Extensions of Certain Graph-based Algorithms or Preconditioning

David Fritzsche, Andreas Frommer,
and Daniel B. Szyld

Report 06-05-29
May 2006, Revised January 2007

EXTENSIONS OF CERTAIN GRAPH-BASED ALGORITHMS FOR PRECONDITIONING[*]

DAVID FRITZSCHE[†], ANDREAS FROMMER[‡], AND DANIEL B. SZYLD[†]

Abstract. The original TPABLO algorithms are a collection of algorithms which compute a symmetric permutation of a linear system such that the permuted system has a relatively full block diagonal with relatively large nonzero entries. This block diagonal can then be used as a preconditioner. We propose and analyze three extensions of this approach: we incorporate a nonsymmetric permutation to obtain a large diagonal, we use a more general parametrization for TPABLO, and we use a block Gauss-Seidel preconditioner which can be implemented to have the same execution time as the corresponding block Jacobi preconditioner. Experiments are presented showing that for certain classes of matrices, the block Gauss-Seidel preconditioner used with the system permuted with the new algorithm can outperform the best ILUT preconditioners in a large set of experiments.

Key words. block partitioning, PABLO, TPABLO, preconditioning, block Gauss-Seidel, ILU decomposition

AMS subject classifications. 65F50, 65F10, 65F05

1. Introduction.

We consider a nonsingular linear system

$$Ax = b,\ A \in \mathbb{R}^{n\times n},\ x, b \in \mathbb{R}^n, \tag{1.1}$$

with n large and A sparse. We do not assume A to be symmetric. Direct methods, based on Gaussian elimination, have reached a mature state for this general system, see, e.g., [18], [19], [22]. They may, however, suffer from severe fill-in so that memory resources may become insufficient and/or execution times may become too large.

Modern iterative methods for the solution of (1.1) work best with a good preconditioner, and at each step, the basic operation is usually one matrix-vector multiplication with A and one application of the preconditioner. A good preconditioner contains as much information about the matrix A as possible. Incomplete factorizations of A (ILU preconditioners) have proven to work well in many cases, [4].

The PABLO (parametrized block ordering) and TPABLO (threshold PABLO) approaches to preconditioning proceed in the following manner: a permutation matrix $P \in \mathbb{R}^{n\times n}$ is computed such that the permuted matrix $\widehat{A} = PAP^T$ has a (relatively) dense block diagonal; see [5], [13], [14], [30]. Block sizes are determined dynamically. In the simplest case the preconditioner is taken as the block diagonal $\widehat{D}$ with respect to these blocks, i.e., one solves the permuted system

$$\widehat{A}\widehat{x} = \widehat{b}, \quad \text{where} \quad \widehat{A} = PAP^T,\ \widehat{x} = Px,\ \text{and}\ \widehat{b} = Pb,$$

iteratively using GMRES (or some other Krylov subspace method) with preconditioner $\widehat{D}$; see, e.g., [29], [34], [36] for their description. To solve the preconditioning system $\widehat{D}s = r$, a (complete) LU decomposition $\widehat{D} = \widehat{L}\,\widehat{U}$ is computed. Since this reduces to an LU decomposition on each of the diagonal blocks, we can use readily available packages. For small and dense blocks we use level 3 BLAS based LAPACK

[*]Original version dated 29 May 2006. Revised version dated 23 January 2007

[†]Department of Mathematics, Temple University, Philadelphia, PA 19122-6094, {fritzsche,szyld}@temple.edu. Supported in part by the National Science Foundation under grant DMS-0207525 and by the U.S. Department of Energy under grant DE-FG02-05ER25672.

[‡]Faculty of Mathematics and Science, Bergische Universität Wuppertal, D-42097 Wuppertal, Germany, frommer@math.uni-wuppertal.de

[2] routines and for large and sparse blocks we use UMFPACK [17]. Moreover, the LU decomposition of $\widehat{D}$ could be done in parallel.

Whereas PABLO determines the blocks for $\widehat{D}$ by just using structural information of A given through its associated directed graph, the TPABLO variants also take the numerical values in A into account.

In the present paper, which builds upon [28], we propose three extensions of the PABLO and TPABLO preconditioners. First, we apply a non-symmetric permutation to put large entries on the diagonal. We discuss this in section 2. In section 3 we introduce new parametrizations to be used with PABLO and TPABLO. We describe the resulting algorithmic framework, termed XPABLO, in detail and analyze its computational complexity. In other words XPABLO is more general, it includes new different criteria to produce the blocking permutation, and it reduces to the previous versions for specific parametrizations. In section 4 we discuss how one can use either the lower or the upper block triangular part of the matrix as a preconditioner at the same cost as the block diagonal preconditioner.

After reviewing several practical issues related to the implementation in section 5, we finally give results of numerical experiments in section 6, including a comparison with ILU preconditioning. In section 7, we present some concluding remarks and an outlook to further development.

In addition to the references already mentioned, several authors have explored reorderings and partitioning techniques to accelerate preconditioning. Among them, we cite [6], [7], [8], [9], [23]; see also the sections on the effect of orderings in the survey [4].

We conclude this introduction by noting that PABLO variants have been applied successfully in the construction of preconditioners in settings that are different from the one presented in this paper. A slightly modified version of PABLO has been used successfully in a parallel computing setting to build block diagonal preconditioners for difficult problems in Computational Fluid Dynamics (CFD) and Chemical Engineering; see, e.g., [24]. The threshold variants TPABLO and XPABLO can be used as a method to find blocks with large entries in dense or sparse matrices. Specifically, in [26] and [27], XPABLO is used to find such blocks to build preconditioners for the CSYM method [12] for solving systems with complex symmetric matrices. The situation there is that an efficient preconditioner has to be factorized into its complex symmetric singular value decomposition, so that block diagonal preconditioners seem to be the only practical way. In the examples presented in [26] the iteration count is reduced by up to 30% when using this approach.

2. Nonsymmetric permutations and diagonal scalings. Let σ be the permutation associated with the permutation matrix $Q \in \mathbb{R}^{n\times n}$, i.e. for all $x \in \mathbb{R}^n$

$$(Qx)_i = x_{\sigma(i)}, \quad i = 1, \ldots, n.$$

A *maximum transversal* of the nonsingular matrix $A \in \mathbb{R}^{n\times n}$ is any permutation σ for which

$$(QA)_{ii} = a_{\sigma(i),i} \neq 0, \quad i = 1, \ldots, n.$$

A maximum transversal exists for the class of structurally nonsingular matrices which contains the class of nonsingular matrices; see, e.g., [19]. In [20], two different types of transversals with respect to different notions of a "weight" were introduced according to the following definition.

Definition 2.1. *A maximum transversal σ*

(i) is a bottleneck transversal *if it maximizes*

$$\min_{i=1,\dots,n} |a_{\sigma(i),i}|$$

over all maximum transversals.

(ii) is a maximum product transversal*, if it maximizes*

$$\prod_{i=1}^{n} |a_{\sigma(i),i}|$$

over all maximum transversals.

