}$ (symbol '+' in the figures), $h = \frac{1}{120}$ (symbol '$*$') and $h = \frac{1}{240}$ (symbol 'o'). In all experiments we take the righthand side such that the discrete solution is given by $\mathbb{1}/\|\mathbb{1}\|_2$ and we use zero as the starting vector.

Experiment 1. We take $\lambda = 0$. We choose the parameter value *Dimbound* $= 50$ in the decomposition phase. In the CR-preconditioner we use in step $\overline{\mathrm{D1}}$ in the decomposition phase the method P1, P2 described in Sect. 3. We take $\beta = 0.6$ in (7). For this problem we do not need the check for diagonal dominance in P3 (this follows from the analysis in [18] and is confirmed by numerical experiments). In step $\overline{\mathrm{D4}}$ in the decomposition phase we use the method as in S1, S2 (i.e. (10), (11)) with $k = 1$, i.e. the symmetric variant. We do not use the thresholding strategy described in S3. Numerical results for the ICCG and CR-CG methods are shown in Fig. 1 and Fig. 2. In Fig. 1 one can observe the well-known dependence of the convergence rate of the ICCG method on the mesh size h. When h is halved then, in order to obtain a fixed error reduction, one needs approximately twice as many ICCG iterations. This h dependence is much weaker for the CR-CG method (although there still seems to be a mild h-dependence). Also note that the ICCG method shows relatively slow convergence in the first phase of the solution process (superlinear convergence behaviour), whereas the CR-CG method has an almost linear convergence behaviour. For $h = \frac{1}{240}$, to reduce the starting error with a factor 100 one needs about 70 ICCG iterations but only 2 CR-CG iterations.

Remark 2. We give an indication of the storage needed for the methods in Experiment 1 for the case $h = \frac{1}{240}$, i.e. $n = 239^2$. For the symmetric matrix $\mathbf{A}$ storage for approximately $2\frac{1}{2}n$ entries is needed. The blocks $\mathbf{A}_1^{rr}$, $\mathbf{A}_1^{br}$, $\mathbf{A}_1^{rb}$, constructed in the first step (i.e. $i = 1$) of the decomposition phase of the CR-preconditioner, are blocks from the matrix $\mathbf{A}$, after a suitable permutation. Hence one does not need additional storage for these blocks. The union of the blocks $\mathbf{A}_i^{br} = (\mathbf{A}_i^{rb})^T$ over all levels $i > 1$ contains approximately $2.4n$ nonzero entries. For the union of the symmetric blocks $\mathbf{A}_i^{rr}$ over all levels $i > 1$ we need storage for approximately $1.3n$ entries. The storage needed for

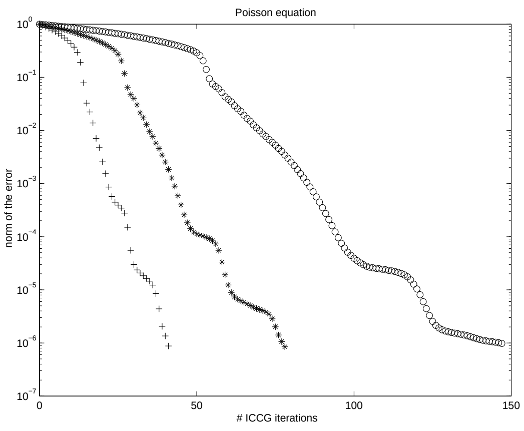

Fig. 1. ICCG

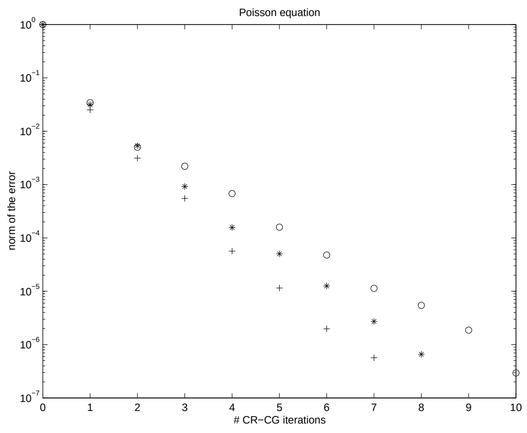

Fig. 2. CR-CG

the approximate Schur complement on the highest level is negligible. It follows that for the CR-preconditioner (additional) storage for approximately $3.7n$ entries is needed. For the incomplete Choleski preconditioner storage for approximately $3n$ entries is needed. Hence, for this problem, the CR-preconditioner needs only 20–30 percent more storage than the IC-preconditioner. The numerical experiments show that this statement also holds for the cases $h = \frac{1}{60}$ and $h = \frac{1}{120}$ in Experiment 1.

