How fast the Laplace equation was solved in 1995

E.F.F. Botta, K. Dekker, Y. Notay, A. van der Ploeg, C. Vuik,
F.W. Wubs, P.M. de Zeeuw

University of Groningen
Department of Mathematics

Report W-9701
January 1997

How fast the Laplace equation was solved in 1995

E. F. F. Botta[a], K. Dekker[b], Y. Notay[c],
A. van der Ploeg[d], C. Vuik[b], F. W. Wubs[a], P. M. de Zeeuw[d]

[a] Dept. of Mathematics, University of Groningen, P.O. Box 800, 9700 AV Groningen

[b] Dept. of Mathematics, University of Delft, P.O. Box 5031, 2600 GA Delft

[c] Service de Métrologie Nucléaire, Université Libre de Bruxelles (C.P. 165),
50, Av. F.D. Roosevelt, B-1050 Brussels, Belgium

[d] Dept. of Numerical Mathematics, CWI, P.O. Box 94079, 1090 GB Amsterdam

Abstract

On the occasion of the third centenary of the appointment of Johann Bernoulli at the University of Groningen, a number of linear systems solvers for some Laplace-like equations have been compared during a one-day workshop. CPU-times of several advanced solvers measured on the same computer (an HP-755 workstation) are presented, which makes it possible to draw clear conclusions about the performance of these solvers.

1 Introduction

To mark the 300th anniversary of the appointment of Johann Bernoulli as professor at the University of Groningen a number of scientific activities occurred including a one-day workshop entitled "Laplace Symphony". The subject of this workshop was the comparison of solvers for a number of Laplace-like equations. These equations arise in many subject areas using mathematical models, for example, in fluid dynamics, electro-magnetics and astrophysics. The numerical solution of these time-dependent and/or non-linear partial differential equations requires the repeated solution of large sparse systems of linear equations. In an attempt to provide more insight about the efficiency of linear solvers for discretised PDE's, it is relevant to compare them for discretised Laplace operators.

Of course, it is neither practical nor sensible to cover all aspects of efficient implementation on scalar, vector and parallel computers. Therefore, we focus on the computing time and the memory usage. The investigation is made on an HP-755 workstation enabling us to perform direct comparison of the methods.

Section 2 summarizes the six test cases studied at the workshop. A short description of each method is given in Section 3. The CPU-time comparison of the methods is presented in Section 4. In the last section, we draw some conclusions.

2 Test cases

In the first five cases, the system of linear equations $Au = b$ to be solved has a matrix A which is symmetric and positive (semi-)definite. The remaining one has a mild non-symmetry due to a symmetry disturbing boundary treatment.

The first two problems follow from the discretisation of the Laplace equation on a uniform grid in 2 and 3 dimensions, respectively. These are adopted because a large number of methods are applicable to these problems including those that directly exploit symmetry, e.g., cyclic reduction and FFT approaches.

In order to study grid effects, the third problem concerns a Laplace operator discretised on a highly stretched grid. Reference [4] shows that the convergence rate of many commonplace iterative methods deteriorates for such problems. Similar convergence deterioration is seen for problems with strong discontinuities in the coefficients, an example of which has been included as test problem four. The fifth problem arises from a higher-order discretisation on an unstructured grid. This test matrix has a very irregular sparsity pattern and lacks the diagonal dominance property.

The last problem deals with a system of linear equations with a non-symmetric coefficient matrix. Reference [26] shows that preconditioning is needed in order to solve the problem in a reasonable time making it an attractive test for the preconditioners we consider.

For each problem the right-hand side is the zero vector and a starting vector is given. (Of course, all solvers were able to deal with general right-hand side vectors and general starting vectors.) It was not allowed to assume and exploit symmetry in the (trivial) solution. In each case the stopping criterion for the iteration is

$$u_{max}^{(n)} - u_{min}^{(n)} < 10^{-6}(u_{max}^{(0)} - u_{min}^{(0)}) \tag{1}$$

We now give a more detailed description of the test cases:

1. *Uniform-2D.* The Laplace equation on the unit square with boundary condition $u = 0$ everywhere and a standard five-point discretisation on a uniform grid with mesh size $1/M$, $M = 256, 512, 1024$. The starting vector is the function

$$u(x, y) = (xy(1 - x)(1 - y))^2 e^{x^2 y} \tag{2}$$

evaluated at the grid points.

2. *Uniform-3D.* The Laplace equation on the unit cube with condition $u = 0$ on the entire boundary and a standard seven-point discretisation on a uniform grid with mesh size $1/M$, $M = 24, 48, 96$. The starting vector is the function

$$u(x, y, z) = (xyz(1-x)(1-y)(1-z))^2 e^{x^2yz} \tag{3}$$

evaluated at the grid points.

3. *Stretched.* The Laplace equation on the unit square with boundary conditions $\frac{\partial u}{\partial n} = 0$ everywhere and a standard five-point discretisation on a non-uniform $(M+1) \times (M+1)$-grid, $M = 128, 256, 512$. In the neighborhood of the boundaries the grid is refined in such a way that the ratio of maximum mesh size to minimum mesh size is equal to 1000 and the ratio of subsequent mesh sizes is kept constant. The starting vector is again given by (2). Due to the Neumann boundary conditions, the solution is determined up to a constant, hence the coefficient matrix is singular.

   This problem is of interest for unsteady incompressible Navier-Stokes solvers, where at each time step the pressure has to be computed from a Poisson equation for which a system with a discretized Laplace operator needs to be solved.

4. *Discontinuous.* The stationary diffusion equation

$$-\frac{\partial}{\partial x}(D(x,y)\frac{\partial}{\partial x}u(x,y)) - \frac{\partial}{\partial y}(D(x,y)\frac{\partial}{\partial y}u(x,y)) = 0 \tag{4}$$

   on the unit square with boundary condition $\frac{\partial u}{\partial n} = 0$ everywhere and a standard five-point discretisation on a uniform grid. The function $D(x, y)$ is defined by

$$D(x,y) = 10000 \text{ for } x \leq 0.3 \ \wedge \ y \leq 0.8,$$
$$D(x,y) = 1 \text{ elsewhere.}$$

   The number of mesh intervals in one direction is a multiple of 10, hence the boundaries of areas with different diffusion coefficients coincide with grid lines. Since the discretisation, like the underlying PDE, is conservative it is well defined at these boundaries. The mesh-size is $1/400$, hence there are $(401)^2$ unknowns. The starting vector is again (2). Discontinuous coefficients arise, for example, in equations for semi-conductors and reservoir simulations.

5. *Unstructured.* The system arises from a finite-element discretisation of the Laplace equation on the domain shown in Fig. 1.