The paper [20] proposes and analyzes various algorithms to compute bottleneck transversals. Extensive testing shows that in practice the computational complexity of these algorithms is reported to behave like $\mathcal{O}(n+nz)$, where nz is the total number of nonzeros in the matrix, although the theoretical worst case upper bound may be much larger. Algorithms for computing bottleneck and maximum product transversals have been incorporated in the HSL-library as algorithm MC64; see [20], [21] for their description. As a by-product, when computing a maximum product transversal, MC64 also delivers diagonal scaling matrices $\hat{C}$ and $\hat{R}$ such that

$$\hat{A} = Q(\hat{R}A\hat{C}) \tag{2.1}$$

is an I-matrix, i.e., $|\hat{A}_{ij}| \le 1$ for all i,j and $\hat{A}_{ii} = 1$, $i = 1,\dots,n$.

The rationale for these different approaches to obtaining diagonals with large weight is heuristic: Large diagonals tend to decrease the need for pivoting in a direct elimination method. ILU-type preconditioners can also benefit greatly [6], [35]. Moreover, an iterative method with diagonal preconditioner may be expected to converge more rapidly if the diagonal is large compared to the off-diagonal part of the matrix.

In this paper, we use maximum product transversals and we always apply the scaling computed by MC64, i.e., we always transform A to an I-matrix. For our test problems in section 6 this gave us always better results than using the usually cheaper bottleneck transversal.

3. XPABLO: an extension of PABLO and TPABLO.

PABLO, TPABLO and XPABLO compute a (symmetric) permutation for A, and a block partition of it, based on information of the (directed) graph of A and the size of the entries of A. Let us quickly recall that the directed graph or digraph $G(A) = (V, E)$ associated with A is given as

$$V = \{1,\dots,n\}, \quad E = \{(i,j) : a_{ij} \neq 0\}.$$

Let $nz = |E|$, the cardinality of E. Using a somewhat non-standard terminology, we say that vertex $i \in V$ is *incident* with edge $e = (k,\ell)$, if $i = k$ or $i = \ell$. This terminology makes no distinction between the starting point k and the end point ℓ of the edge e. The two vertices k and ℓ incident to (k,ℓ) are called *adjacent*. Given a set of vertices $W \subseteq V$, the adjacency set $\operatorname{adj}(W)$ contains all vertices which are not in W but which are adjacent to a vertex in W, i.e.,

$$\operatorname{adj}(W) = \big\{ j \in V : j \notin W \text{ and } j \text{ is adjacent to some } i \in W \big\}.$$

To describe the PABLO algorithms, we need additional notation for subgraphs. If $G = (V,E)$ and $V' \subseteq V$, the induced sub-digraph $G|_{V'}$ is $G' = (V', E')$ with

$E' = \{(i,j) \in E \,:\, i,j \in V'\}$. Frequently, we use quantities that refer to an induced sub-digraph $G' = G|_{V'}$. We indicate this using the 'restriction notation' $|_{V'}$. For example, $\mathrm{adj}|_{V'}(W)$ refers to the adjacency set in the sub-digraph induced by V'. We write $|E|_{V'}$ to denote the cardinality of E'.

PABLO, TPABLO and XPABLO produce a partitioning of V into m disjoint, non-empty subsets V_κ, $\kappa = 1,\dots,m$ (the "blocks"). These blocks are built one at a time. We now describe how this is done, assuming the following situation: Blocks $V_1,\dots,V_{\nu-1}$ have already been built. Given the current block V_ν and a candidate vertex $i \in \mathrm{adj}|_{\overline{V}}(V_\nu)$ with $\overline{V} = V \setminus \bigcup_{\kappa=1}^{\nu-1} V_\kappa$, the algorithms decide whether or not to incorporate i into the current block using a criterion τ. The following definitions describe some basic criteria. They can be combined logically to yield a variety of different criteria, including, as we shall see, the traditional PABLO and TPABLO criteria. Algorithm 1 towards the end of the section shows how candidate vertices are selected; see also [30].

Definition 3.1. *Let $G = (V,E)$ be a digraph, $V' \subseteq V$ and $G' = (V',E')$ be the induced subgraph. The* fullness $\varphi(V')$ *of V' is defined as*

$$\varphi(V') = \begin{cases} \dfrac{|E'|}{|V'|^2 - |V'|} & \textit{if } |V'| > 1 \\ 0 & \textit{if } |V'| \le 1. \end{cases}$$

The fullness thus defined measures the number of edges in G' compared to the maximally possible number $|V'|^2 - |V'|$ for a complete digraph. The definition of $\varphi(\{v\}) = 0$ follows PABLO in the interpretation that a graph with only one vertex is empty, but other definitions may be useful depending on the problems one needs to solve.

Given $\alpha > 0$, in our generic situation, we say (see [30]) that vertex i satisfies the *fullness criterion* (with fullness parameter α) if

$$\varphi(V_\nu \cup \{i\}) \ge \alpha\varphi(V_\nu). \tag{FC}$$

Note that (FC) can be fulfilled even when $\alpha > 1$.

Definition 3.2. *The* degree $\deg(i)$ *of a vertex i in a digraph $G = (V,E)$ is the number of all in- and outgoing edges at i:*

$$\deg(i) = \big|\{e \in E \,:\, i \textit{ is incident with } e\}\big|.$$

If $V' \subseteq V$, we write, in a slight abuse of the restriction notation, $\deg|_{V'}(i)$ *to denote* $\deg(i)$ *in the graph induced by $V' \cup \{i\}$.*

Definition 3.3. *If i is a vertex in a digraph $G = (V,E)$ with* $\deg(i) > 0$*, and if $V' \subseteq V$, the* connectivity *of i with respect to V' is the fraction*

$$\frac{\deg|_{V'}(i)}{\deg(i)}.$$

In our generic situation, given $\beta > 0$ we say that vertex i satisfies the *connectivity criterion* (with connectivity parameter β) if

$$\deg|_{V_\nu}(i) \ge \beta \deg|_{\overline{V}}(i). \tag{CC}$$

The connectivity criterion means that in $G|_{\overline{V}}$ at least a fraction β of all edges incident with i have their other incident vertex in V_ν. The criterion is never met if $\beta > 1$ (and $\deg(i) > 0$).

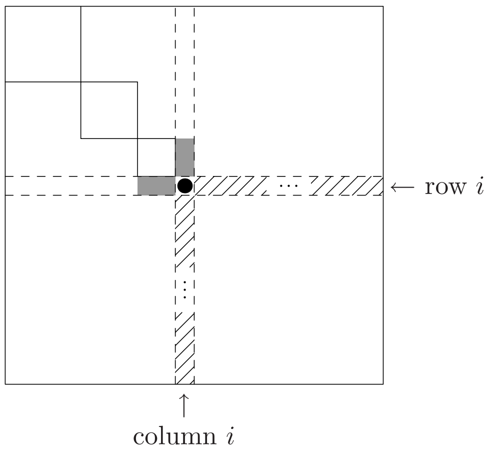

FIG. 3.1. *Illustration of the fullness criterion (FC) and the connectivity criterion (CC).*

Figure 3.1 illustrates the fullness and the connectivity criterion for the digraph $G(A)$ in terms of the pattern of the matrix A. The matrix is assumed to be already symmetrically permuted such that the blocks built up so far (there are 3 such blocks) appear first. The black diagonal entry corresponds to the candidate vertex to be included into the third block. The fullness criterion requires that the grey parts of row i and column i must not be too empty. The connectivity criterion means that the hatched parts of these rows and columns should not contain too many elements as compared to the grey parts. Row and column i are only considered together, not individually. The white parts of row i and column i are not taken into consideration when deciding on vertex i.

