Comparison of algebraic solution methods on a set of benchmark problems in linear elasticity

Part I:

Direct solution methods and incomplete factorization preconditioned conjugate gradient methods

Owe Axelsson[1], Igor Kaporin[2], Andrey Kucherov[3], Maya Neytcheva[4]

Abstract

A set of benchmark problems in linear elasticity which are typically very ill-conditioned is defined. Various direct and iterative solution methods are presented and tested on this set. In the present Part I of the study, direct methods and incomplete factorization preconditioned method are tested and compared.

Part II:

Iterative solution methods of Schur complement and related types and parallel implementations

Owe Axelsson[1], Igor Kaporin[2], Igor Konshin[2], Andrey Kucherov[3], Maya Neytcheva[4], Ben Polman[4], and Alexey Yeremin[5]

Abstract

In Part II of the study, Schur complement based and related iterative solution schemes are tested on the same set of benchmark problems.

[1]Department of Mathematics, University of Nijmegen, The Netherlands, e-mail: axelsson@sci.kun.nl

[2]Center for Supercomputer and Massively Parallel Applications,Computing Center of Russian Academy of Sciences,Vavilova 40, Moscow 117967, Russia, e-mail: kaporin@ccas.ru, horse@ccas.ru

[3]Moscow State University, Department of Computational Mathematics and Cybernetics, 119899 Moscow, Russia, e-mail: ab_k@mail.ru

[4]Department of Mathematics, University of Nijmegen, The Netherlands, e-mail: neytchev@sci.kun.nl, polman@sci.kun.nl

[5]Computing Center of the Moscow State University, Vorobievy Gory, Moscow 119899, Russia, e-mail: badger@vvv.srcc.msu.su

Contents

Part I

Direct solution methods and incomplete factorization preconditioned conjugate gradient methods

1 Introduction

Ill-conditioned systems arise in various applications in science and engineering. In this paper we are especially interested in systems from structural engineering. Such problems are, in general, characterized by large aspect ratios (with respect to geometry and anisotropic materials), jump in coefficients (the structure being composed of different materials) and sometimes nearly incompressible materials. This causes very small eigenvalues in the matrix of the resulting discrete system of equations to be solved. The systems can also be nearly singular due to a small portion of Dirichlet boundary conditions, which nearly permits a rotational degree of freedom.

All these properties and, in particular, combinations of two or more of them causes very ill-conditioned algebraic systems. In addition, especially for problems in three space dimensions, the size of the systems can be very large, in the order of millions of variables. Finding efficient solution methods to get a reasonable solution time for such problem remains still a challenge both for sequential and parallel computations.

The matrices arising in structural engineering are normally symmetric and positive definite matrices and we deal only with such systems here.

There exist direct solution methods based on factorization in triangular factors which are very efficient for small and medium sized problems. Examples of such methods are reordering methods, such as the minimal degree ordering ([2], [14]), and the nested dissection ordering method ([12], [13]). The latter is a recursive method, repeatedly partitioning the system in 2×2 block form by reordering, in order to enable an efficient reduction of the system to a smaller size via its Schur complement. For an $n \times n$ ($d = 2$) or $n \times n \times n$ ($d = 3$) mesh the computational complexity for the factorization grows typically as $O(n^{3d-2})$, $n \to \infty$. This growth is due to the amount of fill-in which arises in factorization methods.

Iterative solution methods have in general a smaller asymptotic growth of computational complexity. However, for ill-conditioned systems they may require an excessive amount of iterations. Although each iteration step may be cheap the total cost may therefore be larger and the break-even point between iterative and direct solvers may take place only for very large systems. This holds in particular for very ill-conditioned systems, which is of our concern. Furthermore, when solving several systems with the same matrix, the initial factorization cost in a direct solver can be amortized over the systems and since the solution cost grows slower than the factorization cost (as $O(n^{2d-1})$, or even only as $O((\log n)^2)$ for the nested dissection method in $2D$), the break-even point is even further increased.

Therefore, in order for an iterative method to be competitive not only on excessively large sized problems one must use an efficient preconditioner. Such preconditioners must be close in some respect to direct solvers. A natural choice are the incomplete factorization methods based on some drop tolerance criterion, i.e., fill-in entries which arise during the factorization of the matrix are discarded when their relative size is sufficiently small. By choosing a proper drop tolerance in this way one may avoid the major part of the amount of fill-in without loosing too much in accuracy as a preconditioner. Since fill-in may cause new fill-in at later stages in the factorization procedure, the amount saved can be of orders of magnitudes smaller than for the exact factorization.

Classical incomplete factorization methods. such as ILU(0) (cf. [22]) and its relaxed version RILU (cf. [7]) have been mostly analysed for M-matrices. Since the matrices occurring in our study are symmetric positive definite but not M-matrices, we need more robust solvers. These are based on some drop tolerance and control of pivot entries.

The first version of such a method is the method by Ajiz and Jennings [1]. This is a first order method, i.e., the deleted entries give a first order perturbation to the exact factorization. A more efficient version, RIC2S, a method of second order has recently been developed by Kaporin [19]. For such preconditioning methods, the computational cost increases as $O(n^{d+1})$, when combined with a conjugate gradient iterative acceleration method. The condition number grows namely as $O(n^2)$, as it does for the unpreconditioned method, but the major constant, a in the asymptotic expansion of the cost function $an^2 + bn$ is much smaller.

It has recently been shown ([19]) that such methods have a robust behaviour and can well compete with direct solvers, However, the cost of the method, in particular for its factorization turns out to be sensitive to the choice of the drop tolerance parameter, and, it seems that one must tune this parameter for each problem to make the method efficient. Furthermore, due to the increase of the computational cost, it may still be somewhat too costly for very large sized problems. For some classes of matrices there exist modified methods by row compensation for instance, or relaxed version of those ([5]). Here the deleted fill-in entries are moved to the diagonal entry in the same row, possibly multiplied first with some relaxation parameter ω, $(0 < \omega < 1)$. When $\omega = 1 - c/n$, $c > 0$, then it can be shown for M-matrices (see [5], for instance) that the condition number reduces to $O(n)$ and the total cost grows only as $O(n^{d+\frac{1}{2}})$.

As already mentioned, matrices arising in structural mechanics however, are not M-matrices and the row-equilibrium method may lead to indefinite preconditioners when applied on them. In some cases, one can use a separate displacement ordering and base a modified preconditioner on the so obtained (d) block-diagonal matrices, see e.g. [6] for details. However, this method is not efficient when the constant in the Korns (ellipticity condition) inequality is small, which is typically the case for the previously mentioned ill-conditioned systems.

There are alternative methods which incorporate the row equilibrium via the matrix inverse. Here the preconditioner takes the form

$$B = M^{-1} + \sigma V A_V^{-1} V^T \tag{1.1}$$

where B approximates the inverse (A^{-1}) of the given matrix and M is a preconditioner to A, such as an incomplete factorization. Further, the column vectors of V (an $n \times m$ matrix, $m \ll n$) span a subspace, typically consisting of approximations of the eigenvectors to the smallest eigenvalues of $M^{-1}A$. Further $A_V = V^T AV$. Unless m is very small, it can be efficient to replace A_V with an approximation, see [26] for details. Finally, σ is a positive scaling parameter to be chosen such that the smallest eigenvalues are moved sufficiently upwards while the other eigenvalues are not moved too much to avoid an significant increase of the maximum eigenvalue.

Since B contains two terms where the first is positive definite and the second is positive semidefinite, B itself is positive definite. Hence there can be no loss of positive definiteness.

The preconditioner (1.1) is one of the methods one can use to improve the quality of the preconditioner M.

Another and in fact related, method is based on partitioning the matrix A in two-by-two block form and solving the reduced, Schur complement system. The coupling of domain decomposition and Schur complement approaches have been used in combination with direct solution methods to enable solution even of quite large problems. Hereby, each subdomain corresponds to a superelement in the underlying structure, such as a wing in an airplane. The Schur complement matrix was formed explicitly and then solved by a direct method. The computation of the Schur complement, however, can be costly. In an iterative solution method, there is no need to compute the Schur complement matrix explicitly, as it is only the actions of it in matrix vector multiplications which are needed. Thereby the inner systems corresponding to the eliminated degrees of freedom can possibly be solved by an 'inner' iteration method also. Furthermore, the computational complexity of the latter increases much slower than the former with increasing problem sizes.

The advantage using Schur complement systems is that it has smaller size, in some applications much smaller size, and is better conditioned than the original system. Furthermore its preconditioner can be computed with much less expense. The disadvantage is that the inner systems can still be quite expensive to solve. Both the outer system, i.e. the Schur complement system and the inner system need efficient preconditioners. As it turns out a quite efficient preconditioner to the Schur complement matrix is a coarse mesh matrix when the coarse mesh is not too much apart from the fine mesh. However, in general it must be further improved to lower the number of outer iterations, because they multiply the number of inner iterations required during each outer iterations. The modification of the preconditioner to improve its accuracy and asymptotic condition number can here be done via the relation

$$S_2^{-1} = (A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1} = A_{22}^{-1} + A_{22}^{-1}A_{21}S_1^{-1}A_{12}A_{22}^{-1} \tag{1.2}$$

where $S_1 = A_1 - A_{12}A_{22}^{-1}A_{12}$ (which can be derived using the Sherman-Morrison formula). Note that the second term in (1.2) has a form similar to that in (1.1).

In this paper a domain decomposition like ordering has been chosen to partition the matrix for the Schur complement method. Together with the other solution methods it will be presented with its computational complexity and applied on a set of very extensive benchmark problems. It will also be combined and compared with the other tested methods.

The remainder of the paper is organized as follows.

In Section 2 the set of benchmark problems to be used for the tests are described. In the following sections the various methods and the tests done with them are described. Section 3 deals with direct solution methods and the results here form a platform on which to build the further comparisons with the iterative solvers. Section 4 describes the RIC2S factorization method. The tests of this method shows how the method compares with the direct solution method and with the (block) Jacobi preconditioned conjugate gradient method. It also shows the sensitivity of the performance of the method to the threshold parameters occurring in the method.

Part II of this paper begins with an introduction to Schur complement based and related preconditioners. Section 7 describes then the augmented subspace approximate inverse preconditioner and Section 8 the Schur complement based solvers. The latter have been combined with the RIC2S solvers. Section 9 shows how the ASP method performs on problems with several right hand sides. Section 10 describes a parallel implementation of a somewhat adjusted form of the RIC2S method. Section 11 illustrates the parallel performance of a nonoverlapping block-Jacobi preconditioned CG utilizing the block structure of the stiffness matrix introduced by some special domain decomposition orderings. Section 12 discusses some other approaches, such as the algebraic multilevel iteration method which, however, require a multilevel structure not available in the benchmark problems and combined explicit-implicit block matrix factorization method and its numerical performance. Finally, Section 13 discusses the tests done and the comparisons of the results.