We briefly discuss the arithmetic work, for $h = \frac{1}{240}$, needed for one evaluation of the CR preconditioner (i.e. one call of `ULsolve`$(1, \mathbf{b})$). As a unit of arithmetic work we use one matrix-vector multiplication with the given matrix $\mathbf{A}$, denoted by MATVEC. The total arithmetic work needed in the `distributeoperation` and `collectoperation` over all levels $i \geq 1$ is approximately 2 MATVEC. In the two calls of `solveoperation` on level i we need arithmetic work comparable to 4 Jacobi iterations applied to a system with matrix $\mathbf{A}_i^{rr}$. Adding these costs over all levels $i \geq 1$ results in approximately $2\frac{1}{2}$ MATVEC arithmetic work. The costs for `highestlevelsolve` are negligible. It follows that the arithmetic costs for one CR-CG iteration are approximately $2\frac{1}{2}$ times the costs of one ICCG iteration. The same statement holds for the cases $h = \frac{1}{60}$ and $h = \frac{1}{120}$ in Experiment 1.

Experiment 2. We consider $\lambda = 19.73$. Application of the MATLAB function CHOLINC yields a well-defined incomplete Choleski factorization of the matrix $\mathbf{A}$. The results for the ICCG method are shown in Fig. 3. We take all components and all parameter values in the CR algortihm as in Experiment 1. The results for the CR-CG method are given in Fig. 4. It turns out that, both with respect to storage and with respect to computational costs per iteration, results very similar to those formulated in Remark 2 hold. Note that the CR-CG algorithm is much more efficient than the ICCG method, but both methods have a large stagnation phase at the beginning.

5 Application to an Indefinite Helmholtz Problem

In this section we consider a discrete Helmholtz problem as in Sect. 4 (with $h = \frac{1}{60}, \frac{1}{120}, \frac{1}{240}$) but now for $\lambda = 100$, $\lambda = 200$. In these cases the problem is indefinite. In all three cases, $h = \frac{1}{60}, \frac{1}{120}, \frac{1}{240}$, the discrete operator has 6 negative eigenvalues (counted with multiplicity) if $\lambda = 100$ and 13 negative eigenvalues if $\lambda = 200$. The CG method is no longer applicable. The MINRES method could be used for this type of problem. If, however, one wants to combine this method with preconditioning then one needs a symmetric positive definite preconditioner. It is not clear

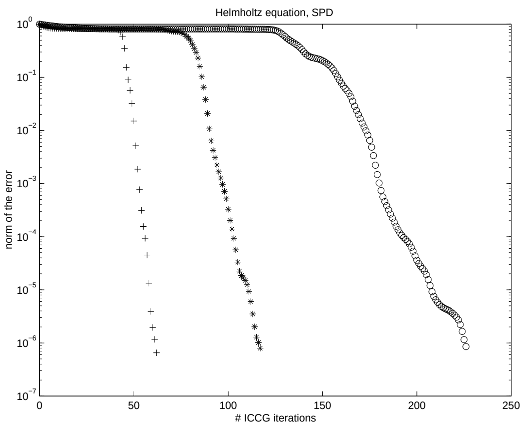

Fig. 3. ICCG

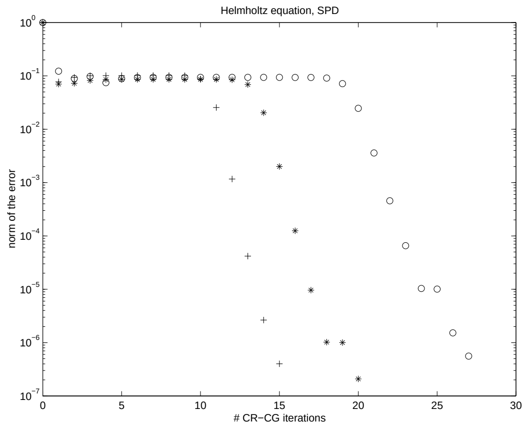

Fig. 4. CR-CG

how to construct an efficient spd preconditioner for this problem. We will use a (preconditioned) GMRES method with restart after 5 iterations (GMRES(5)) as a solver. For the preconditioner we use standard ILU techniques and the approximate CR method. For the ILU preconditioner the MATLAB function LUINC is applied which can be used for computing the standard ILU(0) factorization and for computing an ILU factorization based on droptolerances (ILU(*eps*), where *eps* denotes the drop tolerance). The righthand side and starting vector are as in Sect. 4.