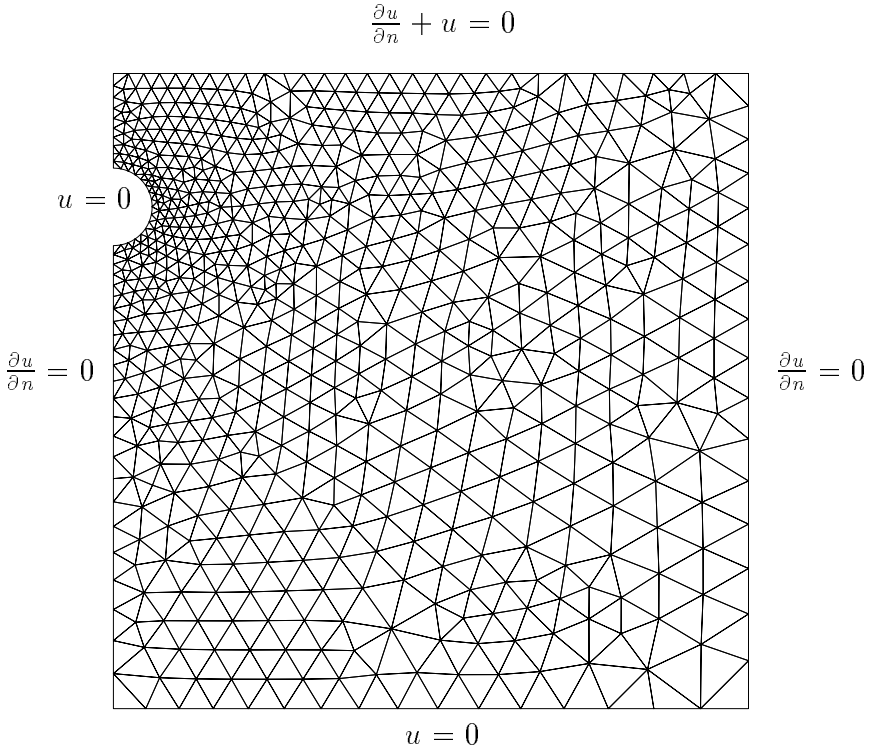

Figure 1: The domain and the boundary conditions for Problem 5.

This figure also shows the boundary conditions and a coarse finite element mesh. We use a refinement of this mesh leading to a matrix of dimension 25759 and 147155 non-zero entries. This problem comes from a simplified model of the temperature distribution in the ground near a gas pipe. The discretisation is performed by the finite element package SEPRAN using quadratic isoparametric triangles. This implies that the coefficient matrix is certainly not an M-matrix, since it has both positive and negative entries outside the main diagonal.

6. *Non-symmetric.* This problem (proposed by C. Vuik [26]) concerns a pressure calculation for the incompressible Navier-Stokes equations. For the discretisation a finite volume technique is used combined with boundary fitted coordinates. This results in a structured matrix with at most 9 non-zero elements per row. The matrix looks like a discretisation of a Laplace equation, however it is weakly non-symmetric due to the treatment of the boundary conditions. The physical domain and a coarse finite volume grid are given in Fig. 2. Neumann boundary conditions are posed on Boundary 1, 2, and 3, whereas on Boundary 4 a Dirichlet condition is used. The problem is solved on an $M \times 4M$-grid with $M = 16, 32, 64, 128$.

3 Description of solvers

The solvers used in the comparison will be shortly described in this section. We start with three public domain methods, of which the first two are direct solvers. These will be followed by solvers developed by the authors.

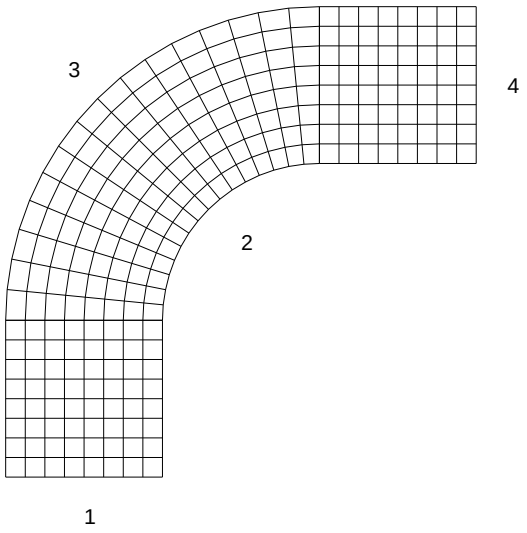

Figure 2: The domain and grid for the non-symmetric problem

3.1 FISHPAK

FISHPAK is a package for the solution of separable elliptic partial differential equations (implemented by Adams, Swarztrauber and Sweet, 1980). It can handle periodic, Neumann and Dirichlet boundary conditions only. In two dimensions it consists of a generalization of the Buneman cyclic reduction algorithm [6] developed in the 1970s. In contrast to Buneman's algorithm this variant [21] can handle arbitrary grid-sizes and the computation time shows a much smoother behavior as a function of the grid-size than the FFT (Fast Fourier Transform) based approach. However, in three dimensions FFTs are used in order to reduce the problem to a set of tridiagonal problems. For this task it makes use of routines from FFTPACK of Swarztrauber (version April 1985). The computation time of the 3D case depends strongly on the grid-sizes, as FFTs are fast only if the size of the vector on which it operates is a highly composite number. This is the case for Problems 1 and 2.

The memory required is about 1 real per unknown in both 2D and 3D, hence only 8 bytes per unknown. This is due to the fact that the problem is separable so only work arrays of a lower dimension need to be stored.

3.2 UMFPACK/SuperLU

UMFPACK is a package for solving non-symmetric systems of linear equations with arbitrary sparsity pattern. It is based on a combined unifrontal/multifrontal algorithm that enables a general fill-in reduction ordering to be applied. In the tests, version 2.0 (September 1995) [7] is used. We also considered the package SuperLU (October 1995) which has the same functionality as UMFPACK. It uses so-called supernodes and BLAS to optimize performance [9]. Both methods aim to reduce the data movement in order to exploit the cache. In order to limit the number of results we only show results of UMFPACK. The performance of SuperLU is comparable. The memory requirement of these methods is

proportional to the fill in the decomposition and can become rather large. They even run out of memory (1/2 gigabyte) for the larger problems.

3.3 ILUT(SPARSKIT)

ILUT is a preconditioner supplied in SPARSKIT [19] which uses a dual thresholding strategy for dropping elements. The strategy works as follows

1. At a certain stage in the decomposition any element whose size is less than some user specified tolerance *tol* (relative to the norm of the current row in U) is dropped.

2. The fill of each row of L and U is limited by a user specified number $lfil$. Only the largest $lfil$ elements in each row of L and U are kept.