2 Set of benchmark problems

The various solution techniques studied in this paper are tested and compared on a set of benchmark problems, described briefly below. Problems 1, 2, 5 and 6 are real-life problems, provided by The Netherlands Organization for Applied Scientific Research (TNO) and The Netherlands Rijkswaterstaat Institute. Problems 3 and 4 emulate real-life problems from soil mechanics with simplified geometry.

2.1 2D problems

PROBLEM 1 *[Dam] This is a cross-section of a concrete dam built on a rock massif. The linear elasticity problem is solved for the dam and a portion of the underlying rock foundation. The geometry of the model and the coarse mesh used are shown in Figure 1. The material properties of the concrete body are* $E = 30.0\,10^9\ N/m^2$, $\nu = 0.2$, *and for the rock bed -* $E = 6.0\,10^9\ N/m^2$, $\nu = 0.25$. *To compute the surface forces due to pressure of the water (on* $\Gamma\,4 - \Gamma\,6$*), the formula* $p = \rho\, g\, h\ \ N/m^2$ *is used, where* $\rho = 10^3\ \ kg/m^3$ *is the specific weight of the water,* $g = 9.80665\ N/kg$ *is the gravity constant and* h *is the height of the water at the corresponding part of the boundary.*

Homogeneous Dirichlet boundary conditions are assumed on $\Gamma\,1$ *and homogeneous Neumann boundary conditions – on* $\Gamma\,2$,$\Gamma\,3$, $\Gamma\,7 - \Gamma\,11$.

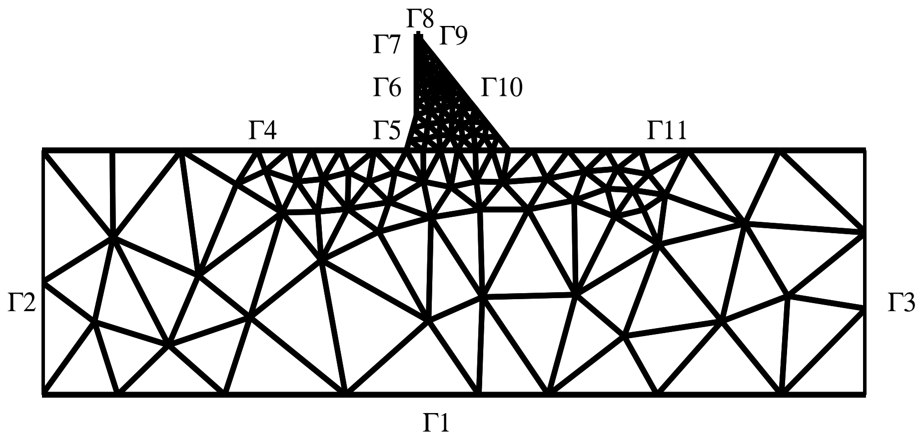
The coarse mesh consists of 204 elements and 129 nodes. The problem has been tested for meshes obtained after 0 to 4 refinement steps and the corresponding degrees of freedom are as follows:

Level of refinement	*0*	*1*	*2*	*3*	*4*
Degrees of freedom	*3434*	*13474*	*53958*	*210562*	*838914*

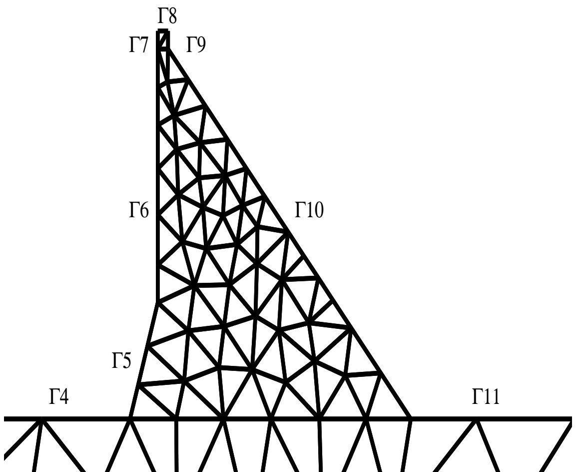
PROBLEM 2 *[Bridge] This problem is a cross-section of a bridge construction which is 48.70 m broad and 7.89 m high. It is of homogeneous material with* $\nu = 0.2$ *and* $E = 3\,10^7$. *There are no surface forces applied, the body is loaded by its own weight. The initial discretization is shown in Figure 2(a) and a detail of the same is illustrated in Figure 2(b).*

The coarse mesh consists of 1412 elements and 861 nodes.

The problem has been tested for meshes obtained after 0 to 5 refinement steps and the corresponding degrees of freedom are as follows:

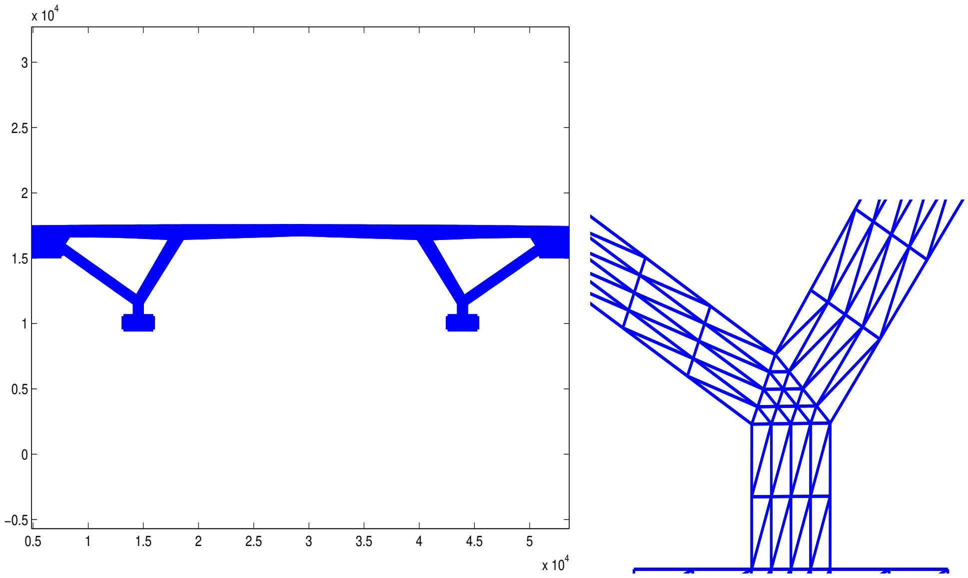


(a)



(b) A zoom of the mesh on the dam itself

Figure 1: Problem 1: 'Dam'



(a) The geometry of the domain (b) A detail of the initial mesh

Figure 2: Problem 2: 'Bridge'

Level of refinement	0	1	2	3	4	5
Degrees of freedom	1722	6270	23838	92862	366462	1455870

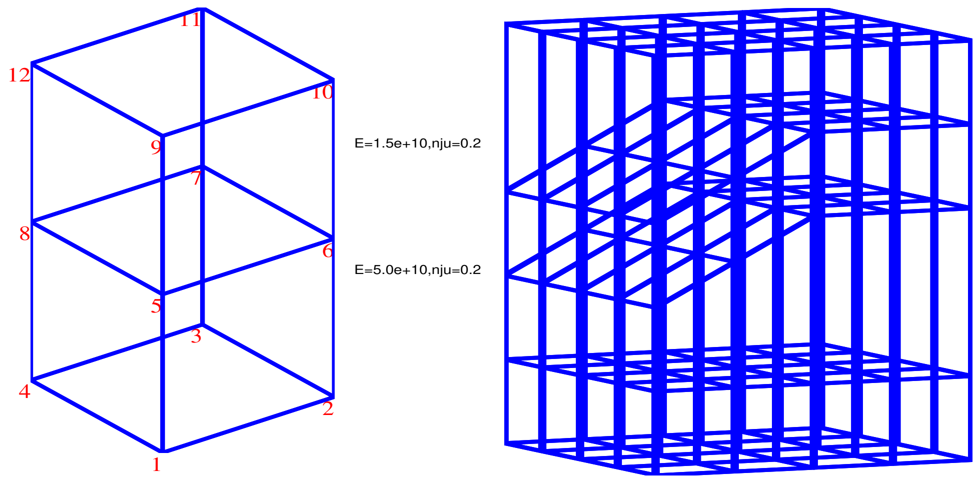
2.2 3D problems

PROBLEM 3 *[Bricks] This example is with two materials occupying a region* $\Omega = \{-1 \le x \le 1, -1 \le y \le 1 - 1 \le y \le 3\}$, *with characteristics* $\nu = 0.2$, $E = 1.5\,10^{10}$ *and* $\nu = 0.2$, $E = 5\,10^{10}$, *as illustrated in Figure 3(a). Homogeneous Dirichlet boundary conditions are imposed at the bottom and homogeneous Neumann conditions elsewhere. The inhomogeneous body is subjected to a force in* x*-direction of size* $-5e+7$.

The coarse mesh consists of 2 elements and 12 nodes. The problem has been tested for meshes obtained after 0 to 4 refinement steps and the corresponding degrees of freedom are as follows:

Level of refinement	0	1	2	3	4
Degrees of freedom	135	675	4131	28611	212355

PROBLEM 4 *[Soil] This example models a soil mechanics problem and involves four different materials, occupying a domain of* $1000 \times 1000 \times 2500$ cm *with four horizontal layers with different material parameters (top-to-bottom)* $\nu = 0.4, 0.35, 0.25, 0.2$ *and* $E = 0.54$, $E = 0.9$, $E = 1.4$ *and* $E = 2.1$, *respectively, see Figure 3(b) The body is subjected to a vertical load of* $-2.8\,10^{-3}$, *applied to the top side. The boundary conditions are homogeneous Dirichlet on* $\{0 \le x, y \le 1000, z = 0\}$ *and homogeneous Neumann elsewhere.*



(a) Initial mesh for Problem 3 (b) Initial mesh for Problem 4

Figure 3: Soil mechanics prototype test problems: 'Bricks' and 'Soil'

The coarse mesh consists of 64 3D brick elements and 125 nodes. The problem has been tested for meshes obtained after 0 to 4 refinement steps and the corresponding degrees of freedom are as follows:

Level of refinement	*0*	*1*	*2*	*3*	*4*
Degrees of freedom	*375*	*2187*	*14739*	*107811*	*823875*

PROBLEM 5 *[Detail] This problem is a part of the full 3D model of the bridge construction, illustrated in Figure 3(b). The T-shaped domain is of homogeneous material with* $\nu = 0.29$ *and* $E = 2.068e + 11$.

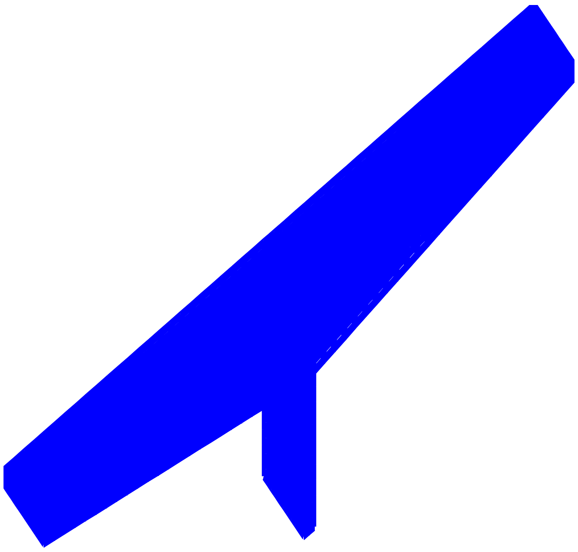
A part of the structure of the stiffness matrix corresponding to the coarse mesh is depicted in Figure 4(b).