The PABLO algorithm of [30] adds a candidate vertex $i \in \text{adj}|_{\overline{V}}(V_\nu)$ to the current block V_ν if and only if v satisfies the fullness *or* the connectivity criterion for some pre-chosen parameters α and β. We formulate this by saying that PABLO uses the criterion τ defined as

$$\tau = \text{FC} \vee \text{CC}. \tag{PABLO}$$

This combined PABLO criterion only uses structural information of A.

In its threshold counterparts TPABLO1 and TPABLO2 (see [5], [13]), numerical values of the matrix entries are also taken into account. For a precise statement, and also to formulate our generalization XPABLO, we introduce additional notation and terminology. From now on we consider the digraph $G(A)$ as an edge-weighted digraph where edge $e = (i, j)$ has weight $w(e) = |a_{ij}|$.

DEFINITION 3.4. *Given $A \in \mathbb{R}^{n\times n}$ and $\gamma > 0$, we write $A^{>\gamma} \in \mathbb{R}^{n\times n}$ for the matrix*

$$(A^{>\gamma})_{ij} = \begin{cases} a_{ij} & \textit{if } |a_{ij}| > \gamma \\ 0 & \textit{otherwise.} \end{cases}$$

We use the superscript notation $^{>\gamma}$ in an intuitive manner at various places. For example, if $G = G(A)$ is the digraph of A, then $G^{>\gamma} = G(A^{>\gamma})$. Another example is the following definition.

DEFINITION 3.5. *Let $G = (V, E)$ be a digraph, $V' \subseteq V$ and $\gamma \geq 0$. The* threshold fullness *$\varphi^{>\gamma}(V')$ of V' is the fullness $\varphi(V')$ of V' in the graph $G^{>\gamma}$.*

In our generic situation, given a threshold parameter $\gamma > 0$ and a threshold fullness parameter $\theta \in [0,1]$, we say that i satisfies the *threshold fullness criterion* if

$$\varphi^{>\gamma}(V_\nu \cup \{i\}) \geq \theta. \tag{TFC}$$

In contrast to the plain fullness criterion, the threshold fullness criterion just measures the fullness the new block $V_\nu \cup \{i\}$ has in $A^{>\gamma}$ without relating it to the fullness of V_ν.

DEFINITION 3.6. *Let $G = (V, E)$ be a digraph, $V' \subseteq V$ and $\gamma \geq 0$. If i is a vertex with* $\deg|_{V'}(i) > 0$*, the* threshold connectivity *of i with respect to V' is the fraction*

$$\frac{\deg|_{V'}^{>\gamma}(i)}{\deg|_{V'}(i)}.$$

Threshold connectivity compares the number of heavy edges of i in $G|_{V'}$ to all such edges. As opposed to plain connectivity according to Definition 3.3, it does not consider any edges going to nodes outside V'.

In our generic situation, given a threshold connectivity parameter ζ, we say that the *threshold connectivity criterion* holds if

$$\deg|_{V_\nu}^{>\gamma}(i) \geq \zeta \deg|_{V_\nu}(i). \tag{TCC}$$

This criterion measures how many of the new entries of the current block are large. A useful choice for the threshold connectivity parameter is $\zeta = 1/2n$. For this ζ the right hand side of (TCC) is always strictly less than one since $\deg|_{V_\nu}(i) < 2n$ for all $i \in V$. Hence the threshold connectivity criterion holds if and only if $\deg|_{V_\nu}^{>\gamma}(i) \geq 1$.

In addition to the threshold parameter γ, XPABLO accepts also an additional threshold parameter δ, which is used to filter out nonzero matrix entries with very small magnitude, both from the graph $G(A)$ and from the threshold graph $G(A^{>\gamma})$, i.e., XPABLO really works on the graphs $G(A^{>\delta})$ and $G(A^{>\max(\gamma,\delta)})$. So we have $0 \leq \delta < \gamma$ in any useful application. Note that δ only changes the graphs on which XPABLO is operating and not the underlying matrix.

We now have four criteria at hand upon which we can decide whether to include a new vertex to a current block or not. These criteria can be logically combined resulting in various XPABLO criteria, denoted by τ. The following list shows how XPABLO is reduced to PABLO or TPABLO by choosing a specific criterion τ and specific values for some of the parameters.

1. With $\tau = \text{FC} \vee \text{CC}$, XPABLO is equivalent to PABLO.
2. With $\tau = (\text{FC} \vee \text{CC}) \wedge \text{TCC}$ and $\zeta = 1/(2n)$, XPABLO is equivalent to TPBALO1.
3. With $\tau = (\text{FC} \vee \text{CC}) \wedge \text{TFC}$ and $\theta = 1$, XPABLO is equivalent to TPABLO2.

The default criterion for XPABLO is $\tau = \text{FC} \vee \text{CC} \vee \text{TCC}$ with $\zeta = 1/(2n)$; see section 5.2 for a more detailed discussion of XPABLO default parameters.

Once XPABLO has determined a partitioning of V into blocks $V_1, \ldots, V_m$, let π be a permutation that groups the vertices $1, \ldots, n$, from V into these blocks, i.e. $V_1 = \{\pi(1), \ldots, \pi(k_1)\}$, $V_2 = \{\pi(k_1 + 1), \ldots, \pi(k_2)\}$, $\ldots$, $V_m = \{\pi(k_{m-1} + 1), \ldots, \pi(k_m)\}$, with $k_j = \sum_{i=1}^{j} |V_j|$. Then the permuted matrix PAP^T is naturally partitioned into m blocks of size $|V_j|$, the j-th diagonal block corresponding to the vertex set V_j. The threshold criteria may now be interpreted as placing large entries on these diagonal blocks. This is made more precise in the following proposition which follows immediately upon inspection of the various criteria.

PROPOSITION 3.7.

1. *If* $\tau = (FC \vee CC) \vee TCC$ *and* $\zeta = 1/(2n)$*, then all entries in the off-diagonal blocks of* PAP^T *have modulus less than* γ*.*
2. *If* $\tau = (FC \vee CC) \wedge TCC$ *and* $\zeta = 1$*, then all nonzero off-diagonal entries of every diagonal block of* PAP^T *have modulus* $\geq \gamma$*.*

Algorithm 1 now describes the XPABLO algorithm in more detail. It uses three data structures C, Q, and B to manage the vertices. The set B denotes the current block being built. During the algorithm, quantities needed for τ such as $\deg|_{V'}(i)$, $\deg|_{V'}^{>\gamma}(i)$, etc., are always updated immediately. The notation $\text{mult}(i,j)$ ($\in \{0,1,2\}$) for two nodes i, j refers to the number of directed edges between i and j.

The time complexity of Algorithm 1 is given in the following result.

PROPOSITION 3.8. *The XPABLO algorithm can be implemented with time complexity* $\mathcal{O}(n + |E|)$ *whenever an evaluation of criterion* τ *has cost* $\mathcal{O}(1)$*.*

Proof. We assume that $G = G(A)$ is given stored in an adjacency list representation including the values for $\text{mult}(i,j)$; see, e.g., [19]. Note that for the common sparse matrix formats $G(A)$ can be obtained with cost $\mathcal{O}(n + |E|)$.