ILUT can handle non-symmetric systems and arbitrary sparsity patterns. It does not make any reordering of the unknowns. (Though their exists a version with partial pivoting.) In our case we used $lfil = 10$ and *tol* was set to 10^{-3} and 10^{-7} for Problems 1,2 and Problems 3-6, respectively. In all cases this led to the maximum fill-in per row of 2 times $lfil$ hence approximately 20. This preconditioner is used within Bi-CGSTAB which is also supplied in SPARSKIT. The memory needed is about 33 double precision reals and 25 integers per unknown hence in total 364 bytes per unknown.

3.4 ICCG with diagonal and hyperplane ordering

ICCG uses the Conjugate Gradient method with a (relaxed) incomplete Cholesky-decomposition without fill-in as a preconditioner [24]. The rate of convergence slightly depends on a parameter α in this decomposition. The choice $\alpha = 0$ yields the classical incomplete decomposition, $\alpha = 1$ gives Gustafsson's modification. We do not try to optimise for α, but choose $\alpha = 0.98, 0.98, 0.90, 0.95$ for the first four problems, respectively.

In the 2D-problems (1, 3, 4) we reorder the unknowns in the forward and backward substitutions implicitly along the diagonals of the grid. This amounts to a frontal approach in the solution process. All computations for the unknowns on a diagonal can be performed independently, which may result in high execution rates on vector and parallel machines. However, implicit reordering amounts to loops with large strides, and a probably inefficient use of the cache on workstations. This technique is, of course, only applicable to structured grids in two dimensions.

In the structured 3D-problem (*Uniform-3D*) the unknowns are explicitly reordered along hyperplanes. As all computations on a hyperplane can be done in parallel, this approach results in high performance on parallel and vector computers [23, 8]. Here, the loops in the forward/backward substitution process have stride one, but the computations involve some indirect addressing. Nevertheless, data locality is obtained by the explicit reordering, which leads to a more favourable use of the cache. We apply Eisenstat's implementation [10] in order to avoid multiplication with the matrix A.

The amount of memory used is fixed for each problem. In the 2D-problems we need 9 double precision numbers (72 bytes) per unknown, including storage for the elements of A and the solution x. In the 3D-problem we use 8 index arrays for indirect adressing

in addition to 10 double precision numbers per unknown. This amounts to 112 bytes per unknown. In case memory is a bottleneck, the storage required can be reduced to 84 bytes per unknown by recomputation of 7 index arrays in each iteration. The overhead involved is negligible on scalar processors.

3.5 MILU-*rrb*, MILU-*rrb&rcm* and DRIC

The solver presented in this section essentially relies on the module ITSOL [16] developed at the University of Brussels and now available publicly. This module solves sparse positive (semi-)definite linear systems by the conjugate gradient method and makes use of a variety of incomplete LU based preconditioners. It was challenging to test them on the proposed problems. (ITSOL is essentially a research tool for testing the numerical efficiency of preconditioners, i.e., little effort has been done to optimize the code in terms of computing time.)

For the test problems *Uniform-2D* and *Discontinuous* we used a preconditioner which consists in performing a MILU factorization with respect to a recursive red-black (*rrb*) ordering of the unknowns, fill-in entries being accepted provided that the red unknowns in a same level remain uncoupled. This method originates from [1, 5] and has recently been proved of near optimal order [17]. The method proved easy to implement requiring little memory. (The preconditioner needs less than twice the amount of space needed to store the system matrix.) Including the iteration vectors and indirect addressing vectors, the memory requirement is about 16 words of 8 bytes (i.e. 128 bytes) per unknown.

The method offers a nice compromise between efficiency and ease of implementation, but is not applicable to 3D or unstructured problems. It also performs poorly if the PDE is anisotropic or if, like in the problem *Stretched*, anisotropy is introduced at the discretization level. For this problem, an efficient method was obtained by using the observation that the MILU factorization produces a good preconditioner for such cases provided one uses a reverse Cuthill-McKee (*rcm*) type ordering and accepts a modest level of fill-in [13]. This leads us to reorder the matrix according to the *rrb* approach in regions where the local mesh sizes are not much different, and according to an *rcm*-like algorithm in the highly stretched regions. By using the appropriate fill-in strategy in each region, this yields a preconditioner which performs like the purely MILU-*rrb* does on isotropic problems. Memory requirements are similar and in some cases even slightly smaller with the mixed method. However, we confess that it is difficult to find an ordering algorithm which would allow the use of this combined approach in a "black box" fashion. From this point of view, it is worth noting the recent progress made in [15], where a modification of MILU-*rrb* factorizations is proposed to make them robust with respect to anisotropy and/or grid stretching in a way totally transparent for the user.

Finally, for the problems *Uniform-3D* and *Unstructured*, we used a less efficient but more "general purpose" method, namely a perturbed MILU preconditioner with perturbations added automatically according to the DRIC algorithm [14]. Natural ordering was used for the problem *Uniform-3D*, for which much better CPU times would have been obtained with a code using direct addressing. For the problem *Unstructured*, we reorder the unknowns according to a variant of the *rcm* algorithm (see [13]). Instead of factorizing the system matrix, which might be unstable or lead to poor performances, we factorize

the M-matrix obtained by deleting the positive off-diagonal entries and subtracting them from the diagonal (so as to preserve the row-sum); we refer to [20] and the references therein for an analysis of this reduction technique. For the problem *Uniform 3D*, the memory requirement is as in previous cases about 128 bytes per unknown, whereas, for the problem *Unstructured*, it is about 206 bytes per unknown because the system matrix is somewhat denser.

3.6 NGILU and MRILU

NGILU (Nested Grids ILU) combines an incomplete LU-decomposition with a reordering similar to the partitioning of unknowns in multi-grid, with the objective to obtain grid-independent convergence, [22]. This technique is only applicable to structured grids. It has been used for the first, second and fourth test problem.

For the other test problems we have used a more general renumbering which is determined during the factorization and based upon the matrix instead of the underlying grid. In the sequel of this paper, the resulting preconditioning technique is denoted by MRILU (Matrix Renumbered ILU) [3]. Below we give a brief description of this method.

In the first step of the algorithm we reduce the system by using a renumbering such that the coefficient matrix obtains the block structure,

$$\begin{pmatrix} D_1 & S_{12} \\ S_{21} & S_{22} \end{pmatrix} \tag{5}$$

in which D_1 is a diagonal matrix with a dimension as large as possible. This partitioning can be done for matrices A having an arbitrary sparsity pattern (see, for example, [18]).