The coarse mesh consists of 2800 3D brick elements and 3828 nodes. The problem has been tested for meshes obtained after 0 to 2 refinement steps and the corresponding degrees of freedom are as follows:

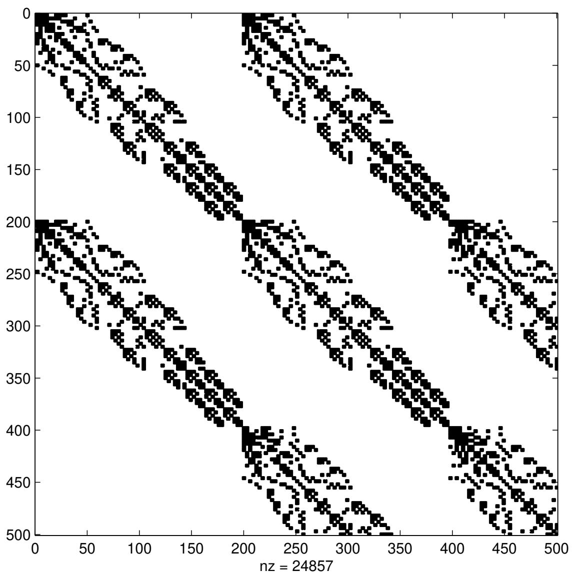
Level of refinement	*0*	*1*	*2*
Degrees of freedom	*11484*	*79065*	*584127*

PROBLEM 6 *[Sluice] This problem is a part of a sluice gate and its geometry is seen in Figure 5(a). There are boundary conditions of Dirichlet type imposed at the 'bottom' nodes shown in red in Figure 5(b). The body is homogeneous with* $\nu = 0.2$ *and* $E = 3.2e + 07$.

The coarse mesh consists of 372 3D brick elements and 724 nodes. The problem has been tested for meshes obtained after 0 to 3 refinement steps and the corresponding degrees of freedom are as follows:

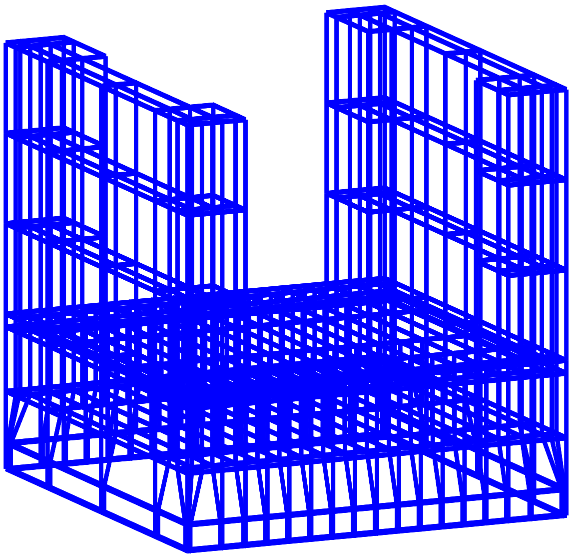


(a) The shape of the domain

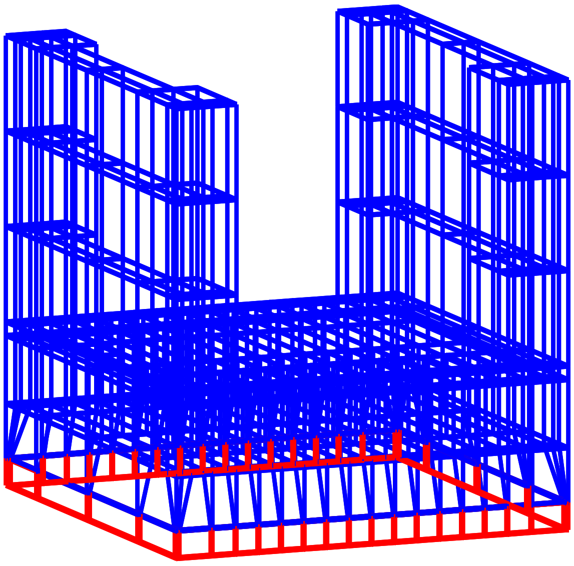


(b) A part of the coarse-mesh stiffness matrix

Figure 4: Problem 5: ’Detail’



(a) initial mesh



(b) Dirichlet b.c. points (in red)

Figure 5: Problem 6: 'Sluice'

Level of refinement	*0*	*1*	*2*	*3*
Degrees of freedom	*2172*	*12957*	*87159*	*633579*

2.3 The computer facility

Unless stated explicitly, all numerical examples have been performed on a shared memory SMP SUN Enterprise 3000 computer with eight UltraSparc 170 MHz CPUs and with 1GB main memory.

3 Direct solution methods

In this section, large scale s.p.d. systems arising in practical 2D and 3D linear elasticity problems are solved by sparse direct methods. We focus on testing of three reordering schemes which are the most popular in a practice, namely the Reverse Cuthill-McKee, the Nested Dissection and the Minimum Degree algorithms. We present data on the dependence of the performance of the solver on the size of discrete problems tested as well as on the ordering scheme used. The sparse direct methods are compared with respect to execution time, number of arithmetic operations and amount of the core memory required. All computations are conducted on a Sun Enterprise 3000 workstation and for this machine we also display and discuss the observed Mflops rate, a characteristic which is strongly dependent on computer hardware and algorithm structure utilized.

3.1 Introduction

We consider the problem $Ax = b$ where A is a sparse $n \times n$ s.p.d. matrix, and present numerical results obtained with the use of three popular ordering algorithms which produce a proper permutation matrix P and of corresponding direct solvers, which solve the reordered system numerically

$$PAP^T y = Pb, \quad x = P^T y.$$

As is well known the only purpose of reordering of s.p.d. matrices is to reduce the fill-in when the Cholesky decomposition is applied to the permuted matrix $PAP^T = LL^T$. This decomposition is stable for any symmetric permutation of rows and columns. That's why unlike the case of nonsymmetric or indefinite matrices there is no need in choosing pivot with respect to numerical stability and we can focus on minimizing fill-in of the triangular matrix L. The discussed s.p.d. properties make possible to divide the whole solution process into three relatively independent steps:

1) *Ordering*, which results in finding a "good" permutation matrix P.

2) *Symbolic and numerical factorization*.

The goal of the symbolic factorization step is to determine the position of the nonzero entries in the factor matrix L, to determine the amount of the memory required and the storage scheme by which the factor matrix L and right-hand-side are represented.

Besides, the actual values of right-hand-side b and matrix A should be properly permuted and incorporated into the storage scheme. Numerical factorization which computes the entries of the Cholesky factor L is normally the most time consuming part of the solution process.

Often step 2 is divided into three independent parts - symbolic factorization, allocation and numerical factorization. This approach underlines the fact that the steps preceding the numerical factorization could be performed only once when we have to solve a sequence of linear systems with matrices with the same sparsity pattern. For brevity,we consider the symbolic factorization as a part of the numerical factorization step.

3) *Solution* step itself consists of solving two triangular systems and obtaining the solution vector x from the vector y by inverse permutation:

$$Lw = Pb, \quad L^T y = w, \quad x = P^T y.$$

The problem of finding of optimal ordering is NP-complete, so only some heuristic approaches are acceptable in practice.

In next subsection we consider three different ordering schemes and describe shortly the corresponding storage schemes. In the last subsection we display and discuss the results of numerical experiments obtained on the benchmark collection of matrices, defined in Section 2.

3.2 Description of the direct solution methods

We have tested numerically three ordering algorithms described in the book of A. George and J. Liu [13] and implemented in the Waterloo sparse matrix package SPARSPAK [13], [15].

Below we shall use some common graph notations. In particular, we say that a pair of vertices $(x_i, x_j), \quad i \neq j$, is an edge in a graph $G^A = (X, E)$ of a symmetric matrix A if and if $a_{ij} \neq 0$, where X is a set of vertices, E - a set of edges. Two vertices x, y called adjoint, if $(x, y) \in E$.

If $Y \in X$ is a given set of vertices of the graph G^A, then an adjoint set for Y (with respect to X) is

$$Adj(Y) = \{x \in X - Y| \quad \{x, y\} \in E \quad \text{for some } y \in Y\}.$$

A degree $|Y|$ of $Y \in X$ is the number of elements in $Adj(Y)$. In particular, the degree of a vertex y is defined as a number of vertices adjoint to y.

1). **GenRCM** - Generalized Reverse Cuthill-McKee algorithm [13],[9], [20]. The purpose of this algorithm is to minimize the envelope (in other words a band of variable width) of the permuted matrix. In graph notations this algorithm reads as follows

Algorithm GenRCM.

1. *Initialization*. Choose a starting (root) vertex r and set $x_1 = r$.
2. *Main loop*. For $i = 1, ..., n$ find all non-numbered neighbours of x_i and number them in the increasing order of their degrees.
3. *Reverse order*. The reverse Cuthill-McKee ordering is $y_1, ..., y_n$, where $y_i = x_{n+1-i}$.

One can see that GenRCM aspires to number the vertices adjoint to the already ordered first.

The choice of a root vertex is of a special interest. One inexpensive and acceptable solution consists of taking a pseudo-peripheral vertex as a root. The corresponding algorithm from SPARSPAK is based on the rooted level structure (RLS).

Let x is a given vertex (i.e. a root). The RLS with root x is a partitioning $L(x)$ of the set X defined as (c.f. [13]) $L(x) = \{L_0(x), L_1(x), ..., L_{l(x)}(x)\}$, where

$$L_0(x) = \{x\}, \quad L_1(x) = Adj(L_0(x)), L_i(x) = Adj(L_{i-1}(x)) - L_{i-2}(x), \qquad i = 2, 3, ..., l(x).$$

We note that vertices of each i-th level could be connected to themselves and to vertices of the preceding and next levels. Roughly speaking we are looking for such a root r for which its RLS length $l(r)$ is maximal.

The corresponding algorithm from SPARSPAK reads as follows.

1. *Initialization*. Choose an arbitrary node r in X.
2. Construct RLS with a root r and determine its length $l(r)$.
3. Choose in $L_{l(r)}(r)$ a vertex x of minimal degree.
4. a) Construct a RLS rooted at x and determine its length $l(x)$.
   b) If $l(x) > l(r)$ set $r = x$ and goto step 3.
5. Otherwise, x is a pseudo peripheral vertex.

As was shown in [20] the inversion on step 3 could never decrease but often improve the quality of the ordering.