Scanning the adjacency list representation, we build up $G^{>\gamma}$ and $\text{mult}^{>\gamma}(i,j)$ with complexity $\mathcal{O}(|E|)$. Taking Q as a queue, C as a doubly linked list with an external index and B as a singly linked list, all insert and remove operations we use with these lists as well as their flushing can be done with cost $\mathcal{O}(1)$. As was already shown in [30], the entire PABLO algorithm has then complexity $\mathcal{O}(n + |E|)$. The main observation to obtain this result is to realize that each node i is inserted into Q at most $\deg(i)$ times.

By assumption, the cost for testing criterion τ is $\mathcal{O}(1)$ as it is in PABLO. Therefore, the additional work in XPABLO as compared to PABLO is that we have to update the new quantities $|E|_B^{>\gamma}$, $\varphi^{>\gamma}(B)$ and $\deg|_B^{>\gamma}$ used in (TCC) and (TFC). These updates follow the ones to be done for $|E|_B$, $\varphi(B)$ and $\deg|_B$ by PABLO. In total, this means $\mathcal{O}(n)$ extra operations to update $|E|_B^{>\gamma}$ and $\varphi^{>\gamma}(B)$ and $\mathcal{O}(|E|)$ extra operations to update $\deg|_B^{>\gamma}$. So this is an additional cost of $\mathcal{O}(n + |E|)$, which establishes the proposition. ☐

Let us note that in our practical implementations we indeed build up $G^{>\gamma}$ explicitly. This is in contrast to the implementation of TPABLO in [13]. As is explained in [13], the implementation there is not more than d times as costly as PABLO when d is the maximum degree in G. We prefer to build up $G^{>\gamma}$ explicitly, since then TPABLO requires only about twice the work that PABLO does, so that our implementation is in most cases faster than that in [13].

We defer a detailed discussion on how to choose an XPABLO criterion τ and the various parameters (α, β, γ, θ, ζ) to the sections on practical issues and on numerical results (sections 5 and 6).

4. Efficient Block Gauss-Seidel Preconditioning. Let A denote the matrix obtained after the preprocessing steps and after applying the XPABLO permutation P of section 3, i.e., $A \leftarrow PQ(RAC)P^T$. Then A has a block structure

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mm} \end{bmatrix}, \qquad A_{ij} \in \mathbb{R}^{n_i \times n_j}$$

```
input: a digraph $G(A) = (V, E)$ and a criterion $\tau$
output: a partitioning $\{V_1, \ldots, V_m\}$ of $V$
set $C := V$, $Q := \emptyset$, $B := \emptyset$ and set $\kappa := 1$
set $\deg|_B(i) := 0$ and $\deg|_B^{>\gamma}(i) := 0$ for all $i \in V$
while $C \neq \emptyset$ do
    remove a vertex $i$ from $C$, and place it in $B$
    set $|E|_B := 0$, $|E|_B^{>\gamma} := 0$, $\varphi(B) := 0$, $\varphi^{>\gamma}(B) := 0$
    for all vertices $j$ adjacent to $i$ do
        move $j$ from $C$ to $Q$ if $j \in C$
        $\deg|_B(j) := \deg|_B(j) + \mathrm{mult}(i,j)$
        $\deg|_B^{>\gamma}(j) := \deg|_B^{>\gamma}(j) + \mathrm{mult}^{>\gamma}(i,j)$
    end for
    while $Q \neq \emptyset$ do
        remove a vertex $i$ from $Q$
        if $i$ fulfills the XPABLO criterion $\tau$ then
            insert $i$ into $B$
            set $|E|_B := |E|_B + \deg|_B(i)$
            set $|E|_B^{>\gamma} := |E|_B^{>\gamma} + \deg|_B^{>\gamma}(i)$
            update $\varphi(B)$ and $\varphi^{>\gamma}(B)$     {this requires $|E|_B$ and $|E|_B^{>\gamma}$}
            for all vertices $j$ adjacent to $i$ do
                move $j$ from $C$ to $Q$ if $j \in C$
                $\deg|_B(j) := \deg|_B(j) + \mathrm{mult}(i,j)$
                $\deg|_B^{>\gamma}(j) := \deg|_B^{>\gamma}(j) + \mathrm{mult}^{>\gamma}(i,j)$
            end for
        else
            insert vertex $i$ into $C$
        end if
    end while
    set $V_\kappa := B$ and $\kappa := \kappa + 1$     {finish current block}
    for all vertices $i \in B$ do
        for all vertices $j$ adjacent to $i$ do
            set $\deg|_B(j) := 0$ and $\deg|_B^{>\gamma}(j) := 0$
            $\deg(j) := \deg(j) - \mathrm{mult}(i,j)$     {needed for (CC)}
        end for
    end for
    set $B := \emptyset$
end while
```

Algorithm 1: XPABLO with criterion τ based on (CC), (FC), (TCC), and (TFC)

with $n_i = |V_i|$ being the size of the blocks corresponding to the XPABLO permutation P. Each diagonal block $D_i = A_{ii}$ in the block diagonal $D = \mathrm{diag}(D_1, \ldots, D_m)$ should be (relatively) full and should have (relatively) large nonzero entries. Even with A nonsingular, some of the diagonal blocks D_i can be singular; see section 5.1 for a discussion of this issue. For the moment we assume all D_i to be nonsingular.

We can compute the (row pivoted) LU factorization $D_i = \Pi_i L_i' U_i'$ for each block, so that $D = \Pi L' U'$, $\Pi = \mathrm{diag}(\Pi_i)$, $L' = \mathrm{diag}(L_i')$, $U' = \mathrm{diag}(U_i')$, and use this LU factorization of D when it comes to solving linear systems of the form

$$Ds = r \tag{4.1}$$

in a preconditioned iterative method with preconditioner D. This block diagonal preconditioning approach (with PABLO and TPABLO1, TPABLO2) was used in [14], where the numerical results were quite encouraging. Note that when the diagonal block D_i is dense, we can use the optimized dense matrix linear algebra code from LAPACK to perform these factorizations very efficiently [2]. When they are sparse, a sparse factorization is performed; we use the package UMFPACK [17].

An obvious extension of this idea is to use block Gauss-Seidel preconditioning, i.e., to include a block triangular part of A into the preconditioner. In the following discussion we will use the lower block triangular part, i.e., a forward block Gauss-Seidel iteration, as the preconditioner. It is easy to see that Proposition 4.1 below also holds when the preconditioner M is the upper block triangular part of A.

Note that block Gauss-Seidel iteration without acceleration by a Krylov subspace method was used in the original PABLO paper [30]. Using the established block structure the block Gauss-Seidel preconditioner M can be written as

$$M = \begin{bmatrix} D_1 & 0 & \cdots & 0 \\ A_{21} & D_2 & & \vdots \\ \vdots & & \ddots & 0 \\ A_{m1} & A_{m2} & \cdots & D_m \end{bmatrix}. \tag{4.2}$$

The usual interfaces for (left) preconditioned sparse iterative solvers separate the operation $z = M^{-1}Ax$ into a matrix vector multiplication $y = Ax$ and a solve operation $z = M^{-1}y$. Viewed in this way, computing $M^{-1}Ax$ seems to be much more expensive than computing $D^{-1}Ax$. But it is not more expensive at all, if we do not separate the operations, as we show in the next result. This result is basically a form of the well-known Eisenstat trick [25], [31], [32, pp. 208 and pp. 225]. Using this trick we could also implement efficient block SOR preconditioners, but this is not pursued in this paper.