At step k, the unknowns are permuted in such a way that the $k-$th system of linear equations $S_k x = b$ can be partitioned as

$$\begin{pmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \tag{6}$$

in which the block S_{11} is strongly diagonally dominant. Next, this block is approximated by a diagonal matrix P with the same row sums. With this approximation x_1 can easily be eliminated and the reduced system is given by the Schur-complement $S_{k+1} = S_{22} - S_{21}P^{-1}S_{12}$. This approach can be combined with a dropping strategy as described in [22]: any element in S_{21} or S_{12} which is smaller than a given threshold parameter is lumped on the main diagonal. We continue the approach described above until we obtain a Schur-complement that is small enough to be solved with a standard method and so we finally arrive at the incomplete factorization $S_1 = LU + R$ where R is called the residual matrix.

It appears that a successful approach is to demand that the row sums of $|R|$ do not exceed a preset threshold parameter ε. Hence the lumping strategy and partitioning of the Schur-complements is done in such a way that $\sum_j |r_{ij}| \leq \varepsilon$ for each i.

For the first five test problems, the resulting preconditioning technique is combined with the Conjugate Gradient method. The last test problem is solved with preconditioned Bi-CGSTAB.

NGILU used about 140 bytes per unknown for the problems where it is applied (Problems 1,2 and 4). As it is based on a structured grid it needs in general less memory than

MRILU. However, for Problem 3 (*Stretched*) MRILU takes advantage of the structure in the matrix and also needs only 140 bytes per unknown. For Problem 5 and 6 MRILU needs about 240 and 430 bytes per unknown, respectively. The higher memory requirements for the last problem are due to the non-symmetry. Of course, the needed memory depends on the choice of the dropping parameter. It can be decreased at the cost of some extra iterations.

3.7 MGD9V

This code is intended for a class of problems that is wider than the Laplace equation. Its scope is the solution of linear systems resulting from the 9-point discretisation of the following general linear second-order elliptic partial differential equation in two dimensions:

$$Lu \equiv -\nabla \cdot (D(x)\nabla u(x)) + b(x) \cdot \nabla u(x) + c(x)u(x) = f(x) \tag{7}$$

on a bounded domain $\Omega \subset \mathbb{R}^2$ with suitable boundary conditions. $D(x)$ is a positive definite 2×2 matrix function and $c(x) \geq 0$. D and c are allowed to be discontinuous across an interface in Ω. The user supplies the discretisation of (7) (e.g. by a finite element or finite volume technique):

$$L_n \overline{u}_n = f_n \tag{8}$$

where $\overline{u}_n$ and f_n are grid-functions defined on the grid Ω_n. The convection is allowed to be dominant; roughly speaking $h\|b\| > \|D\|$. The code performs only for the scalar case and within the constraints of a regular domain and a structured grid. The code is of black-box type: no interference of the user with the algorithm is required beyond specification of the discrete system of equations and its right-hand side. Incomplete line LU-factorization (ILLU), also called *incomplete block LU-factorization*, is used as basic iterative method (for a description see [28] and the references mentioned there). Like for other basic iterative methods, the convergence of ILLU on its own is slow for low-frequent components in the residual. The algorithm of MGD9V is a multigrid method: it accelerates a basic iterative technique by coarse grid corrections, resolving the low-frequent components on coarser grids with increasing mesh-size. Let u_n be an approximation of $\overline{u}_n$, the coarse grid correction (CGC) then reads:

$$d_{n-1} = R_{n-1}(f_n - L_n u_n); \tag{9a}$$

$$\text{solve } L_{n-1} e_{n-1} = d_{n-1}; \tag{9b}$$

$$\tilde{u}_n = u_n + P_n e_{n-1}. \tag{9c}$$

R_{n-1} is the restriction operator that transfers the residual from the fine grid Ω_n onto the coarse grid Ω_{n-1}, P_n is the prolongation operator that transfers a correction for the solution from the coarse to the fine grid. Once the prolongation has been defined, we choose $R_{n-1} \equiv P_n^T$. The operator L_{n-1} is defined by the sequence of operations

$$L_{n-1} = R_{n-1} L_n P_n. \tag{10}$$

The code computes the coarse grid matrix of L_{n-1}, thus relieving the user of this task. After the CGC the residual consists of short wavelength components only, which are

reduced efficiently by (ILLU) relaxation. This completes one so-called multigrid cycle. The algorithm is applied in a recursive manner with respect to the solution of (9b).

Bilinear interpolation would be the obvious choice for the prolongation. However, this yields an excruciatingly slow algorithm in the case of discontinuous diffusion coefficients. Therefore, an alternative prolongation P_n has been constructed that satisfies jump conditions across interfaces. Moreover, we let it handle the case of dominant advection by means of biased interpolation (of upwind-type, this way the mesh Péclet number is preserved when the grid is coarsened). These improvements have been achieved by extracting the necessary information implicitly stored in L_n. This causes the prolongation P_n to be different at each grid-point. The above makes the efficient implementation of (10) a non-trivial task. A full description of the whole algorithm can be found in [29]. The storage requirements of MGD9V amount to $\frac{68}{3}$ reals or 181 bytes per unknown (matrix, right-hand side and solution included).

3.8 The RILU_fill preconditioner

We describe the RILU_fill preconditioner for the non-symmetric problem. The coefficient matrix of this problem has 9 non-zero elements per row. If the number of finite volumes in the x_1 direction is denoted by n_1, then the following elements of the pressure matrix A are possibly non-zero:

$$a_{ij} \neq 0 \text{ for } j - i \in \{-n_1 - 1, -n_1, -n_1 + 1, -1, 0, 1, n_1 - 1, n_1, n_1 + 1\}.$$

The *RILU_fill* preconditioner is based on an Incomplete LU decomposition [12]. The amount of fill-in, which is allowed, can be varied by the parameter n_{fill}. If $n_{\text{fill}} = 0$ then the lower triangular matrix L and the upper triangular matrix U are such that the non-zero pattern of $L + U$ is the same as the non-zero pattern of A [26]. Suppose $n_{\text{fill}} > 0$ and n_{fill} is even, then the set $P_{n_{\text{fill}}}$ is defined as

$$P_{n_{\text{fill}}} = [-n_1 - 1, -n_1 + 1 + \frac{n_{\text{fill}}}{2}] \cup [-1 - \frac{n_{\text{fill}}}{2}, 0] \ .$$

The non-zero elements of L and U are:

$$l_{ij} \neq 0 \text{ for } j - i \in P_{n_{\text{fill}}} \text{ and } u_{ij} \neq 0 \text{ for } i - j \in P_{n_{\text{fill}}} \ .$$

Note that $9 + 2 * n_{\text{fill}}$ extra vectors are needed to store the preconditioner.