The complexity of the algorithm is bounded from above by $O(m\ nzA)$, where m is a maximum degree of vertices, nzA - number of nonzero entries of matrix A.

This algorithm is usually accomplished by a relatively simple envelope storage scheme in which all entries of the envelope of the permuted matrix are stored by rows. These row envelope entries are stored consequently from the first left to the last diagonal one. (In SPARSPAK implementation diagonal entries are stored alone).

In practice this algorithm could construct the RCM ordering very quickly.

Unfortunately the fill-in of each envelope row occurs also very quickly. That is why usually almost all envelope entries should be computed and saved. So the number of arithmetic operations and the amount of the memory required for this algorithm is usually unacceptable for large scale problems. However the results of our numerical comparisons showed that it remains still interesting for many actual applications (see next subsection).

Unlike the envelope method the other storage scheme keeps only nonzero entries of the Cholesky factor. Fortunately it is not necessary to store column indexes for each nonzero entry of a matrix L. The ingenious observation made by A. Sherman [29] says that the tails of many columns modified in a course of decomposition possess the same sparsity structure. Therefore column indices of many entries could be replaced by reference to the last entries of proper generative column. This compressed scheme is very efficient for large scale problems.

The compressed scheme is utilized by the two other ordering methods tested.

2) **GenND - The Nested Dissection algorithm** uses the global information about the matrix. Each recursive step of the algorithm results in finding a separator of a each connected graph component. A separator is a subset of vertices whose removal subdivides the graph into two or more components. Several strategies how to determine a separator in a graph are known (see, for example, [13]). To subdivide a graph SPARSPAK constructs the RLS structure for each connected component and takes the middle RLS level as a separator. Numbering the vertices of the separator last results in the following structure of the permuted matrix with prescribed zero blocks in positions $(2,1)$ and $(1,2)$

$$\begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}.$$

Under the assumption that subdivided components are of equal size the algorithm requires no more than $\log_2 n$ steps to terminate. It is well known that the ND algorithm is optimal (within a constant factor) for the class of model 9-point two-dimensional grid problems posed on regular $m \times m$-meshes. In this case a direct solver based on the ND ordering requires $O(m^3)$ arithmetic operations for matrix factorization and $O(m^2 log_2 m)$ arithmetic operations to solve triangular systems. Accordingly, the Cholesky factor contains $O(m^2 log_2 m)$ nonzero entries. These are the *best low order bounds* derived for direct elimination methods. In the three dimensional case and model 27-point grid problems on cubic $m \times m \times m$ meshes the number of factorization operations is estimated as $O(m^6)$. Therefore one can expect iterative methods to be superior in case of solving almost all 3D grid problems and 2D problems except small sized problems. This holds in particular for reasonably well-conditioned problems and not too irregular grids.

3) **GenQMD - The Quotient Minimum Degree** is the third ordering algorithm tested. This method was also taken and tested in SPARSPAK implementation [13].

This algorithm aims to minimize a local fill-in taking a vertex of minimum degree at each elimination step. The straightforward implementation of the algorithm is time consuming since the degree of numerous vertices adjoint to the eliminated one must be recomputed at each step. Many important modifications have been made in order to improve the performance of the MD algorithm and this research remains still active (see, for example [14] for a review up to 1988 and some recent papers [2], [28]).

In many references the MD algorithm is recommended as a general purpose fill-reducing reordering scheme. Its wide acceptance is largely due to its effectiveness in reducing fill and its efficient implementation [3]. Based on our results of numerical experiments we could not share such an optimistic evaluation of the MD algorithm. It should be noted that usually authors limit the numerical comparisons by considering model test problems (like 2D 9-point or 3D 27-point finite-element problems on regular meshes) or some real-life tests from well-known Harwell-Boeing test matrix collection [10] but in the latter case reported results concern matrices of size not larger than, say, 50000.

Our benchmark collection of test matrices contains series of problems posed on enlarging 2D and 3D meshes from small to medium and very large size. For some problems meshes are quite irregular.

We found the MD method loses to ND or even GenRCM methods on all (except one) of the 23 tasks from our test matrix collection. Moreover, on many tests it showed disappointing results in the ordering time, factorization time, or both. In several cases, the time for ordering significantly exceeds the time for factorization and solution. For example, for test program Bridge, $n = 366462, nz(A) = 4993118$, the ratio is 5.27: the QMD requires 2168.6 seconds for ordering, 410.8 seconds for factorization and 6.23 - for solving the triangular systems. We note, that another version of the MD algorithm, implemented in YSMP - Yale Sparse Matrix Package [21], spent on ordering only 14.73 seconds what is much better and comparable with the ordering time 15.86 seconds which was spent by the GenND program (we use the initial ordering as it was produced by the finite element matrix generator). On this test the best ordering was produced by GenND: 2.64e7 nonzero entries in factor L, GenQMD goes second with $nz(L) = 2.97$e7, and last is SDRV from YSMP: $nz(L) = 3.22$e7. Since in almost all tests GenQMD found better ordering than SDRV/YSMP we include the former version of the MD method only, in spite of the fact that GenQMD is a time consuming algorithm.

3.3 Numerical experiments

The results are displayed in details in Tables 1-6. The following notations are used in the tables to represent the data obtained on each of the six test problems solved and three direct solution methods tested:

n	is a number of equations, the order of a matrix A;
nzA	a number of nonzero entries in the given matrix A, which include the entries on the diagonal, in lower and upper triangles of a matrix A, $\quad nzA = nz(L_A + L_A^T - D_A)$;
Ordering time	time required to construct the permutation matrix (array) P of size n and its inverse P^T;
Factorization time	time for a factorization step of the algorithm (see introduction, i.e. including time for symbolic factorization;
Total time	time required to solve an actual problem, is counted as a sum of timings mentioned above;
True residual	equals to the ratio $\frac{\|\|b-Ax\|\|}{\|\|b\|\|}$, where x is the computed solution;
$OpsFCT, OpsSLV$	number of arithmetic operations required to perform the factorization and solution steps of the algorithm.
$OpsFCT/n, OpsSLV/n$	relative number of operations;
$Mflops$	actual speed of computations measured in million of arithmetic operation per second (to compute this value we divide the total number of arithmetic operations by the time required to factorize the matrix and solve two triangular systems only);

$Memory$	the size of core memory in Mbytes, which was required to run a direct method. We include in this storage a space for saving of the original matrix given in SPARSPAK row-wise format, this storage also includes place for the Cholesky factor L with corresponding integer indices, for right hand side and some temporary integer and real vector used by the algorithm;
$nz(L)$	the number of nonzero entries in the Cholesky factor L;
$nz(L)/nz(L_A)$	the relative fill-in of (left triangle of) the matrix A in the course of the Cholesky decomposition:
$nz(L)/n$	relative density of the computed Cholesky factor.

Besides, for brevity in all tables we use the notation RCM, ND and QMD instead of GenRCM, GenND and GenQMD correspondingly.

The presented results show that all direct methods solved 23 tests successfully with small true residuals . The remaining 4 tests, which are the largest 3D tasks with really large matrices, could not be solved due to core memory limitations. For all 2D tests including the largest ones 1 GB of core memory turned out to be enough.

A comparison of direct methods in total time, required to solve the task, gives us the Table 1 of winners in total time.

The Reverse Cuthill-McKee is the winner with respect to total time for all small scale problems. For large scale problems GenND showed the best results. The QMD method is out of competition.

If we do not take into account the time spent on ordering, then the QMD becomes a compatible and it could be recommended for solving problems of small size (see the Table 1 of winners in fill-in, in which a best ordering and corresponding fill-in $nz(L)/nz(L_A)$ are displayed).

The GenND remains the best choice for large scale problems.

For convenience of the reader, a comparison in total time of direct methods and the iterative preconditioned method RIC2S, described in Section 4, is given in a Table 2.

As is seen from Table 2 the direct method GenRCM could be highly recommended for solving all problems on meshes with a sufficiently small number of nodes (the initial or the second meshes). The GenRCM showed the best MFlops rate because this method implements direct addressing to the entries of the Cholesky factor, while the other two use the compressed storage scheme in which the address are computed via column indices. How to implement the methods based on compressed storage scheme on the modern computers/workstations having a hierarchy of memory, in particular a fast cache memory, efficiently is a question of special interest (for promising results in this direction, see [24], [27]).

The direct method GenND could be recommended for problems on intermediate meshes in 3D case and specially recommended for almost all large 2D problems including the largest considered in this numerical comparison. It is viable method for solving in subdomains within an outer iterative procedure, where as an extra advantage the factorization, is done once, and its cost is amortized over many subsequent solutions.

Problem	Best method in time					Best method in fill-in, $nz(L)/nz(L_A)$				
Bridge	RCM	RCM	RCM	ND	ND	QMD	QMD	QMD	QMD	QMD
1,2,3,4,5	0.052	0.3987	3.875	39.54	342.1	2.74	4.00	5.81	7.99	9.98
Dam	RCM	RCM	ND	ND	ND	OMD	QMD	ND	ND	ND
0,1,2,3,4	0.2989	3.256	34.0	313.1	2755	4.67	6.56	9.03	11.25	13.40
Bricks	RCM	RCM	RCM	ND		QMD	QMD	ND	ND	
1,2,3,4	0.0059	0.119	9.40	654.9		1.29	3.15	7.30	15.42	
Soil	RCM	QMD	ND	ND		QMD	ND	ND	ND	
0,1,2,3	0.035	3.875	246.21	18484		2.29	5.51	12.35	28.4	
Detail	RCM	RCM				RCM	ND			
1,2	11.43	1625				7.32	17.92			
Sluice	RCM	RCM	ND			QMD	ND	ND		
0,1,2	0.4977	52.10	4367			3.55	8.22	17.83		

Table 1: Best direct method in total time and fill-in

Bridge	RCM	RCM	RCM	ND	ND
1,2,3,4,5	0.052	0.3987	3.875	39.54	342.1
Dam	RCM	RCM	ND	ND	RIC2S
0,1,2,3,4	0.2989	3.256	34.0	313.1	2482.9
Bricks	RCM	RCM	RIC2S	RIC2S	
1,2,3,4	0.0059	0.119	4.74	53.1	
Soil	RCM	RIC2S	RIC2S	RIC2S	
0,1,2,3	0.035	2.13	25.2	351.1	
Detail	RCM	RIC2S			
1,2	11.43	1352			
Sluice	RCM	RIC2S	RIC2S		
0,1,2	0.4977	27.4	442.6		