PROPOSITION 4.1. *Let $A \in \mathbb{R}^{n\times n}$ be a matrix with an $m \times m$ block structure such that all diagonal blocks $D_i = A_{ii}$, $i = 1, \ldots, m$, are nonsingular. We denote $D = \operatorname{diag}(D_i)$ to be the block-diagonal part and M to be the lower block triangular part (including the diagonal blocks) of A. The preconditioned matrix-vector multiplications $D^{-1}Ax$ and $M^{-1}Ax$ take exactly the same number of operations.*

Proof. Let $z = M^{-1}Ax$ and $\tilde{z} = D^{-1}Ax$. For any vector $w \in \mathbb{R}^n$ we write w_i to denote the i-th block, conformal with the block structure of A. We split A as $A = D - L - U$ into the block diagonal part $D = \operatorname{diag}(D_i)$ (same as before), the block lower triangular part L with $L_{ij} = -A_{ij}$ for $i > j$ and the block upper triangular part U with $U_{ij} = -A_{ij}$ for $i < j$. Both L and U have a zero block diagonal.

Using this notation the block-triangular preconditioner is $M = D - L$ and we can rewrite the preconditioned matrix-vector multiplication as

$$\begin{aligned} M^{-1}Ax &= (D-L)^{-1}(D-L-U)x \\ &= (D-L)^{-1}(D-L)x - (D-L)^{-1}Ux \\ &= x - (D-L)^{-1}Ux. \end{aligned}$$

If we set $y = Ux$ and $v = (D-L)^{-1}y$ we get $z_i = x_i - v_i$, $i = 1, \ldots, m$, where

$$v_i = D_i^{-1}\Big(y_i + \sum_{j=1}^{i-1} L_{ij}v_j\Big) \qquad \text{and} \qquad y_i = \sum_{j=i+1}^{m} U_{ij}x_j.$$

We can further simplify the formula for v_i to

$$v_i = -D_i^{-1}\Big(\sum_{j=1}^{i-1} A_{ij}v_j + \sum_{j=i+1}^{m} A_{ij}x_j\Big). \tag{4.3}$$

Following the same procedure, we write $\tilde{z} = D^{-1}Ax = x - \tilde{v}$ and get $\tilde{z}_i = x_i - \tilde{v}_i$ where

$$\tilde{v}_i = -D_i^{-1}\Big(\sum_{j=1}^{i-1} A_{ij}x_j + \sum_{j=i+1}^{m} A_{ij}x_j\Big). \tag{4.4}$$

But the computations (4.3) and (4.4) use the exact same amount of operations if we assume that dense vectors are used. $\square$

5. Practical issues. Before reporting on our numerical experiments, we address some practical issues when using XPABLO in a preconditioning framework. We concentrate on the two main issues, namely numerical stability and the choice of a good set of values for all the different XPABLO parameters.

5.1. Robustness. We consider the issue of numerical instability (or even the singularity) of the diagonal blocks D_ν in the block-triangular preconditioner M. If A is a general nonsingular matrix, there is no guarantee that all the blocks D_i are nonsingular, nor that M is nonsingular. We mention that difficulties with singular blocks were very rare in our numerical experiments using the recommended parameters, as described in section 5.2. Problems with singular and nearly-singular blocks are handled as follows: If a diagonal block D_ν happens to be singular or numerically singular, i.e., if the numerical factorization fails, we modify the diagonal entries such that the block becomes diagonal dominant.

The problems described cannot occur for the class of H-matrices. Recall that a matrix A is termed an H-matrix, if there exist weights u_j, $j = 1,\dots,n$, $u_j > 0$ such that for all $j = 1,\dots,n$

$$|a_{ii}| \cdot u_i > \sum_{\substack{j=1,\dots,n \\ j\neq i}} |a_{ij}| \cdot u_j.$$

The class of H-matrices contains the class of M-matrices, and they arise, e.g., in certain discretizations of (elliptic) boundary value problems; see, e.g., [10]. We have the following result.

PROPOSITION 5.1. *Assume that $A \in \mathbb{R}^{n\times n}$ is an H-matrix or that A is symmetric and positive definite. Then M in* (4.2) *is nonsingular, i.e., D_ν is nonsingular for each $\nu = 1,\dots,m$.*

Proof. Assume first that A is an H-matrix. Then M is also an H-matrix (with the same weights u_j as for A). Therefore, the diagonal blocks D_ν of M are H-matrices as well and they are thus non-singular. Moreover, their LU factorizations can be performed without pivoting; see, e.g., [1].

If A is symmetric positive definite, each of the blocks D_ν, which are principal submatrices of A, are symmetric positive definite as well. In particular, they are all non-singular and they admit a Cholesky factorization and an LU factorization without pivoting. $\square$

In view of the maximum transversal transformation $A \to Q(RAC)$ and the two-sided XPABLO permutation $A \to PAP^T$ it is important to notice that the H-matrix property, symmetry and positive definiteness are preserved under the XPABLO permutation. In general, however, these properties will not be conserved under the transversal transformation, although the I-matrix scaling of MC64 will have a tendency to preserve the H-matrix property.

5.2. Choosing Parameters. We also have to consider carefully the choice of the XPABLO parameters and the actual criterion τ. A first important point in practice is that of fixing a maximum block size in Algorithm 1. Indeed, it may happen that with the criteria and the parameters in use, the algorithm would produce only one block, the whole matrix, and this is clearly not useful. We therefore fix a maximum block size, and we modify Algorithm 1 so that it closes the current block B as soon as it reaches this maximum size. We often also fix a minimum block size. This block size is achieved by merging adjacent blocks which are too small. Note that we do not merge blocks if we would exceed the maximum block size, even if we retain small blocks. For block Jacobi or block Gauss-Seidel preconditioning we recommend to set the minimum block size to 200 and the maximum block size to 1000.

The next, very crucial issue is that of finding adequate parameters and a suitable XPABLO criterion τ. Good parameters for each given matrix could be found by trying a variety of different parameter settings and then choosing the best one (with respect to the resources being used, time and storage). Of course, this is too costly in practice. We performed a long series of computations on many test problems, and we can give suggested values for most of the parameters resulting in run times of the preconditioned iterative methods which are quite close to the individually best ones. The suggested choices are for both block Jacobi and block Gauss-Seidel preconditioning are presented in Table 5.1. If this recommendations are used, the only difference between XPABLO and TPABLO1 is a small change in τ. XPABLO has τ = FC $\vee$ CC $\vee$ TCC and TPABLO1 has τ = (FC $\vee$ CC) $\wedge$ TCC. Nevertheless in almost all cases XPABLO is clearly to be preferred compared to TPABLO1.

Preconditioner	τ	α	β	γ	δ	ζ
block Jacobi	FC $\vee$ CC $\vee$ TCC	1.1	0.6	$\frac{\sum \lvert a_{ij}\rvert}{nz}$	0.05	$1/(2n)$
block Gauss-Seidel	FC $\vee$ TCC		–			

TABLE 5.1
Recommended XPABLO criterion τ and parameter values for block Jacobi and block Gauss-Seidel preconditioning.