The main diagonal elements of L are equal to 1. In ILU_fill the remaining elements of L and U are calculated by the following rule:

$$(LU)_{ij} = a_{ij} \text{ for all } i - j \in P_{n_{\text{fill}}} \text{ and } j - i \in P_{n_{\text{fill}}} \backslash \{0\}.$$

For problems where the solution is a smoothly varying function, it is a good idea to use the Modified ILU preconditioner [11], or the Relaxed ILU preconditioner [2] instead of the classic ILU preconditioner [12]. The RILU_fill(α) preconditioner is an average of the ILU_fill and MILU_fill preconditioner. In the MILU_fill preconditioner the following rules are used:

Table 1: The methods used in the comparison

Not.	Description	Problem					
		1	2	3	4	5	6
D	ICCG with diagonal (2D) or hyperplane ordering (3D) (Dekker)	*	*	*	*		
F	FISHPAK (public domain)	*	*				
MR	MRILU (Botta, v/d Ploeg and Wubs)			*		*	*
NG	NGILU (Botta, v/d Ploeg and Wubs)	*	*		*		
N1	MILU-*rrb* (Notay)	*			*		
N2	DRIC (Notay)		*			*	
N3	MILU-*rrb*&*rcm* (Notay)			*			
S	ILUT/Bi-CGSTAB (SPARSKIT, public domain)	*	*	*	*	*	*
U	UMFPACK (public domain)	*	*	*	*	*	*
V	RILU_fill (Vuik)						*
Z	MGD9V (De Zeeuw)			*	*		

$$\text{rowsum}(LU)_i = \text{rowsum}(A)_i, \text{ and}$$
$$(LU)_{ij} = a_{ij} \text{ for all } j-i, i-j \in P_{n_{\text{fill}}} \backslash \{0\}.$$

In [27] numerical experiments are presented to compare RILU_fill(α) with different choices of n_{fill} and α.

The results reported in Section 4 are obtained with Bi-CGSTAB [25] as the iterative method and RILU_fill(α) as the preconditioner with the choices: $n_{\text{fill}} = 8$ and $\alpha = 1$. The memory requirements per unknown for this combination are: matrix: 9 entries, right-hand side: 1, Bi-CGSTAB: 10, and the preconditioner: 25. This implies that the method needs $45 \times 8 = 360$ bytes per unknown.

4 CPU-times per unknown

In this section, we present the CPU-times for several solvers for the test cases described in Section 2. Of the methods based on an exact decomposition we present only a few results of UMFPACK. (Results of SuperLU look very similar and have been omitted). The timings of UMFPACK given below are based on a single application of the LU decomposition, hence, no iterative refinement is performed.

The solvers described in the previous section are indicated in the plots in the present section as shown in Table 1. Moreover, this table shows to which problem a particular method is applied. This shows more or less the range to which a certain method can be applied. We remark that MGD9V can also be used for the first problem and MRILU for all. The numbering of the problems will be the same as that in the Section 2.

The CPU-times have been measured on an HP-755 workstation. All solvers were written in standard FORTRAN 77 and the programs have been compiled with the command `fort77 +O3 progname.f`. Both the results with and without preprocessing (e.g. the work which has to be done only once if several systems with the same coefficient matrix, but with different right-hand sides, have to be solved) are shown. It should be mentioned

that all solvers, except ILUT, UMFPACK and MGD9V, exploit the symmetry in the coefficient matrix for Problems 1-5. This exploitation halves the storage requirement and the time for the factorization, but has in general little effect on the solution phase.

1. *Uniform-2D.* In Fig. 3 the performance of all methods applied to this problem is shown and in Fig. 4 a magnification of this plot is made in order to be able to discriminate between the better methods.

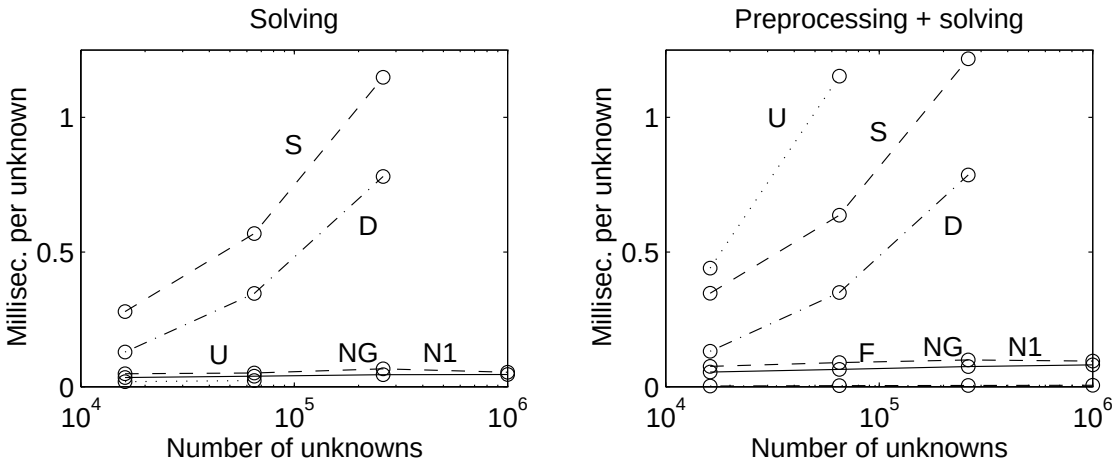

Figure 3: Timings for *Uniform-2D*; left for solving and right for preprocessing and solving

For this problem, the package FISHPAK gives the best results (see the right plot in Fig. 4). From the results of NGILU and MILU-*rrb* (marked N1 on the figure) it appears that the conjugate gradient method combined with a proper preconditioning technique performs very well. UMFPACK has a fast solution procedure. However, the factorization phase is expensive and the case M=512 requires too much memory. The poor performance of ILUT (marked by S) and ICCG with diagonal ordering (marked by D) with respect to the other methods shows the importance of (near) grid-independency for huge problems.