Table 2: Best among iterative RIC2S and direct methods in total time

Method	Ordering	Factorization	Solution	Total time	True res.
Bridge					
$n = 1722$ $nzA = 20938$					
RCM	0.0079	0.0385	0.00373	0.0502	3.65-11
ND	0.0255	0.0664	0.00586	0.0978	3.65-11
QMD	0.0863	0.0468	0.00488	0.1380	3.57-11
$n = 6270$ $nzA = 80726$					
RCM	0.0321	0.3311	0.0356	0.3987	1.81-09
ND	0.1270	0.5167	0.0390	0.6826	1.33-09
QMD	0.6852	0.3735	0.0350	1.0937	1.30-09
$n = 23838$ $nzA = 316752$					
RCM	0.1440	3.4762	0.2550	3.875	1.72-09
ND	0.6476	4.0399	0.2062	4.894	1.25-09
QMD	9.1588	3.5092	0.1952	12.863	1.22-09
$n = 92862$ $nzA = 1254552$					
RCM	0.601	43.306	1.908	45.82	9.25-09
ND	3.296	35.139	1.109	39.54	5.73-09
QMD	138.65	32.046	1.139	171.84	6.15-09
$n = 366462$ $nzA = 4993118$					
RCM	2.552	1100.7	25.01	1127.7	5.18-08
ND	15.86	320.8	5.43	342.1	2.41-08
QMD	2168.6	410.8	6,23	2585.6	2.66-08
Dam					
$n = 3474$ $nzA = 45682$					
RCM	0.0150	0.2603	0.0237	0.2989	5.45-13
ND	0.0565	0.3619	0.0226	0.4411	3.32-13
QMD	0.2291	0.2875	0.0203	0.5370	3.19-13
$n = 13474$ $nzA = 182502$					
RCM	0.0688	3.0018	0.1853	3.256	2.58-12
ND	0.3141	3.4374	0.1314	3.883	1.32-12
QMD	2.5228	2.6335	0.1237	5.280	1.30-12
$n = 53058$ $nzA = 729582$					
RCM	0.3179	47.014	1.4091	48.841	1.30-11
ND	1.7335	31.600	0.6948	34.028	5.17-12
QMD	40.9980	31.200	0.7148	72.913	5.51-12
$n = 210562$ $nzA = 2917310$					
RCM	1.41	1303.30	11.366	1316.0	6.32-11
ND	9.41	300.10	3.558	313.1	1.91-11
QMD	777.35	310.97	3.838	1091.2	2.08-11
$n = 838914$ $nzA = 11667410$					
RCM	5.442	out of	memory	-	-
ND	41.48	2696.36	17.16	2755.0	6.66-11
QMD	12751.00	3819.30	40.97	16612.0	7.54-11

Table 3: Timing and residual results for the $2D$ problems

Method	$OpsFct$	$OpsSlv$	Ops Fct/n	Ops Slv/n	MFlops	Memory Mbytes	$nz(L)$	$nz(L)/$ $nz(L_A)$	$nz(L)$ $/n$
Brdige									
$n = 1722$ $nzA = 20938$									
RCM	5.93+5	8.40+4	344.2	48.7	20.37	0.62	4.03+4	3.71	24.39
ND	6.23+5	7.85+4	362.0	45.6	12.19	0.67	3.75+4	3.46	22.78
QMD	3.53+5	6.21+4	205.1	36.1	10.62	0.61	2.94+4	2.74	18.05
$n = 6270$ $nzA = 80726$									
RCM	7.41+6	5.68+5	1181.8	90.5	25.70	3.30	2.78+5	6.52	45.25
ND	5.31+6	4.10+5	846.4	65.4	11.88	2.99	1.99+5	4.72	32.71
QMD	3.45+6	3.49+5	549.8	55.6	11.16	2.75	1.68+5	4.00	27.81
$n = 23838$ $nzA = 316752$									
RCM	1.05+8	4.18+6	4421.5	175.5	32.56	20.40	2.07+6	12.29	87.77
ND	4.40+7	2.11+6	1845.6	88.7	11.86	13.63	1.03+6	6.21	44.33
QMD	3.49+7	1.98+6	1463.4	82.9	11.00	13.10	9.65+5	5.81	41.48
$n = 92862$ $nzA = 1254552$									
RCM	1.56+9	3.17+7	1.68+4	341.5	37.30	138.53	1.58+7	23.54	170.8
ND	3.76+8	1.08+7	4.04+3	115.9	11.20	63.10	5.28+6	7.99	57.9
QMD	3.40+8	1.08+7	3.67+3	116.3	11.18	63.24	5.31+6	8.01	58.1
$n = 366462$ $nzA = 4993118$									
RCM	2.39+10	2.47+8	6.53+4	673.5	22.07	1011.2	1.23+8	46.04	336.7
ND	3.16+09	5.35+7	8.63+3	145.9	10.11	291.9	2.64+7	9.98	73.0
QMD	3.94+09	6.02+7	1.08+4	164.2	9.81	317.4	2.97+7	11.22	82.1
Dam									
$n = 3474$ $nzA = 45682$									
RCM	6.52+6	3.99+5	1876.9	114.8	28.23	2.16	195960	8.11	57.41
ND	3.91+6	2.56+5	1125.8	73.7	12.23	1.78	124584	5.21	36.86
QMD	2.87+6	2.29+5	825.9	66.0	11.50	1.67	111214	4.67	33.01
$n = 13474$ $nzA = 182502$									
RCM	1.00+8	3.09+6	7438.5	229.6	35.39	14.36	1.53+6	15.79	114.8
ND	3.74+7	1.39+6	2772.2	102.9	11.59	8.48	6.80+5	7.08	51.5
QMD	2.85+7	1.29+6	2107.9	95.4	11.64	8.08	6.29+5	6.56	47.7
$n = 53058$ $nzA = 729582$									
RCM	1.57+9	2.43+7	2.95+4	458.9	34.19	103.06	1.21+7	31.11	229.5
ND	3.24+8	7.06+6	6106.1	133.2	10.62	39.71	3.48+6	9.03	66.6
QMD	3.24+8	7.26+6	6102.6	136.9	10.76	40.41	3.58+6	9.28	68.4
$n = 210562$ $nzA = 2917310$									
RCM	2.48+10	1.93+8	1.18+5	917.9	19.32	777.91	9.64+7	61.79	459.0
ND	2.84+09	3.52+7	1.34+4	167.1	9.65	185.29	1.74+7	11.25	83.6
QMD	2.96+09	3.74+7	1.40+4	177.7	9.72	193.65	1.85+7	11.96	88.9
$n = 838914$ $nzA = 11667410$									
RCM	out of	memory	-	-	-	-	7.69+8	123.02	916.9
ND	2.35+10	1.68+8	2.81+4	199.7	8.83	843.34	8.29+7	13.40	99.9
QMD	3.17+10	1.99+8	3.77+4	236.9	8.36	961.81	9.85+7	15.89	118.4

Table 4: Arithmetic cost, MFlops speed and memory results for $2D$ problems

Method	Ordering	Factorization	Solution	Total time	True res.
Bricks					
$n = 135$	$nzA = 4313$				
RCM	0.0017	0.0040	0.0003	0.0059	2.95-14
ND	0.0019	0.0062	0.0005	0.0086	1.14-14
QMD	0.0109	0.0056	0.0004	0.0170	1.62-14
$n = 675$	$nzA = 32817$				
RCM	0.0119	0.1042	0.0031	0.119	1.05-13
ND	0.0190	0.2303	0.0060	0.255	4.72-14
QMD	0.1552	0.2560	0.0063	0.417	5.92-14
$n = 4131$	$nzA = 255515$				
RCM	0.0987	9.1881	0.1085	9.40	4.85-13
ND	0.2252	16.892	0.1645	17.28	2.55-13
QMD	1.9759	25.543	0.1991	27.72	2.74-13
$n = 28611$	$nzA = 2016125$				
RCM	0.821	1189.4	4.903	1195.1	3.85-12
ND	1.907	650.9	2.134	654.9	1.05-12
QMD	35.654	3537.8	5.607	3579.1	1.55-12
Soil					
$n = 375$	$nzA = 15029$				
RCM	0.0040	0.0297	0.0013	0.0350	9.03-15
ND	0.0071	0.0544	0.0021	0.0636	6.29-15
QMD	0.0475	0.0574	0.0022	0.1070	6.06-15
$n = 2187$	$nzA = 122441$				
RCM	0.0337	4.2352	0.0599	4.329	4.77-14
ND	0.0901	3.8116	0.0619	3.964	2.55-14
QMD	0.5192	3.3087	0.0466	3.875	2.28-14
$n = 14739$	$nzA = 500688$				
RCM	0.3280	672.82	1.918	675.07	3.69-13
ND	1.3481	243.76	1.110	246.21	1.03-13
QMD	8.5856	707.95	1.549	718.09	1.43-13
$n = 107811$	$nzA = 7925773$				
RCM	2.643	out of	memory	-	-
ND	15.627	18420.3	27.695	18464.0	4.67-13
QMD	319.297	out of	memory	-	-
Detail					
$n = 11484$	$nzA = 749516$				
RCM	0.235	10.87	0.32	11.43	4.01-10
ND	0.737	61.15	0.55	62.43	2.71-10
QMD	7.021	68.36	0.60	74.38	2.77-10
$n = 79065$	$nzA = 5726409$				
RCM	1.873	1615.21	7.61	1624.7	3.78-09
ND	9.297	3971.28	9.84	3990.4	1.66-09
QMD	164.27	6946.96	20.76	7132.0	2.09-09
Sluice					
$n = 2172$	$nzA = 102094$				
RCM	0.0273	0.446	0.0243	0.4977	1.58-13
ND	0.0700	1.241	0.0338	1.3448	9.98-14
QMD	0.4659	1.013	0.0310	1.0126	1.05-13
$n = 12957$	$nzA = 775815$				
RCM	0.3012	51.283	0.525	52.10	1.35-12
ND	0.9229	69.925	0.567	71.42	8.28-13
QMD	8.0626	103.620	0.685	112.37	8.90-13
$n = 87159$	$nzA = 6007351$				
RCM	2.46	5461.27	26.490	5490.2	1.60-11
ND	10.48	4346.22	10.285	4367.0	5.99-12
QMD	193.73	8992.54	15.016	9201.3	8.22-12