As a last issue, we further discuss our choice for γ, the threshold parameter used in (TCC) and (TFC). Of all the parameter recommendations the one for γ is the weakest in the sense that it is the most likely candidate for not being a good choice for a particular problem. Therefore we present a different recommendation, which also gives good overall results and for some problems even better results than the default choice. After having scaled (and permuted) the matrix according to (2.1), it seems reasonable to use γ as a 'percentage' parameter, i.e., we chose γ such that a given fraction $\gamma' \in [0,1]$ of the nonzero elements are dropped when passing from A to $A^{>\gamma}$. Since we know the number $|E|$ of nonzeros in A, finding the corresponding γ is an instance of the *k-th largest element problem*: γ is the value of the k-th largest element of A with $k = \lfloor \gamma' |E| \rfloor$. Note that the k-th largest element problem can be solved with linear complexity, i.e., its cost is $\mathcal{O}(|E|)$; see, e.g., [15]. In practical computations, finding the average of the magnitude of the nonzeros is much simpler and faster. Experiments have shown that $\gamma' = 0.7$, i.e., using a γ such that 70% of the nonzero entries of the matrix have magnitude below γ, is a good choice for many problems.

6. Numerical Experiments. In this section we present numerical results using XPABLO as a tool for preconditioning. We start by showing the improvements of XPABLO over PABLO and TPABLO, and then provide a detailed comparison of XPABLO and ILUT as a tool for preconditioning. All experiments were run on an Intel Core 2 processor. The test programs are written in C, C++ and Fortran and were compiled using the GNU Compiler Collection (GCC). Identical optimization flags were used in all cases. This allows us to compare the execution times and not only the iteration counts. The test matrices are from the University of Florida Sparse Matrix collection [16]. Some basic properties of our test matrices are summarized in Table 6.1. For most matrices we use the provided right hand side b. If no right hand side is provided, we use $b := Ae$, where $e = (1, \dots, 1)^T$ is the vector of all ones.

Matrix	n	nz	Group
`CAVITY16`	4 562	137 887	DRIVCAV
`CAVITY26`	4 562	138 040	DRIVCAV
`EX19`	12 005	259 577	FIDAP
`EX35`	19 716	227 872	FIDAP
`GARON1`	3 175	84 723	Garon
`GARON2`	13 535	373 235	Garon
`RAEFSKY2`	3 242	293 551	Simon
`RAEFSKY3`	21 200	1 488 768	Simon
`SHYY41`	4 720	20 042	Shyy
`SHYY161`	76 480	329 762	Shyy
`BARRIER2-1`	113 076	2 129 496	Schenk ISEI
`BARRIER2-3`	113 076	2 129 496	Schenk ISEI
`IGBT3`	10 938	130 500	Schenk ISEI
`NMOS3`	18 588	237 130	Schenk ISEI
`OHNE2`	181 343	6 869 939	Schenk ISEI
`PARA-4`	153 226	2 930 882	Schenk ISEI
`PARA-8`	155 924	2 094 873	Schenk ISEI
`2D_54019_HIGHK`	54 019	486 129	Schenk IBMSDS
`3D_51448_3D`	51 448	537 038	Schenk IBMSDS
`IBM_MATRIX_2`	51 448	537 038	Schenk IBMSDS
`MATRIX_9`	103 430	1 205 518	Schenk IBMSDS

TABLE 6.1
Summary information on the test matrices

Preliminary tests had shown that the XPABLO framework can perform well for problems stemming from computational fluid dynamics (CFD) and semiconductor device simulation. Therefore the test matrices were specifically chosen from these application areas. In Table 6.1 the matrices above the horizontal line are CFD problems, the matrices below the line are device simulation problems. We note that PABLO was not designed specifically to solve these kind of problems. As noted in section 1, a modified version of PABLO has been used before for problems in Computational Fluid Dynamics (CFD) [24]. More information on solving systems from semiconductor device simulation can be found in [35].

To illustrate the different phases of the XPABLO framework, Figure 6.1 shows matrix plots of the matrix `GARON1` during four different phases. The bottom right plot shows the matrix of the system we are actually solving.

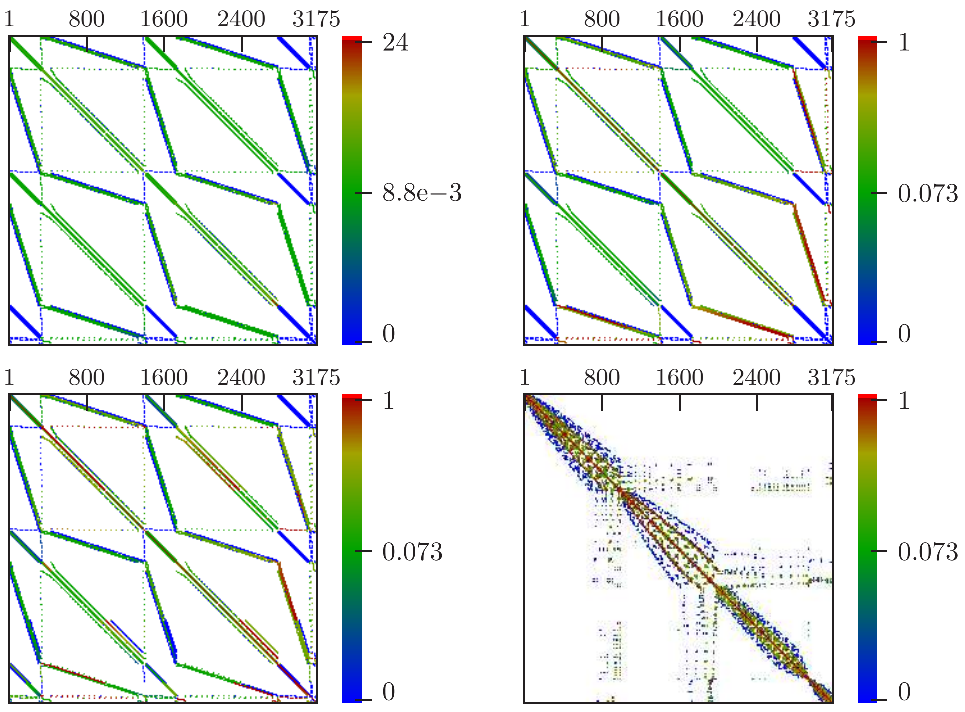

FIG. 6.1. *Matrix Plots of* `GARON1` *during four different phases of the XPABLO framework for preconditioning. Top left: The original matrix. Top right: The matrix scaled by MC64, but not yet permuted to be an I-matrix. Bottom left: After scaling and permuting by MC64. The matrix is now an I-matrix. Bottom right: The matrix after the XPABLO permutation. Note the difference in scale between the top left and the other plots.*

Table 6.2 shows how the execution times and iteration counts using the XPABLO framework compare to PABLO and TPABLO1 based preconditioning. Additionally the improvements gained by using MC64 instead of the default row-column scaling (RCS) is shown. RCS just scales the largest entry in each row and column to one. The following scaling and permutation algorithms are used:

P:	RCS + PABLO
T:	RCS + TPABLO1
X:	RCS + XPABLO
MX:	MC64 + XPABLO

All parameters are set to the values described in section 5, in particular γ is set to the average of all non-zero entries. Only the criterion τ is varied. For MX we additionally use XPABLO in conjunction with MC64. GMRES(50) is used as the solver. The preconditioner is the lower block triangular part of the scaled and permuted matrix, using the blocks found by (T/X)PABLO. A dash indicates that we did not observe convergence after 20 cycles of GMRES(50), i.e., after 1000 iterations. For each matrix the numbers in bold indicate the smallest execution time and the lowest iteration number. More details on the stopping criterion are given later when comparing XPABLO- and ILUT-based preconditioners. The robustness of XPABLO can be readily appreciated.