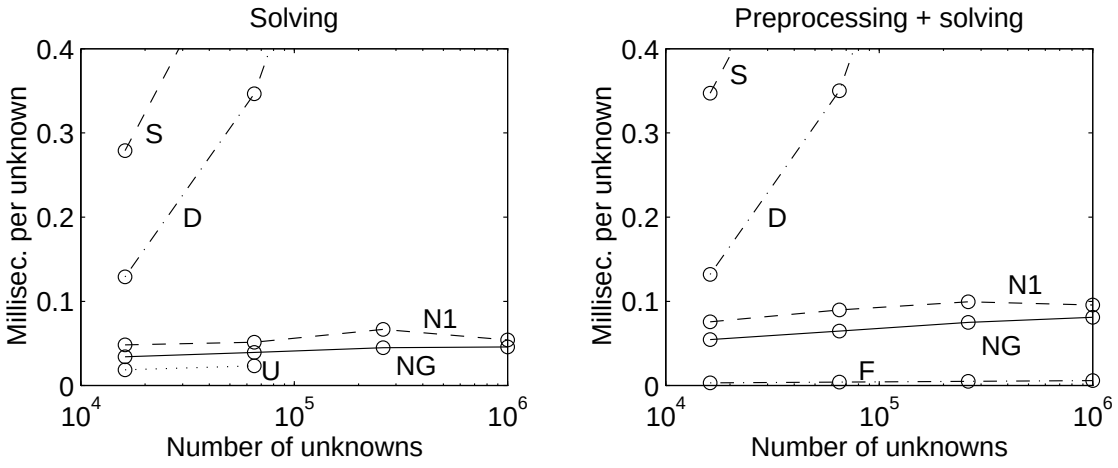

Figure 4: Timings for *Uniform-2D* for the better methods; left for solving and right for preprocessing and solving

2. *Uniform-3D.* Again it appears that FISHPAK gives the best results. It is more than a factor 10 faster than the second best method. The other methods except UMFPACK presented in Fig. 5 use the conjugate gradient method combined with a modified incomplete Choleski decomposition as preconditioning technique. From the results it appears that the difference in performance between ICCG with hyperplane ordering (marked by D) and the other incomplete decompositions is not as large as in the similar 2D-test case. This is due to a much lower condition number of the matrix in this case because there are less points in each spatial direction compared with the 2D case. Notay used DRIC (marked by N2) for this problem since MILU-*rrb* is not applicable in 3D. Hence the curve for the solver of Notay is less horizontal than in 2D. Also the curve associated with NGILU is less horizontal than in the 2D case. Another indication showing that 3D problems are intrinsically harder than 2D problems is the performance of the direct method UMFPACK. For $M = 24$ ($N = 12167$) UMFPACK requires 17 milliseconds per unknown for preprocessing and solving. Hence, for this number of unknowns it is already two orders of magnitude slower than the iterative methods. For higher values of M the memory requirements become too high. This problem shows that direct methods become unpractical in 3D for modest grid resolutions.

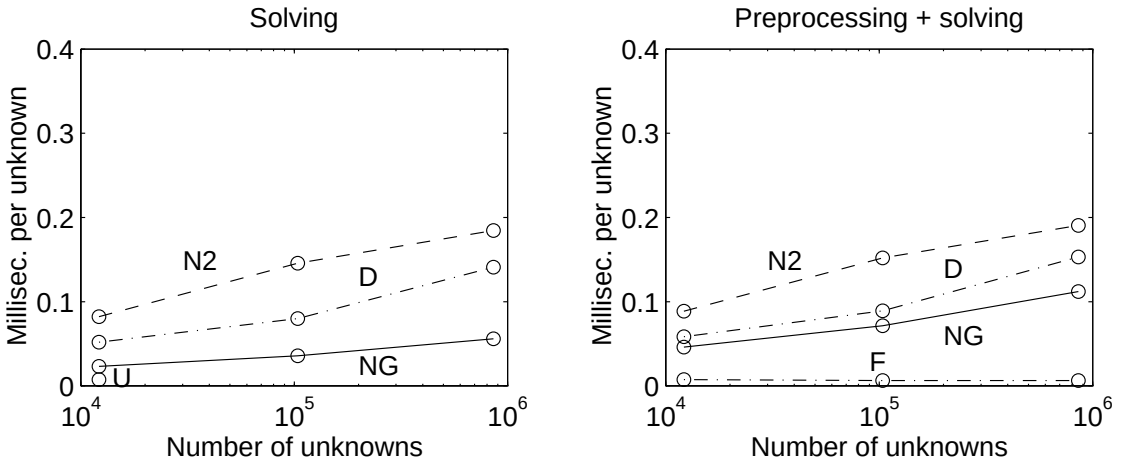

Figure 5: Timings for *Uniform-3D*; left for solving and right for preprocessing and solving

The results of ILUT are not shown in the figures, because for this problem ILUT showed a very slow convergence.

3. *Stretched.* This problem shows how important grid-independency is on highly stretched grids. The matrix has a very large condition number in this case. ICCG with diagonal ordering and ILUT severely suffer from this extreme condition number (see Fig. 6).

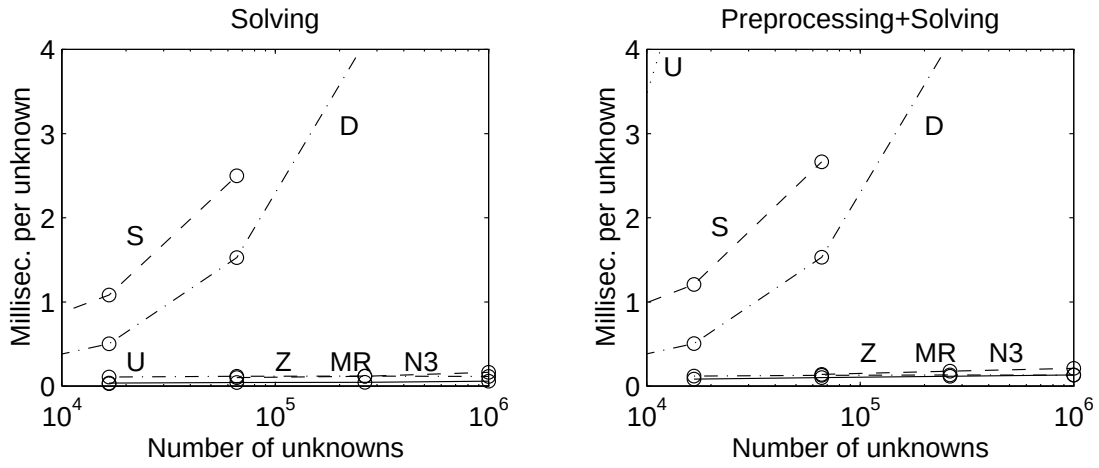

Figure 6: Timings for *Stretched*; left for solving and right for preprocessing and solving

The results of the better methods are presented in Fig. 7. With MILU-*rrb*&*rcm* (marked by N3) and MRILU the CPU-time per unknown increases only slightly with mesh refinement. The multi-grid method MGD9V (marked by Z) is truly grid-independent: the CPU-time per unknown does not increase at all when the grid is refined.

For UMFPACK the story is as before. However, there is a difference with the *Uniform-2D* problem. Though with the *Uniform-2D* problem the case $M = 256$ could be solved by UMFPACK, here it runs out of memory for this value of M. We attribute this to an unlucky pivoting step.