Table 5: Timing and residual results for problem for the $3D$ problems

Method	$OpsFct$	$OpsSlv$	Ops Fct/n	Ops Slv/n	MFlops	Memory Mbytes	$nz(L)$	$nz(L)/$ $nz(L_A)$	$nz(L)$ $/n$
Bricks									
$n = 135$ $nzA = 4313$									
RCM	6.12+4	6.10+3	453.4	45.2	22.99	0.08	3237	1.52	24.98
ND	5.66+4	6.47+3	419.3	47.9	12.98	0.08	3099	1.45	23.96
QMD	4.35+4	5.72+3	322.8	42.4	11.32	0.08	2726	1.29	21.19
$n = 675$ $nzA = 32817$									
RCM	2.89+6	9.51+4	4283.7	140.9	32.83	0.82	5.46+4	3.30	81.9
ND	2.70+6	1.06+5	4003.7	156.6	13.13	0.84	5.21+4	3.16	78.3
QMD	2.99+6	1.06+5	4424.2	156.4	12.70	0.84	5.21+4	3.15	78.2
$n = 4131$ $nzA = 255515$									
RCM	2.06+8	1.96+6	4.98+4	472.9	22.90	12.1	1.18+6	9.15	287.5
ND	1.68+8	1.90+6	4.06+4	458.8	10.12	10.6	9.43+5	7.30	229.6
QMD	2.50+8	2.25+6	6.05+4	544.6	9.93	12.0	1.12+6	8.67	272.3
$n = 28611$ $nzA = 2016125$									
RCM	2.11+10	5.19+7	7.36+5	1813	17.75	265.3	3.16+7	30.93	1105
ND	7.98+09	3.15+7	2.79+5	1103	12.34	146.9	1.57+7	15.42	551
QMD	2.89+10	5.87+7	1.01+6	2050	8.19	250.6	2.93+7	28.68	1025
Soil									
$n = 375$ $nzA = 15029$									
RCM	7.63+5	3.96+4	2036	105.6	30.05	0.34	19506	2.58	53.02
ND	6.30+5	3.56+4	1680	94.9	13.36	0.34	17418	2.31	47.45
QMD	6.62+5	3.53+4	1765	94.0	13.15	0.34	17255	2.29	47.01
$n = 2187$ $nzA = 122441$									
RCM	9.23+7	1.10+6	4.22+4	501.3	22.29	5.66	5.46+5	8.80	250.7
ND	4.10+7	6.87+5	1.87+4	314.1	11.09	4.24	341+5	5.51	157.1
QMD	5.29+7	7.50+5	2.42+4	343.0	16.40	4.49	3.73+5	6.02	171.5
$n = 14739$ $nzA = 500688$									
RCM	1.18+10	3.32+7	8.02+5	2250	17.64	138.3	1.66+7	33.1	1125
ND	2.20+9	1.24+7	1.49+5	839	9.08	60.2	6.17+6	12.3	420
QMD	8.19+9	2.20+7	5.56+5	1491	11.61	96.9	1.10+7	21.9	745
$n = 107811$ $nzA = 7925773$									
RCM	out of	memory					5.25+8	130.9	4875
ND	1.45+11	2.28+8	1.34+6	2115	7.87	974.5	1.14+7	28.4	1057
QMD	out of	memory					2.93+8	73.1	2722
Detail									
$n = 11484$ $nzA = 749516$									
RCM	3.58+8	5.57+6	3.12+4	485	34.11	30.22	2.77+6	7.32	243
ND	5.95+8	6.04+6	5.18+4	526	9.90	32.80	3.01+6	7.93	263
QMD	6.60+8	6.41+6	5.75+4	558	9.80	34.31	3.19+6	8.42	273
$n = 79065$ $nzA = 5726409$									
RCM	2.87+10	1.31+8	3.63+5	1659	17.90	568.67	6.55+7	22.60	830
ND	3.32+10	1.04+8	4.20+5	1315	8.40	471.40	5.19+7	17.92	658
QMD	5.67+10	1.40+8	7.17+5	1775	8.18	610.43	7.01+7	24.18	888
Sluice									
$n = 2172$ $nzA = 102094$									
RCM	1.46+7	4.41+5	6.71+3	203.1	35.48	2.93	2.19+5	4.24	101.7
ND	1.43+7	4.06+5	6.60+3	187.2	12.25	2.91	2.01+5	3.90	93.6
QMD	1.16+7	3.70+5	5.32+3	170.7	12.20	2.78	1.83+5	3.56	85.3
$n = 12957$ $nzA = 775815$									
RCM	1.03+9	9.33+6	7.94+4	720.4	20.36	44.94	4.66+6	11.84	360.3
ND	6.75+8	6.49+6	5.21+4	500.5	9.80	34.90	3.23+6	8.22	250.3
QMD	9.56+8	7.52+6	7.38+4	580.6	9.34	38.98	3.75+6	9.54	290.3
$n = 87159$ $nzA = 6007351$									
RCM	9.69+10	2.36+8	1.11+6	2702	17.78	970.41	1.17+8	38.65	1351
ND	3.62+10	1.09+8	4.15+5	1247	8.36	493.02	5.43+7	17.83	623
QMD	6.92+10	1.52+8	7.94+5	1738	7.71	656.82	7.56+7	24.85	869

Table 6: Arithmetic cost, MFlops speed and memory results for $3D$ problems

4 The $U^TU + U^TR + R^TU$ -decomposition based Conjugate Gradient preconditionings

In this section, the "second-order" incomplete triangular factorization, proposed in [19], is discussed. We consider the problem

$$Ax = \mathbf{b}, \tag{4.1}$$

where A is a sparse $n \times n$ symmetric positive definite (s.p.d.) matrix, and present numerical results obtained with the use of the new robust preconditioning techniques [19] which considerably accelerates the convergence of the PCG iterations applied to the solution of (4.1) and, at the same time, requires relatively small amount of computational resources on each iteration.

As is well known, only few incomplete factorization type techniques are robust (but not always efficient) when applied to general s.p.d. matrices, such as Point Jacobi, Point SSOR, and the Robust Incomplete Cholesky (RIC) of Ajiz and Jennings [1], and (up to its trial-and-error nature) the IC performed on the shifted original matrix as was proposed by Manteuffel [21]. (Some versions of the latter were also considered in [23] and [18].)

With the "robustness" of IC we mean here the theoretically proven absence of the "pivot breakdown" events. Indeed, when applying most known IC algorithms to a general s.p.d. matrix, it is quite typical that at some step a negative pivot appears, and is replaced by some "proper" positive number. The number of such pivot breakdowns can hardly be estimated in advance, and the condition number for the so obtained triangular factor can be *unpredictably large*. Therefore, the resulting preconditioning quality can be very poor, sometimes yielding even an increase of the total solution time as compared to the simplest Jacobi-CG iterative solver. A more proper strategy seems to discard all the IC data obtained as soon as the first breakdown occurs, and, e.g., increase the diagonal shift and restart the factorization as recommended in [23], [18].

Here we will consider a somewhat more complicated and expensive to compute factorization which gives a considerably more efficient preconditioning. This new factorization is based on the decomposition

$$A = U^TU + U^TR + R^TU - S, \tag{4.2}$$

where R (strictly upper triangular) and S (nonnegative definite) are the error matrices (having small elements by the construction), and U (upper triangular) is the approximation to the exact right Cholesky factor of A to be used in the PCG method. The absence of pivot breakdowns and (relative) well-conditioning of U are safely guaranteed by the properties of this decomposition. A reasonable balance of the error terms is attained when, e.g. the diagonal elements of the matrices RR^T and S are of the same order, provided that A is symmetrically scaled to the unit diagonal. Such preconditioning will be further referred to as "Robust Incomplete Cholesky 2nd order Stabilized" factorization (RIC2S, or simply IC2).

4.1 Some theoretical results for the RIC2S preconditioned CG method

Denoting the approximate Cholesky preconditioning matrix by

$$H = (U^TU)^{-1}, \tag{4.3}$$

one can write the PCG solution method (see, for instance, [2]) for the problem (1.1) as follows:

$$\begin{aligned} \mathbf{r}_0 &= \mathbf{b} - A\mathbf{x}_0, \\ \mathbf{p}_0 &= H\mathbf{r}_0; \end{aligned}$$

$$\textbf{for} \;\; i = 0, 1, \dots :$$

$$\alpha_i = \frac{\mathbf{r}_i^T H \mathbf{r}_i}{\mathbf{p}_i^T A \mathbf{p}_i}, \tag{4.4}$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{p}_i \alpha_i, \tag{4.5}$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - A\mathbf{p}_i \alpha_i, \tag{4.6}$$

$$\beta_i = \frac{\mathbf{r}_{i+1}^T H \mathbf{r}_{i+1}}{\mathbf{r}_i^T H \mathbf{r}_i}, \tag{4.7}$$

$$\mathbf{p}_{i+1} = H\mathbf{r}_{i+1} + \mathbf{p}_i \beta_i. \tag{4.8}$$

As is known, cf.[2], the following estimate for the A-norm of the error holds:

$$\frac{\sqrt{\mathbf{r}_i^T A^{-1} \mathbf{r}_i}}{\sqrt{\mathbf{r}_0^T A^{-1} \mathbf{r}_0}} \le \frac{2}{\sigma^i + \sigma^{-i}}, \qquad \sigma = \frac{\sqrt{\varkappa} + 1}{\sqrt{\varkappa} - 1}, \tag{4.9}$$

and the corresponding standard upper bound for the iteration number needed for the ε times reduction of the error norm $\sqrt{\mathbf{r}_i^T A^{-1} \mathbf{r}_i}$ is therefore, approximately,

$$i_{\varkappa}(\varepsilon) = \frac{1}{2} \sqrt{\varkappa} \log \frac{2}{\varepsilon}, \tag{4.10}$$

where

$$\varkappa = \varkappa(HA) = \lambda_{\max}(HA) / \lambda_{\min}(HA) \tag{4.11}$$

is the spectral condition number of the preconditioned matrix HA.

In [19], an estimate was obtained for the condition number $\varkappa(HA)$ resulting from the use of the RIC2S preconditioning (1.2), (2.1). Supposing that the coefficient matrix A is symmetrically scaled to the unit diagonal, i.e, $a_{ii} = 1, i = 1, \dots, n$, one can formulate the following result supporting a proper balance of threshold parameters in the RIC2S factorization: (cf. Theorem 5.2 of [19]): if S is positive definite, then for the spectral condition number $\varkappa$ of the preconditioned matrix

$$M = U^{-T} A U^{-1}$$

we have the estimate

$$\sqrt{\varkappa(M)} \le \sqrt{1 + \|RS^{-1}R^T\|} \left(\sqrt{\frac{\|RR^T\|}{\lambda_{\min}(A)}} + \sqrt{1 + \frac{\|RR^T\| + \|S\|}{\lambda_{\min}(A)}} \right), \tag{4.12}$$

where $\lambda_{\min}(A)$ is the minimum eigenvalue of A.

Henceforth, the matrices RR^T and S should be of the same order (e.g., with respect to the drop tolerance parameter). This result justifies the choice of the parameters τ_1, τ_2, τ_3 involved in the RIC2S algorithm presented below.

REMARK 4.1 There exist also some results on the superlinear convergence of the PCG method, see e.g., [28, 30], where the so-called K-condition number of the preconditioned matrix HA, see [5], is used. In [19], some nontrivial bounds reflecting the special properties of the RIC2S preconditioning were given for the K-condition number. However, since the recommendations on the choice of the incomplete factorization threshold parameters stay qualitatively the same (for instance, it was shown in [19] that the RIC2S-CG iteration number can be bounded from above by a certain monotonically increasing function of the quantity $trace(RR^T + S)$), here we omit these additional details for the sake of brevity.

The quality of a preconditioner can be measured by the reduction of the estimated total computational cost which can be taken as the cost of one iteration multiplied by an estimate of the number of iterations plus the costs of the preconditioner evaluation. The most remarkable property of the RIC2S preconditioning is that it provides an essential reduction of the PCG iteration number using relatively few number of nonzero elements in the preconditioning matrix U. Some numerical evidence supporting this claim was presented in [19], where a comparison of RIC2S with the preconditioning of Ajiz and Jennings [1] was given.