We now describe how we compare XPABLO with ILUT as a tool for preconditioning. We use ILUPACK [11] for our experiments with ILUT. The ILUT implementation

	Times in seconds				Total Iterations			
Matrix	P	T	X	MX	P	T	X	MX
`CAVITY16`	–	1.2	1.3	**0.59**	–	**1**	2	47
`CAVITY26`	–	–	1.3	**0.48**	–	–	**2**	23
`EX19`	–	–	5.5	**4.5**	–	–	506	**408**
`EX35`	–	–	–	**1.3**	–	–	–	**30**
`GARON1`	**0.24**	–	0.4	0.36	**2**	–	41	36
`GARON2`	–	–	3.9	**3.1**	–	–	171	**124**
`RAEFSKY2`	1.9	2	**1.1**	1.2	243	243	**46**	**46**
`RAEFSKY3`	–	–	15	**14**	–	–	310	**296**
`SHYY41`	0.14	0.13	0.11	**0.1**	25	20	**2**	**2**
`SHYY161`	**1.7**	1.7	2.7	1.7	11	**7**	48	14
`BARRIER2-1`	–	–	–	–	–	–	–	–
`BARRIER2-3`	–	–	–	–	–	–	–	–
`IGBT3`	–	–	1.5	**1.4**	–	–	138	**135**
`NMOS3`	–	6.1	**1.4**	1.5	–	548	**41**	47
`OHNE2`	–	–	–	**98**	–	–	–	**245**
`PARA-4`	–	–	**15**	42	–	–	**21**	170
`PARA-8`	–	–	–	**26**	–	–	–	**84**
`2D_54019_HIGHK`	–	–	**7.3**	7.7	–	–	**147**	160
`3D_51448_3D`	–	–	5.2	**4.8**	–	–	64	**55**
`IBM_MATRIX_2`	–	–	**4.6**	4.7	–	–	49	**47**
`MATRIX_9`	–	–	22	**16**	–	–	176	**100**

TABLE 6.2
XPABLO compared to PABLO and TPABLO

has two parameters, the drop tolerance τ and the elbow space ℓ. We write ILUT(τ, ℓ) to denote ILUT preconditioning with the specified parameters. In the experiments the elbow space is chosen large enough so that no overflow occurs. We denote this by writing ILUT(τ, ∞). In preliminary tests the three drop tolerances have been found to be good values for the selected problems. We use both XPABLO and ILUT in conjunction with MC64; see section 2. We precede ILUT additionally with a Reverse Cuthill-McKee ordering (RCM); this follows the recommendations in [7], [8] and [35]. We mention here that RCM is the default ordering in several software packages, e.g., in [37]. We do not use RCM together with XPABLO as our experiments have shown that it does not further improve the convergence.

As already mentioned, we use GMRES(50) as the iterative solver in all our experiments. The stopping criterion is

$$\|r_k\|/\|b\| < \sqrt{\epsilon}, \tag{6.1}$$

where r_k is the residual at the k-th iteration and ϵ is the machine epsilon, i.e., $\sqrt{\epsilon} \approx 10^{-8}$ since we use IEEE 754 double precision arithmetic. We also did tests with the alternative stopping criterion

$$\frac{\|r_k\|_2}{\sqrt{\|A\|_\infty \|A\|_1}\,\|x_k\|_2 + \|b\|_2} \leq \sqrt{\epsilon},$$

which is the default criterion in ILUPACK [11], based on the analysis in [3]; see also [33]. Since the results were very similar and the full test can not be done at each iteration we only show results using the relative residual stopping criterion (6.1). The solver was stopped if no convergence is reached after 20 cycles of GMRES(50), i.e., after 1000 iterations. We use the following preconditioners:

J:	MC64 + XPABLO + block Jacobi
LX:	MC64 + XPABLO + forward block Gauss-Seidel
UX:	MC64 + XPABLO + backward block Gauss-Seidel
I1:	MC64 + RCM + ILUT($10^{-2}, \infty$)
I2:	MC64 + RCM + ILUT($10^{-3}, \infty$)
I3:	MC64 + RCM + ILUT($10^{-4}, \infty$)

The results are summarized in Table 6.3. For XPABLO-based preconditioners we use the recommended parameters described in section 5. A dash indicates that we did not observe convergence. For each matrix the bold numbers indicate the smallest execution time and the lowest iteration number.

	Times in seconds						Total Iterations					
	XPABLO			ILUT			XPABLO			ILUT		
Matrix	J	LX	UX	I1	I2	I3	J	LX	UX	I1	I2	I3
CAVITY16	1.4	**0.59**	–	–	1.5	1.3	193	47	–	–	88	**27**
CAVITY26	3.6	**0.48**	–	–	–	1.3	654	23	–	–	–	**21**
EX19	–	4.5	**1**	–	–	–	–	408	**28**	–	–	–
EX35	1.5	1.3	**1.1**	–	–	–	39	30	**12**	–	–	–
GARON1	0.68	**0.36**	2.5	–	0.74	0.69	90	**36**	703	–	101	45
GARON2	7.6	**3.1**	–	–	–	–	350	**124**	–	–	–	–
RAEFSKY2	3.5	1.2	0.86	**0.61**	2	5.2	223	46	23	28	13	**7**
RAEFSKY3	35	14	**4.2**	5.3	9.1	22	737	296	44	92	14	**7**
SHYY41	1.2	0.1	–	**0.09**	0.12	0.1	739	**2**	–	68	70	61
SHYY161	–	1.7	**1.6**	–	–	–	–	14	**1**	–	–	–
BARRIER2-1	–	–	**8.8**	–	–	–	–	–	**13**	–	–	–
BARRIER2-3	–	–	**9.1**	–	–	–	–	–	**14**	–	–	–
IGBT3	3.4	1.4	0.64	0.83	**0.42**	0.43	334	135	27	115	35	**23**
NMOS3	3.5	1.5	1.6	0.78	**0.7**	0.78	173	47	61	51	27	**16**
OHNE2	210	98	**20**	60	46	69	531	245	**10**	218	85	42
PARA-4	81	42	**13**	20	20	–	369	170	**15**	161	91	–
PARA-8	61	26	**13**	14	15	27	248	84	**18**	131	74	44
2D_54019_HIGHK	17	7.7	2.7	**0.73**	2.2	13	347	160	13	13	**7**	21
3D_51448_3D	8.3	4.8	**3.1**	–	–	–	131	55	**14**	–	–	–
IBM_MATRIX_2	7.6	4.7	**3.2**	15	94	180	119	47	15	**1**	**1**	**1**
MATRIX_9	33	16	**8.2**	12	41	190	253	100	22	**1**	**1**	**1**

TABLE 6.3

Total CPU times for solving a linear system using preconditioned GMRES(50) including all preprocessing and the corresponding iteration numbers.

As it can be observed, the XPABLO-based preconditioners can perform better than the ILUT-based ones in many cases, and in some cases much better. In some examples the XPABLO-based preconditioners perform better, although the iteration counts of the ILUT-based preconditioners are lower. The reason for this is that times for finding and, more importantly, factorizing the preconditioner are not the same in both cases.