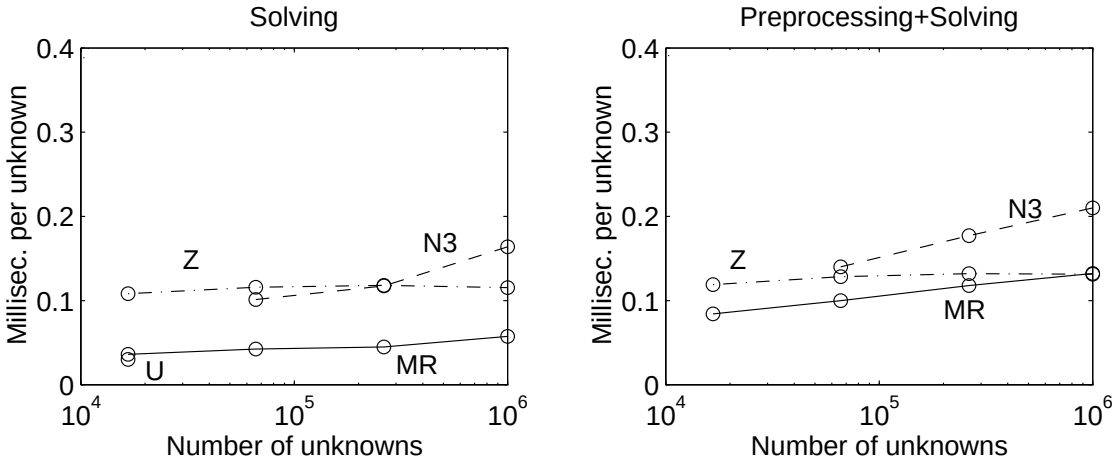

Figure 7: Results of the better methods for *Stretched*; left for solving and right for preprocessing and solving

4. *Discontinuous.* The CPU-times in milliseconds per unknown for both with and without preprocessing are listed in Table 2. The problem run here contains N=160801 unknowns and can be compared with results of the previous 2D problems for similar number of unknowns. Compared to Fig. 4 we observe that NGILU performs about the same, whereas MILU-*rrb* is a little slower. MGD9V is more than a fac-

tor 2.5 slower than its corresponding performance on the stretched grid shown in Fig. 7. ILUT has a slow convergence on this problem and is even outperformed by UMFPACK.

Table 2: Results for *Discontinuous* (N=160801)

	Solving	Preprocessing+Solving
ICCG with diag. ord.	1.20	1.21
MGD9V	0.29	0.30
MILU-*rrb*	0.09	0.15
NGILU	0.05	0.09
ILUT	2.75	2.89
UMFPACK	0.03	1.7

5. *Unstructured.* Again the CPU-times per unknown in milliseconds are listed (see Table 3). For this problem only those methods remain that can handle arbitrary sparsity patterns. The problem size is modest, which explains that here UMFPACK performs well. Here we have a larger gap between MRILU and the solver of Notay because, contrary to MILU-*rrb* (which is not applicable to unstructured problems), DRIC does not exploit any multilevel structure. MRILU needs more than 70 percent of the CPU-time for the construction of the preconditioner. This is substantially more than in the *Stretched* problem where this number is about 50 percent. It appears that MRILU has less fill-in for stretched problems than for "equidistant" problems.

Table 3: Results for *Unstructured* (N=25759)

	Solving	Preprocessing+Solving
DRIC	0.29	0.74
MRILU	0.08	0.21
ILUT	0.97	1.09
UMFPACK	0.03	1.5

6. *Non-symmetric.* For this modest sized problem UMFPACK has the shortest solve time (see Fig. 8). In the factorization phase it is again expensive, but if one has to compute the solution for a large number of right-hand sides it may be worth the effort. Furthermore, RILU_fill and MRILU perform well. The difference between the methods will become more pronounced for higher resolutions, where the grid-independency pays off.

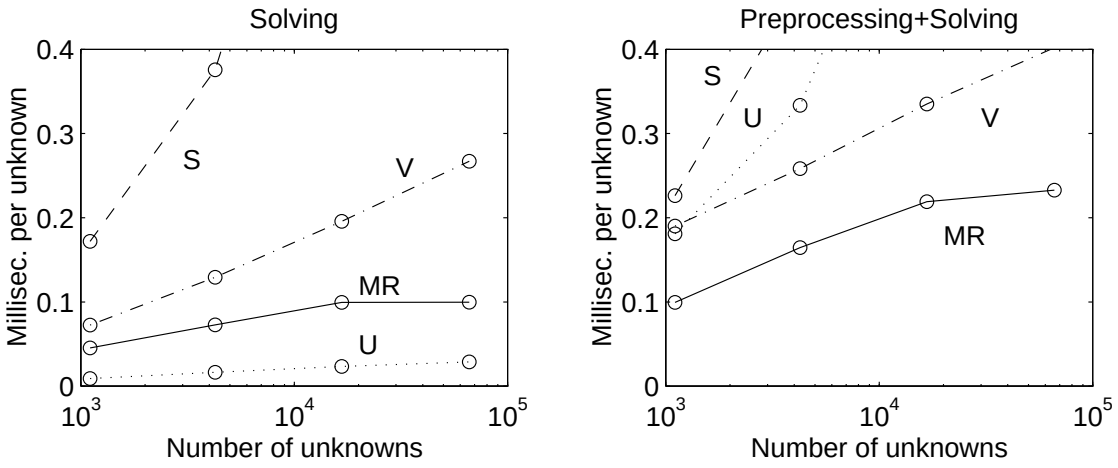

Figure 8: Results for non-symmetric; left for solving and right for preprocessing and solving

5 Conclusions

For Problem 1, FISHPAK gives the best results. This package optimally exploits the symmetry in the problem and solves the systems of linear equations more than ten times faster than the other solvers. A drawback of FISHPAK is that it can only be used for matrices that can be represented by stencils with constant coefficients, whereas all other solvers are developed for a much broader class of applications. In some cases it is possible to use it as a preconditioner, but then the iterative method should converge within about 10 iterations to remain competitive with the other methods. Moreover, in 3D the FISHPAK solver is based on Fast Fourier Transforms which makes it very sensitive to the number of grid points, e.g., changing M from 96 to 97 (a prime) yields an increase of the computation time with a factor 6.

For the uniform-3D problem it is found that direct solvers based on LU factorization become unpractical due to the high level of fill-in needed. ICCG with hyperplane ordering performs much better here than its counterpart in the 2D case. This is due to the better conditioning of the 3D matrix as compared to the one of the 2D case when using the same number of unknowns.

The packages UMFPACK and SuperLU, which are based on a complete decomposition of the coefficient matrix, can be interesting for relatively small systems of equations, especially when the complete decomposition can be used more than once. The decomposition phase is typically two to three orders of magnitude more expensive than the solution phase. It appears that the solution phase of these methods is extremely fast due to a proper cache handling. However, for large systems of equations, both the storage requirements and the CPU-times of these methods are orders of magnitude higher than for iterative methods.