4.2 The Second order Cholesky decomposition algorithm

Let us recall the description of the RIC2S (Robust 2-nd order Incomplete Cholesky Stabilized) algorithm as was given in [19].

Algorithm RIC2S: Robust Incomplete Cholesky 2nd order Stabilized factorization
Input: Unidiagonal s.p.d. matrix A; truncation threshold parameters τ_1, τ_2, and diagonal shift parameter τ_3;

1. Initialize the auxiliary diagonal matrix $D = \{d_i\}_{1\le i\le n}$:
   for $i = 1, \dots, n$:
   $d_i = (1 + \tau_3)a_{ii}$;
   end for
2. Main loop by rows of A to obtain rows of U and R:
   for $i = 1, \dots, n$:
3. Initialize the Row Accumulator vector v by the i-th row of A:
   for $j = i + 1, \dots, n$:
   $v_j = a_{ij}$
   end for
4. Loop over previous rows of U and R:
   for $k = 1, \dots, i - 1$:
5. Update the Row Accumulator vector v:
   for $j = i + 1, \dots, n$:
   $v_j := v_j - u_{ki}u_{kj} - u_{ki}r_{kj} - r_{ki}u_{kj}$
   end for
   end for
6. Sparsify the Row Accumulator vector v and update d:
   for $j = i + 1, \dots, n$:

$\zeta := |v_j|/\sqrt{d_j}$
if $\zeta \le \tau_2$ **then**
$v_j = 0$
$d_i := (1+\zeta)d_i$
$d_j := (1+\zeta)d_j$
end if
end for

7. Normalize the Row Accumulator vector v:
$u_{ii} = \sqrt{d_i}$:
for $j = i+1, \ldots, n$:
$v_j := v_j/u_{ii}$
end for

8. Split the row accumulator vector to obtain the i-th row of U and R
for $j = i+1, \ldots, n$:
if $|v_j| \ge \tau_1$ **then**
$u_{ij} := v_j,$
else
$r_{ij} := v_j;$
end if
end for

9. Update the auxiliary diagonal matrix d:
for $j = i+1, \ldots, n$:
$d_j := d_j - u_{ij}^2$
end for
end for

REMARK 4.2 The actual implementation of the above algorithm is based on the use of sparse matrix storage and processing of the matrices U and R.

REMARK 4.3 In the actual implementation of RIC2S used in the experiments, the coefficient matrix A is not expected to be scaled to the unit diagonal. Instead, the diagonal matrix $\mathrm{Diag}(\delta_1, \ldots, \delta_n) = Diag(A)^{-1/2}$ is calculated in advance, and the Step 3 is modified as follows:

for $j = i+1, \ldots, n$:
$v_j = \delta_i a_{ij} \delta_j$
end for

and the column scaling step of G,

for $i = 1, \ldots, n$:
for $j = i, \ldots, n$:
$g_{ij} := g_{ij}/\delta_j$
end for
end for

is added in the very end of RIC2S calculations.

REMARK 4.4 As is seen, the preconditioning algorithm contains three parameters. However, in our experiments we used the reduced parametrization of RIC2S, exactly as proposed in [19]:

$$\tau_1 = \tau, \qquad \tau_2 = \tau^2, \qquad \tau_3 = 2\tau^2,$$

with some $0 < \tau < 1.0$. Thus, RIC2S acts as a one-parameter algorithm, similarly to the simpler IC method of Ajiz and Jennings [1]. Typically, only certain values of τ from a predetermined discrete set $\{0.03, 0.02, 0.01, 0.007, 0.005, 0.003, 0.002, 0.001, \ldots\}$. were tested for the use. This was possible due to the relative insensitivity of the RIC2S-CG performance with respect to the choice of the parameter τ (i.e., some 30-50% variations in τ near its "optimum" value do not change the RIC2S-CG total solution time essentially, which is not the case for the diagonal shift based techniques [18, 23, 21]). However, the RIC2S preconditioning time increases rather fast as τ decreases, especially for the discretizations of the 3D problems.

4.3 Algorithmic implementation of the PCG iteration

Given an incomplete factorization factor U, we use a very simple and straightforward implementation of the PCG solution algorithm. Both the matrices A (symmetric) and U (upper triangular) are represented by their upper triangles. The matrices are kept in a standard row-wize sparse matrix format. Namely, If the number of nonzero elements of A is nz, then it is stored as two integer and one real arrays, say, $ia(1 : n+1)$ containing pointers to the last elements of each row, $ja(1 : nz)$ containing the column indices taken row-by-row and ordered by the increase within each row, and the array $a(1 : nz)$ containing the corresponding values of nonzero elements. The matrix U is defined by similarly organized arrays ig, jg, g. Typically, more than 90% of the computational time is spent within the following two routines: the one implementing multiplication of the coefficient matrix A by a vector v,

```
      subroutine wsamv (n, a, ja, ia, v, w)
      double precision a(1), v(1), w(1), sum, save, zero
      parameter        (zero=0.0d0)
      integer          ja(1), ia(1), n, i, j, k
c  this routine computes  w = a*v, where
c  a  is represented by its upper triangle in
c  the compressed row-wise format (ordered)
      do 10 i = 1, n
         w(i) = zero
   10 continue
      do 20 i = 1, n
         sum = w(i)
         save = v(i)
         do 30 k = ia(i)+1, ia(i+1)
            j = ja(k)
            sum = sum + a(k)*v(j)
            w(j) = w(j) + save*a(k)
   30    continue
         w(i) = sum
   20 continue
```

```
      return
      end
```

and the one implementing the preconditioner by performing two successive back substitutions with U^T and U:

```
      subroutine ingtg (n, g, jg, ig, v, z)
      double precision g(1), v(1), z(1), save
      integer          jg(1), ig(1), n, l, i, j
c  this routine solves g^t*g*z = v, where
c  g  is an upper triangular matrix in the
c  compressed row-wise format (ordered)
      do i = 1, n
         z(i) = v(i)
      end do
      do i = 1, n
         save = z(i)/g(ig(i)+1)
         z(i) = save
         do l = ig(i)+2, ig(i+1)
            j = jg(l)
            z(j) = z(j) - g(l)*save
         end do
      end do
      do i = n, 1, -1
         save = z(i)
         do l = ig(i)+2, ig(i+1)
            save = save - z(jg(l))*g(l)
         end do
         z(i) = save/g(ig(i)+1)
      end do
      return
      end
```

These routines contain no hidden optimization tricks, cf. [22]. On the contrary, one can rewrite them (and properly redefine the meaning of the array g) in an obvious way in order to improve their performance by avoiding redundant n multiplications and replacing $2n$ divisions by n multiplications in “wsamv” and “ingtg”, respectively.

4.4 Numerical experiments

In this section, some RIC2S preconditioning tuning options are compared, and results are also given for the diagonally preconditioned Jacobi-CG method (the latter also gives an estimate for the spectral condition number of $A_S = D_A^{-\frac{1}{2}} A D_A^{-\frac{1}{2}}$, i.e., the original matrix scaled by its main diagonal).

The ”nearly best” performance results (with respect to the total solution time for a single right hand side) for all problems are presented in Table 7, where in the first three columns the order of the system n, the total number of nonzero elements in the matrix $\mathrm{nz}(A)$, and the spectral condition number of A_S are given. The RIC2S tuning options are given in brackets next to the problem identifier showing the value of the threshold parameter, as well as the type of preordering is identified: “MD” for the minimum degree preordering [21], and “PR” for the profile/bandwidth reduction reordering applied to the coefficient matrix A and performed with the use of some Cuthill-McKee type procedure. The column “density” shows the quantity $nz(U)/nz(D_A + U_A)$ which indicates the (relative) amount of memory needed for the preconditioning, and in the next column the PCG iteration time and the total solution time in seconds are given (the latter includes reordering, RIC2S factorization, and PCG iterations). The iterations are started from $x_0 = 0$, so that $r_0 = b$, and performed until the stopping criterion

$$\|r_k\| \le 10^{-8}\|r_0\|, \qquad \varepsilon = 10^{-8},$$

is satisfied. The actual value of the relative residual norm $\|b - Ax_k\|/\|b\|$ is given in the last column as “true res.”.

REMARK 4.5 In Tables 8, 9, 10 and Figures 6(a), 6(a) the data obtained with $\varepsilon = 10^{-10}$ were used; however, the observed deviation from the results corresponding to $\varepsilon = 10^{-8}$ was rather negligible.

The dependence of the RIC2S-CG performance on the choice of the dropping threshold parameter τ is illustrated in Figure 7(a),7(b), 7(c) for the three largest 3D problems, where we present separate timings for different solution stages. Here, the time for preconditioning, for the iterations, and the total solution time are given versus the (relative) volume of working memory (which is mainly determined by the density of the RIC2S factor) for the three largest 3D problems (Soil, Sluice, and Detail).

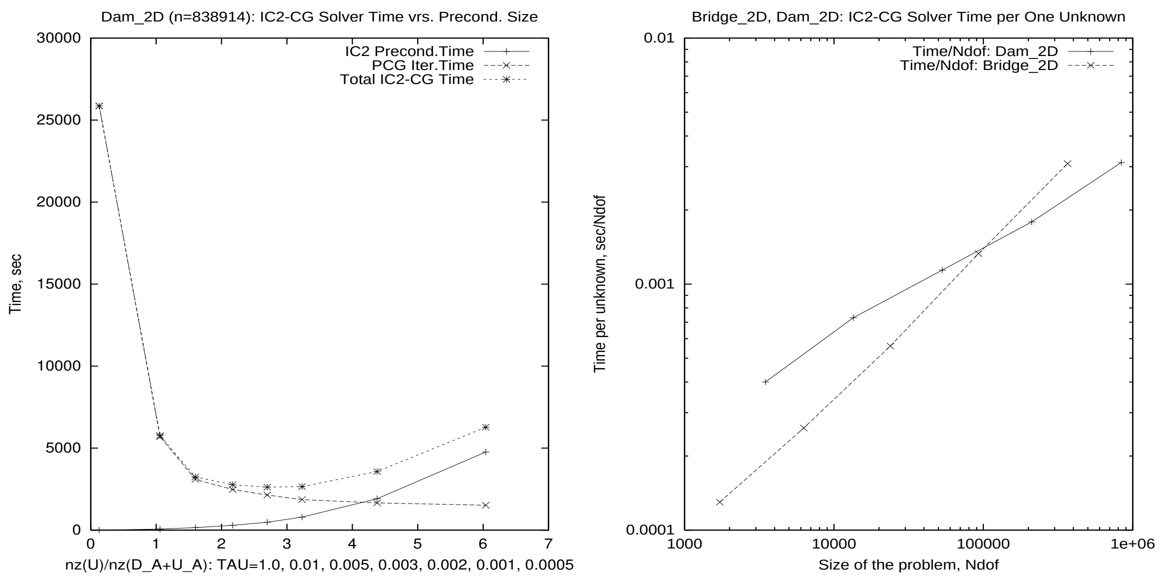
The leftmost points on the plots in Figure 7(a) - 7(c) correspond to the use of the simplest possible (Jacobi) preconditioning by the inverse diagonal of A, i.e. $H = D_A^{-1}$. In Figure 7(d) we present the total solution time per unknown versus the problem size for the same three families of 3D problems.