Experiment 3. We take $\lambda = 100$. The ILU(0) and ILU(*eps*) factorizations are computed using the MATLAB function LUINC. In Table 5 we give the number of nonzero entries in the preconditioner (where we do not make use of symmetry). The convergence behaviour of the GMRES(5) method with ILU left preconditioning (ILU-GMRES(5)) is shown for $h = \frac{1}{60}$, $h = \frac{1}{120}$ in Fig. 5 and Fig. 6, respectively. In these figures we use the following symbols: '+' for ILU(0), 'x' for ILU(0.01), 'o' for ILU(0.005), '$*$' for ILU(0.002). Note that the unit on the horizontal axis is one preconditioned GMRES(5) iteration, which consists of 5 preconditioned GMRES iterations. In Fig. 5 and Fig. 6 we see slow convergence and stagnation phases. Moreover, the dependence of the preconditioner on the threshold parameter *eps* is unpredictable. For example, for $h = \frac{1}{120}$ the result for $eps = 0.002$ is significantly better than for $eps = 0.005$ (after 100 GMRES(5) iterations), whereas for $h = \frac{1}{60}$ it is the other way round. Numerical experiments for the case with $\lambda = 200$ show a

Table 1. Number of nonzero entries in ILU preconditioner

	ILU(0)	ILU(0.01)	ILU(0.005)	ILU(0.002)
$h = \frac{1}{60}$	17169	36828	54833	86659
$h = \frac{1}{120}$	70329	152583	218851	324555

similar unsatisfactory behaviour of GMRES(5) with ILU preconditioning.

Experiment 4. We consider the indefinite problem with $\lambda = 100$, $\lambda = 200$ and apply GMRES(5) with CR preconditioning. If in the decomposition phase we use the same components and parameter values as in Experiments 1 and 2 then the resulting preconditoner is not satisfactory. The main cause for this poor behaviour lies in the fact that if we only use the method P1, P2 in the partitioning step $\overline{\text{D1}}$ then for indefinite problems (strong) diagonal dominance of the $\mathbf{A}_{rr}$ is not