For problems in which an extreme stretching of the grid is used MRILU, MILU-*rrb*&*rcm* and the multi grid package MGD9V, are the only methods that perform well. MGD9V has the nice property that the CPU-time per unknown does not increase at all with mesh refinement. With MRILU and MILU-*rrb* the CPU-time increases only slightly when the mesh is refined. For systems of linear equations with not more than 10^6 unknowns MRILU is the most efficient method.

The only methods able to solve all six test problems are UMFPACK, ILUT and a multilevel-ILU preconditioning technique (NGILU in case of regular, uniform grids, and MRILU otherwise). The last method has the additional advantage to perform well for all test problems. Only for a uniform grid, NGILU is outperformed by the package FISHPAK.

We have focussed mainly on the number of operations necessary to solve $Au = b$, and the efficiency on an HP-755 workstation. On supercomputers the performance will of course be different (this might be an interesting subject for a next workshop).

References

[1] O. Axelsson and V. Eijkhout. The nested recursive two level factorization for nine-point difference matrices. *SIAM J. Sci. Stat. Comput.*, 12:1373–1400, 1991.

[2] O. Axelsson and G. Lindskog. On the eigenvalue distribution of a class of preconditioning methods. *Num. Math.*, 48:479–498, 1986.

[3] E.F.F. Botta and A. van der Ploeg. Preconditioning techniques for matrices with arbitrary sparsity patterns. In *Proceedings of the IX International Conference on Finite Elements in Fluids*, pages 989–998, October 1995. Venice.

[4] E.F.F. Botta and F.W. Wubs. The convergence behaviour of iterative methods on severely stretched grids. *Int. J. Num. Methods Eng.*, 36:3333–3350, 1993.

[5] Cl.W. Brand. An incomplete-factorization preconditioning using red-black ordering. *Numer. Math.*, 61:433–454, 1992.

[6] B. Buzbee, G. Golub, and C. Nielson. On direct methods for solving Poisson's equation. *SIAM J. Numer. Anal.*, 7:627–656, 1970.

[7] T.A. Davis and I.S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. Technical Report TR-95-020, University of Florida, 1995.

[8] K. Dekker. Preconditioned conjugate gradient techniques for the solution of time-dependent differential equations, Faculty of Technical Mathematics and Informatics Report 92-12, Delft University of Technology, 1992.

[9] J.W. Demmel, S.C. Eisenstat, J.R. Gilbert, X.S. Li, and J.W.H. Liu. A supernodal approach to sparse partial pivoting. Technical report, University of California, Berkely, 1995.

[10] S.C. Eisenstat. Efficient implementation of a class of preconditioned conjugate gradient methods. *SIAM J. Sci. Stat. Comput.* 2:1–4, 1981.

[11] I. Gustafsson. A class of first order factorization methods. *BIT*, 18:142–156, 1978.

[12] J.A. Meijerink and H.A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comp.*, 31:148–162, 1977.

[13] Y. Notay. *Ordering methods for approximate factorization preconditioning*, Tech. Rep. IT/IF/14-11, Université Libre de Bruxelles, 1993.

[14] Y. Notay. DRIC : a dynamic version of the RIC method. *Journ. Num. Lin. Alg. with Appl.*, 1:511–532, 1994.

[15] Y. Notay. *An efficient algebraic multilevel preconditioner robust with respect to anisotropies.* in Algebraic Multilevel Iteration Methods with Applications, O. Axelsson and B. Polman, eds., Department of Mathematics, University of Nijmegen, The Netherlands, (1996), pp. 211–228.

[16] Y. Notay, V. Gheur, Z.O. Amar, S. Petrova, and P. Saint-Georges. *ITSOL: an evolving routine for the iterative solution of symmetric positive definite systems.* Tech. Rep. GANMN 95-02, Université Libre de Bruxelles, Brussels, Belgium, 1995 (Revised 1996). *http://homepages.ulb.ac.be/~ynotay* or *ftp://mnserver.ulb.ac.be/pub/itsol/.*

[17] Y. Notay and Z. Ould Amar. *A nearly optimal preconditioning based on recursive red-black orderings.* Journ. Num. Lin. Alg. with Appl (to appear).

[18] Y. Saad. ILUM: A Multi-Elimination ILU Preconditioner for General Sparse Matrices. SIAM J. Sci. Comput., 17:830–847, 1996.

[19] Y. Saad. SPARSKIT, a basic tool kit for sparse matrix computations. Technical report, University of Minnesota, 1994.

[20] P. Saint-Georges. G. Warzee, R. Beauwens, and Y. Notay. High performance PCG solvers for FEM structural analyses, *Int. J. Numer. Meth. Eng.*, 39:1313–1340, 1996.

[21] R.A. Sweet. A cyclic reduction algorithm for solving block tridiagonal systems of arbitrary dimension. *SIAM J. Numer. Anal.*, 14(4):706–720, 1977.

[22] A. van der Ploeg. *Preconditioning for sparse matrices with applications.* PhD thesis, University of Groningen, 1994.

[23] H.A. van der Vorst. ICCG and Related Methods for 3D Problems on Vector Computers. *Comput. Phys. Comm.* 53:223–235, 1989.

[24] H.A. van der Vorst. High performance preconditioning. *SIAM J. Sci. Stat. Comput.*, 10:1174–1185, 1989.

[25] H.A. van der Vorst. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for solution of non-symmetric linear systems. *SIAM J. Sci. Stat. Comp.*, 13:631–644, 1992.

[26] C. Vuik. Fast iterative solvers for the discretized incompressible Navier-Stokes equations. *Int. J. for Num. Meth. Fluids*, 22:195–210, 1996.

[27] C. Vuik. Iterative solution methods for flow problems. In S.D. Margenov and P.S. Vassilevski, editors, *Iterative methods in linear algebra II, Proceedings of the Second IMACS International Symposium on Iterative Methods in Linear Algebra, Blagoevgrad, Bulgaria, June 17-20, 1995*, IMACS series in computational and applied mathematics 3, pages 453–461, Piscataway, 1996. IMACS.

[28] P.M. de Zeeuw, Chapter 14: multigrid and advection, in: C.B. Vreugdenhil and B. Koren, eds., *Numerical Methods for Advection-Diffusion Problems*, Notes on Numerical Fluid Mechanics 45 (Vieweg Verlag, Braunschweig, 1993) 335–351.

[29] P.M. de Zeeuw, Matrix-dependent prolongations and restrictions in a blackbox multigrid solver, *J. Comput. Appl. Math.* 33:1–27, 1990.