7. Conclusion. While XPABLO is not always competitive with sophisticated ILU preconditioners like ILUT, we have seen that XPABLO can perform better than ILUT for many CFD and semiconductor device simulation problems. Furthermore, XPABLO-based preconditioners have much more parallelization potential than ILU-based preconditioners.

Acknowledgment. We thank Michele Benzi and the anonymous referee for their constructive comments. Their help and suggestions are very much appreciated.

REFERENCES

[1] G. Alefeld, *Über die Durchführbarkeit des Gaußschen Algorithmus bei Gleichungen mit Intervallen als Koeffizienten (On the feasibility of the Gaussian algorithm for equations with interval coefficients)*, Computing Supplementum, 1 (1977), pp. 15–19. In German.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, SIAM, Society of Industrial and Applied Mathematics, Philadelphia, third ed., 1999.

[3] M. Arioli, I. Duff, and D. Ruiz, *Stopping criteria for iterative solvers*, SIAM Journal on Matrix Analysis and Applications, 13 (1992), pp. 138–144.

[4] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, Journal of Computational Physics, 182 (2002), pp. 418–477.

[5] M. Benzi, H. Choi, and D. B. Szyld, *Threshold ordering for preconditioning nonsymmetric problems*, in Scientific Computing, Proceedings of the Workshop, 10–12 March 1997, Hong Kong, G. Golub, S.-H. Lui, F. Luk, and R. Plemmons, eds., Singapore, 1997, Springer, pp. 159–165.

[6] M. Benzi, J. C. Haws, and M. Tůma, *Preconditioning highly indefinite and nonsymmetric matrices*, SIAM Journal on Scientific Computing, 22 (2000), pp. 1333–1353.

[7] M. Benzi, W. Joubert, and G. Mateescu, *Numerical experiments with parallel orderings for ILU preconditioners*, Electronic Transactions on Numerical Analysis, 8 (1999), pp. 88–114.

[8] M. Benzi, D. B. Szyld, and A. van Duin, *Orderings for incomplete factorization preconditionings of nonsymmetric problems*, SIAM Journal on Scientific Computing, 20 (1999), pp. 1652–1670.

[9] M. Benzi and M. Tůma, *Orderings for factored approximate inverse preconditioning*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1851–1868.

[10] A. Berman and R. J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences*, Classics in Applied Mathematics, SIAM, Philadelphia, USA, 1994. Originally published by Academic Press, New York, 1979.

[11] M. Bollhöfer and Y. Saad, *ILUPACK - preconditioning software package, release v1.0, May 14, 2004*. Available online at `http://www.tu-berlin.de/ilupack/`.

[12] A. Bunse-Gerstner and R. Stöver, *On a conjugate gradient-type method for solving complex symmetric linear systems*, Linear Algebra and its Applications, 287 (1999), pp. 105–123.

[13] H. Choi and D. B. Szyld, *Application of threshold partitioning of sparse matrices to Markov chains*, in IEEE International Computer Performance and Dependability Symposium IPDS'96, Urbana-Champaign, Illinois, Sept. 1996, pp. 158–165.

[14] H. Choi and D. B. Szyld, *Threshold ordering for preconditioning nonsymmetric problems with highly varying coefficients*, Tech. Report 96-51, Department of Mathematics, Temple University, Philadelphia, May 1996.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, Cambridge, MA: MIT Press, Second ed., 2001.

[16] T. A. Davis, *University of Florida sparse matrix collection*. Available online at `http://www.cise.ufl.edu/research/sparse/matrices/`.

[17] ———, *Algorithm 8xx: UMFPACK: an Unsymmetric-Pattern Multifrontal Method*, ACM Transactions on Mathematical Software, 30 (2004). Available online at `http://www.cise.ufl.edu/research/sparse/umfpack/`.

[18] I. S. Duff, *MA57 - a code for the solution of sparse symmetric definite and indefinite systems*, ACM Transaction on Mathematical Software, 30 (2004), pp. 118–144.

[19] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Monographs on Numerical Analysis, Clarendon Press, Oxford, UK, 1986.

[20] I. S. Duff and J. Koster, *The design and use of algorithms for permuting large entries to the diagonal of sparse matrices*, SIAM Journal on Matrix Analysis and Applications, 20 (1999), pp. 889–901.

[21] ———, *On algorithms for permuting large entries to the diagonal of a sparse matrix*, SIAM Journal on Matrix Analysis and Applications, 22 (2001), pp. 973–996.

[22] I. S. Duff and J. A. Scott, *A parallel direct solver for large sparse highly unsymmetric linear systems*, ACM Transaction on Mathematical Software, 30 (2004), pp. 95–117.

[23] L. C. Dutto, *The effect of ordering on preconditioned GMRES algorithm, for solving the compressible Navier-Stokes equations*, International Journal on Numerical Methods in Engineering, 36 (1993), pp. 457–497.

[24] L. C. Dutto, W. G. Habashi, and M. Fortin, *Parallelizable block diagonal preconditioners for the compressible Navier-Stokes equations*, Computer Methods in Applied Mechanics and Engineering, 117 (1994), pp. 15–47.

[25] S. C. Eisenstat, *Efficient implementation of a class of preconditioned conjugate gradient methods*, SIAM Journal on Scientific and Statistical Computing, 2 (1981), pp. 1–4.

[26] S. Fischer, *Über auf Takagi-Faktorisierungen beruhende Präkonditionierer für CSYM (On Takagi-factorization based preconditioners for CSYM)*, PhD thesis, Department of Mathematics and Science, University of Wuppertal, Germany, March 2006. In German.

[27] S. Fischer and A. Frommer, *Preconditioning CSYM.* In preperation.

[28] D. Fritzsche, *Graph theoretical methods for preconditioners*, master's thesis, Department of Mathematics and Science, University of Wuppertal, Germany, June 2004.

[29] A. Greenbaum, *Iterative methods for solving linear systems*, vol. 17 of Frontiers in Applied Mathematics, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.

[30] J. O'Neil and D. B. Szyld, *A block ordering method for sparse matrices*, SIAM Journal on Scientific and Statistical Computing, 11 (1990), pp. 811–823.

[31] J. M. Ortega, *Efficient implementations of certain iterative methods*, SIAM Journal on Scientific and Statistical Computing, 9 (1988), pp. 882–891.

[32] ———, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York, 1988.

[33] C. C. Paige and Z. Strakos, *Residual and backward error bounds in minimum residual Krylov subspace methods*, SIAM Jourbal on Scientific Computing, 23 (2002), pp. 1898–1923.

[34] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, second ed., 2003.

[35] O. Schenk, S. Röllin, and A. Gupta, *The effects of unsymmetric matrix permutations and scalings in semiconductor device and circuit simulation*, IEEE Transactions on CAD of Integrated Circuits and Systems, 23 (2004), pp. 400–411.

[36] V. Simoncini and D. B. Szyld, *Recent computational developments in Krylov subspace methods for linear systems*, Numerical Linear Algebra with Applications, 14 (2007), pp. 1–59.

[37] R. S. Tuminaro, M. Heroux, S. A. Hutchinson, and J. N. Shadid, *Official Aztec user guide, version 2.1*, Tech. Report SAND99-8801J, Sandia National Laboratory, Albuquerque, New Mexico, November 1999. Available online at `http://www.cs.sandia.gov/CRF/aztec1.html`.