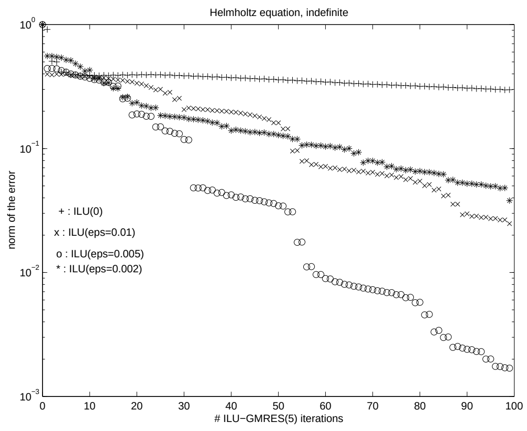

Fig. 5. ILU-GMRES(5), $h = \frac{1}{60}$

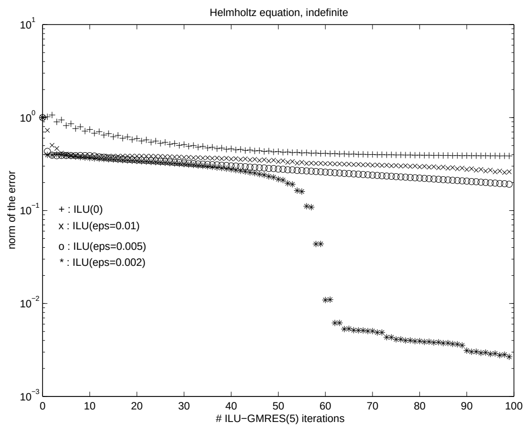

Fig. 6. ILU-GMRES(5), $h = \frac{1}{120}$

guaranteeed. Hence, in addition to P1, P2 we now also use the optional method in P3. This yields a significant improvement of the performance of the preconditoner. In view of the better stability properties, we take the Schur complement approximation $\hat{\mathbf{S}}_{bb}^{(2)}$ in S1, (10). A significant further improvement can be obtained if we allow more fill-in in the preconditioner. For this we simply lower the value of the parameter β in (7). This causes slower coarsening and more fill-in. Based on numerical experiments for an indefinite problem ($\lambda = 200$) of relatively low dimension ($h = \frac{1}{40}$) we choose the parameter value $\beta = 0.4$. The increase of fill-in, due this choice of β, can become very large if one goes to higher levels. However, many fill-in entries turn out to be very small. Hence we now also use the optional thresholding step S3. Based on numerical experiments for a low dimensional problem we take the threshold parameter $\varepsilon_t = 0.001$ in (12). Summarizing, in the decomposition phase we use P1, P2, P3 (in $\overline{\mathrm{D1}}$), S1 (with $\hat{\mathbf{S}}_{bb}^{(2)}$), S2, S3 (in $\overline{\mathrm{D4}}$) with parameter values $Dimbound = 50$, $\beta = 0.4$, $\varepsilon_t = 0.001$. The results of the GMRES(5) method with CR preconditioner are shown in Fig. 7 ($\lambda = 100$) and Fig. 8 ($\lambda = 200$). The use of the symbols '+','$*$','o' is as in Sect. 4.

We give an indication of the storage and arithmetic work needed for the CR preconditioner (cf. Re-

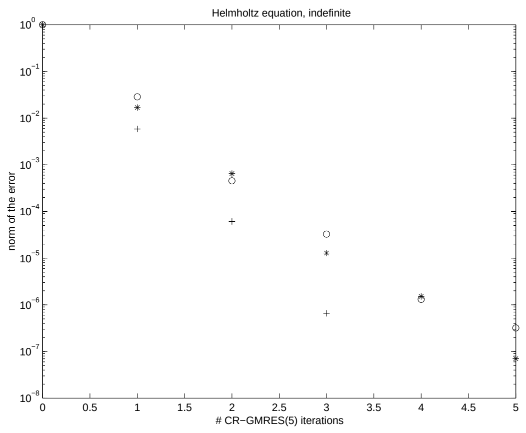

Fig. 7. CR-GMRES(5), $\lambda = 100$

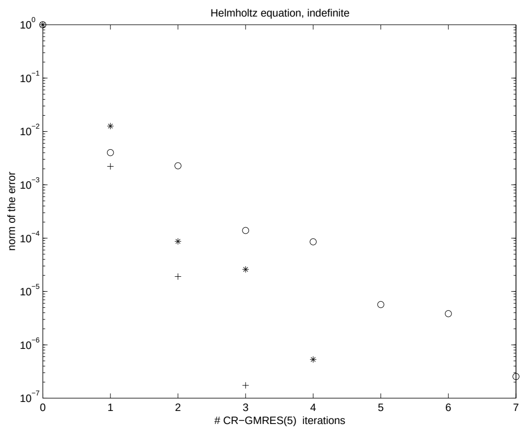

Fig. 8. CR-GMRES(5), $\lambda = 200$

mark 2). We consider the case $h = \frac{1}{240}, \lambda = 200$, i.e. $n = 239^2$. Due to the use of the nonsymmetric Schur complement preconditoner $\hat{\mathbf{S}}_{bb}^{(2)}$ in (10) the preconditioner is in general not symmetric. The union of the blocks $\mathbf{A}_i^{br},\ \mathbf{A}_i^{rb}$ over all levels $i > 1$ contains approximately $25n$ nonzeros entries.The union of the blocks $\mathbf{A}_i^{rr}$ over all levels $i > 1$ contains approximately $5n$ nonzeros entries. It follows that for this preconditioner the storage needed is approximately 6 times as high as for the matrix **A** (if we do not make use of symmetry). Hence these storage costs are quite high (cf. Remark 3). The total arithmetic work needed in the `collectoperation` and in the `distributeoperation` over all levels $i \geq 1$ is approximately $5\frac{1}{2}$ MATVEC. In the two calls of `solveoperation` over all levels $i \geq 1$ we need arithmetic work comparable to 5 MATVEC.
Similar results, both with respect to storage and with respect to arithmetic work , hold for the other cases ($h = \frac{1}{60}, \frac{1}{120},\ \lambda = 100$). For this type of indefinite problem the storage and arithmetic costs appear to be high. Note, however, that these problems are known to be very hard for other iterative solvers like geometric multigrid and Krylov methods with ILU preconditioning (cf. Experiment 3). From Fig. 8 we see that, for the case $h = \frac{1}{240}$, after 7 CR-GMRES(5) iterations the error has been reduced with a factor 10^7, i.e. a factor 10 per CR-GMRES(5) iteration, which corresponds to a factor 1.6 per preconditioned GMRES iteration.

Remark 3. Note that, opposite to ILU preconditioners, the solution phase of the CR preconditioner is easy to parallelize. This parallelization can be realized along the same lines as for a geometric multigrid V-cycle (cf. [4]). The graph partitioning method in P2 can be replaced by a similar method which is suitable for parallelization. Using such a variant, the decomposition phase in the CR preconditioner is also easy to parallelize.

References

1. Bank, R.E., Dupont, T.F., Yserentant, H.: The hierarchical basis multigrid method. Numer. Math. **52** (1988) 427–458
2. Bank, R.E., Smith, R.K.: The incomplete factorization multigraph algorithm. SIAM J. Sci. Comput. **20** (1999) 1349–1364
3. Bank, R.E., Wagner, C.: Multilevel ILU decomposition. Numer. Math **82** (1999) 543–576
4. Bastian, P.: Parallele adaptive Mehrgitterverfahren. Teubner Skripten zur Numerik, Teubner, Stuttgart, Leipzig (1996)
5. Botta, E.E.F., Van der Ploeg, A.: Preconditioning techniques for matrices with arbitrary sparsity patterns. In: Proceedings of the Ninth International Conference on Finite Elements in Fluids, New Trends and Applications (1995) 989–998
6. Botta, E.E.F., Wubs, W.: MRILU: it's the preconditioning that counts. Report W-9703, Department of Mathematics, University of Groningen, The Netherlands (1997)
7. Braess, D.: Towards algebraic multigrid for elliptic problems of second order. Computing **55** (1995) 379–393
8. Bruaset, A.M.: A survey of preconditioned iterative methods. Pitman Research Notes in Mathematics **328** Longman (1995)
9. Dendy, J.E.: Black box multigrid. J. Comput. Phys. **48** (1982) 366–386
10. Golub, G.H., Van Loan, C.: Matrix Computations. Johns Hopkins University Press, second edition (1989)
11. Hackbusch, W.: Multigrid methods and applications. Springer, Berlin, Heidelberg, New York (1985)
12. Hackbusch, W.: Iterative solution of large sparse systems of equations. Springer, New York (1994)
13. Heller, D.: Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems. SIAM J. Numer. Anal. **13** (1976) 484–496
14. Horowitz, E., Sahni, S.: Fundamentals of data structures in Pascal. Pitman, London (1984)
15. Notay, Y.: Using approximate inverses in algebraic multilevel preconditioning. Numer. Math. **80** (1998) 397–417

16. Van der Ploeg, A.: Preconditioning for sparse matrices with applications. PhD thesis, University of Groningen (1994)
17. Reusken, A.: Approximate cyclic reduction preconditioning. In: Multigrid methods 5 (W. Hackbusch and G. Wittum, eds.). Lecture Notes in Computational Science and Engineering **3** (1998) 243–259
18. Reusken, A.: On the approximate cyclic reduction preconditioner. SIAM J. Sci. Comput., to appear
19. Ruge, J.W., Stüben, K.: Algebraic multigrid. In: Multigrid Methods (S.F. McCormick, ed.). SIAM, Philadelphia (1987) 73–130
20. Saad, Y.: Iterative methods for sparse linear systems. PWS Publishing Company, Boston (1996)
21. Saad, Y.: ILUM: a multi-elimination ILU preconditioner for general sparse matrices. SIAM J. Sci. Comput. **17** (1996) 830–847
22. Saad, Y., Zhang, J.: BILUM: block versions of multi-elimination and multi-level ILU preconditioner for general sparse linear systems. Report UMSI 97/126, Department of Computer Science, University of Minnesota (1997)
23. Stüben, K.: Algebraic multigrid (AMG): An introduction with applications. GMD Report **53** (1999)
24. Swarztrauber, P.N.: The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle. SIAM Review **19** (1977) 490–501
25. Vanek, P., Mandel, J., Brezina, M.: Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. Computing **56** (1996) 179–196
26. Wesseling, P.: An introduction to multigrid methods. Wiley, Chichester (1992)