In Tables 8, 9 we present typical results obtained for the two most ill-conditioned problems: the largest ‘Bridge’ problem, and the largest ‘Detail’ problem. Note that columns 3-6 of Table 9 represent the same data as Figure 7(c) does. By “rev.” we have denoted the application of the reverse ordering of the original matrix. (Since the original matrix has a special ”inverse multilevel” ordering, the reversion of it resulted in some ordering similar, in a sense, to the minimum degree one.) Next we give the time in seconds t_{pr}, t_{it}, and t_{tot} needed for the evaluation of the preconditioner, for the subsequent PCG iterations, and the total solution time, respectively. Finally, we present an estimate for the spectral condition number $\varkappa(M)$ and for the maximum eigenvalue $\|M\|$ of the preconditioned matrix $M = U^{-T}AU^{-1}$ obtained in a standard way from the extreme eigenvalues of the tridiagonal matrix constructed from the PCG coefficients α_i and β_i. One can clearly observe an essential reduction of the condition number of M with the decrease of the RIC2S parameter τ, as is predicted by the theory.

It should be noted that for the large 3D problems the MD reordering does not give any improvement in the performance as compared to the PR one. Moreover, for the largest 3D problems, only RIC2S with a profile/bandwidth reordering could properly fit into the limited (1 GB) in-core memory. For the PR reordering, the total volume of memory required for the solution (including

size n	$nz(A)$	$C(A_S)$	iter.	density	time$_{\mathrm{iter}}$	time$_{\mathrm{total}}$	true res.
Bridge (3.0-04 /MD)							
1722	20937	0.399+07	10	2.42	0.07	0.21	3.34-10
6270	80727	0.274+08	16	3.16	0.71	1.53	4.31-09
23838	316755	0.138+09	27	3.93	5.70	12.1	9.75-09
92862	1254552	0.596+09	55	4.64	54.0	110.5	1.24-08
366462	4993117	0.246+10	102	5.02	413.7	852.7	5.43-08
Bridge (3.0-04 /PR)							
1722	20937	0.399+07	11	3.01	0.01	0.31	1.71-09
6270	80727	0.274+08	19	4.13	0.98	2.70	4.46-09
23838	316755	0.138+09	42	5.72	10.27	25.4	2.65-09
92862	1254552	0.596+09	89	7.18	105.1	227.7	1.41-08
366462	4993117	0.246+10	317	7.75	1620.6	2765.6	8.42-08
Dam (3.0-03 /PR)							
3474	45862	0.130+06	19	2.70	0.38	1.42	8.59-09
13474	182502	0.579+06	37	2.75	3.19	9.83	7.74-09
53058	729582	0.241+07	82	2.72	28.9	59.8	8.03-09
210562	2917328	0.979+07	172	2.71	243.9	374.2	6.69-09
838914	11667444	0.394+08	353	2.70	1991.2	2482.9	8.96-09
Bricks (3.0-02 /PR)							
135	4306	0.148+03	15	0.79	0.01	0.04	5.61-09
675	32785	0.606+03	29	0.53	0.10	0.47	8.80-09
4131	255415	0.245+04	58	0.44	2.04	4.74	6.15-09
28611	2016264	0.982+04	117	0.41	32.5	51.3	7.00-09
212355	16024772	0.393+05	238	0.40	514.6	656.5	9.90-09
Soil (3.0-02 /PR)							
375	15017	0.212+04	26	0.60	0.04	0.18	9.09-09
2187	122450	0.980+04	58	0.47	0.92	2.13	3.46-09
14739	986644	0.411+05	129	0.42	15.9	25.2	9.89-09
107811	7925485	0.166+06	261	0.40	279.0	351.1	9.79-09
823875	63549456	0.668+06	561	0.39	4870.1	5431.1	9.32-09
Detail (2.0-02 /PR)							
11484	749446	0.109+09	374	0.57	64.1	73.6	7.57-09
79065	5726430	0.507+09	1518	0.51	1260.2	1352.3	1.09-08
584127	44668305	0.215+10	3470	0.48	21801.7	22566.3	3.09-08
Sluice (2.0-02 /PR)							
2172	102114	0.752+04	58	0.69	0.89	1.78	7.79-09
12957	775743	0.406+05	167	0.54	19.1	27.4	5.83-09
87159	6007266	0.192+06	446	0.47	372.4	442.6	8.33-09
633579	47193500	0.856+06	1028	0.43	6719.7	7295.5	9.62-09

Table 7: Matrix properties, tuned RIC2S parameters, and PCG results for all problems



(a) Dam 2D Problem: time vs. density

(b) Time per D.O.F vs. problem size

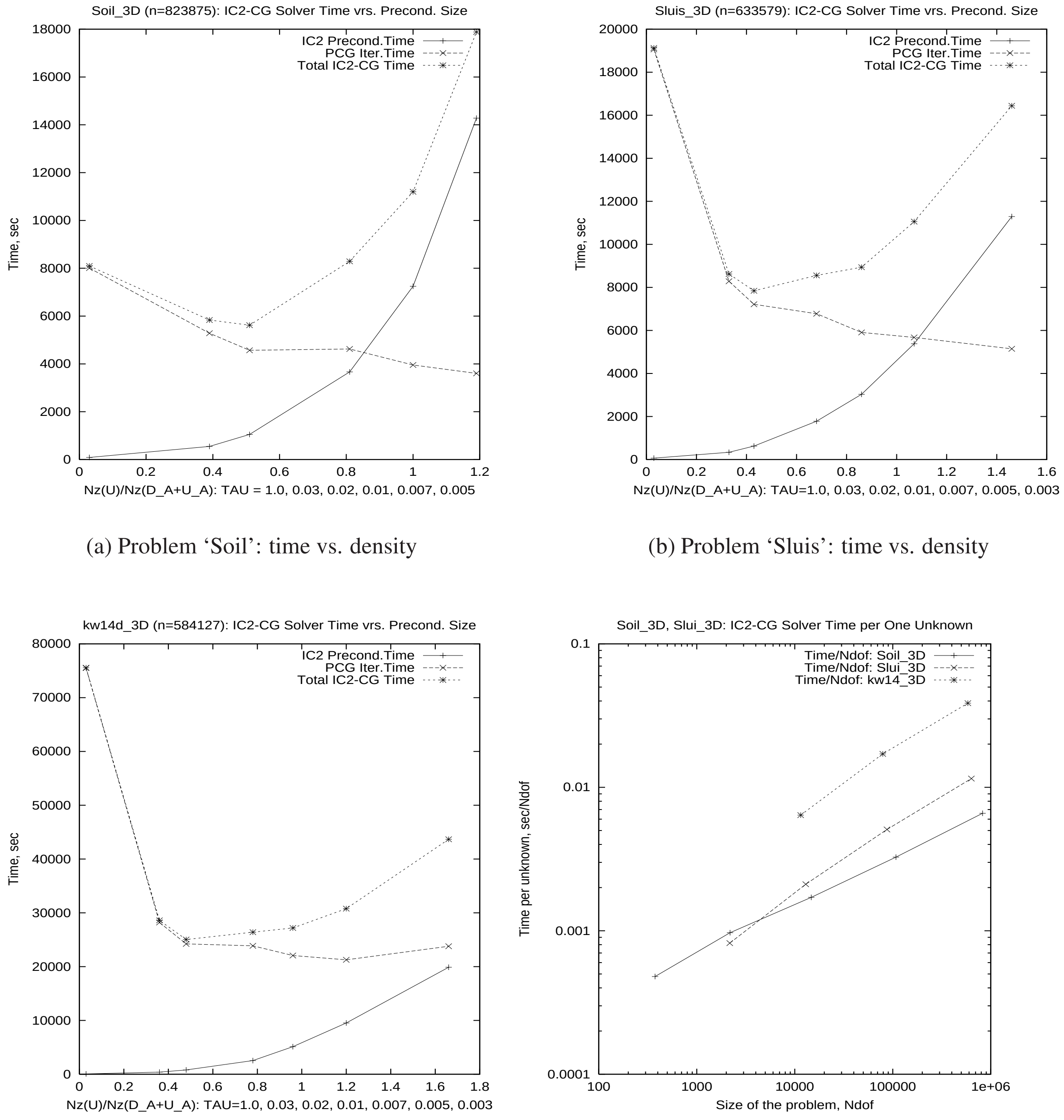
Figure 6: Timing results for 2D problems (‘Bridge’, ‘Dam’)

the matrix data) can be approximately estimated from the data presented in Table 1 using the following formula,

$$\mathrm{MEMORY(in\ \ bytes)} \approx 12\Big(\mathrm{nz}(A)\cdot\max\left(1, \frac{1+\mathrm{density}}{2}\right) + 6n\Big),$$

which is well within 1 GB for all the problems considered. For the largest three 3D problems, ‘Soil’, ‘Detail’, and ‘Sluice’, the observed actual amount of memory used was 856Mb, 645Mb, and 634Mb, respectively.

In Table 10 we present some more detailed performance data for the largest ‘Detail’ problem, such as Mflops rate for different solution stages and the total number of multiplications “OPS” (the total number of arithmetic operations is approximately 2∗OPS). As the dropping tolerance τ decreases, the average performance deteriorates, mainly due to larger time spent in the incomplete factorization stage. Also, one can observe the preconditioner “saturation” effect even at the level of the total operation count, when the increasing preconditioning costs outweigh the operation count reduction attained at the iteration stage.



(a) Problem 'Soil': time vs. density

(b) Problem 'Sluis': time vs. density

(c) Problem 'Detail': time vs. density

(d) Time per D.O.F vs. problem size

Figure 7: Timing results for 3D problems ('Soil', 'Sluice', 'Detail')

τ	perm.	density	t_{pr}	t_{it}	t_{tot}	iter.	$C(M)$	$\|M\|$
Jacobi	PR	0.03	60.4	75503.9	75564.3	19776	2.15+10	4.724
3.0-02	PR	0.36	392.7	28209.4	28602.1	4910	1.10+08	1.431
2.0-02	PR	0.48	801.1	24234.3	25035.4	3852	7.60+07	1.535
1.0-02	PR	0.78	2536.1	23866.4	26402.5	3061	4.22+07	1.669
7.0-03	PR	0.96	5129.7	22064.7	27194.4	2529	3.36+07	1.835
5.0-03	PR	1.20	9522.5	21265.0	30787.5	2219	2.46+07	1.877
3.0-03	PR	1.66	19889.8	23781.9	43671.7	1816	1.48+07	1.816

Table 9: Results for Problem 'Detail', ($n = 584127$)

τ	perm.	density	t_{pr}	t_{it}	t_{tot}	iter.	$C(M)$	$\|M\|$
Jacobi	none	0.14	0.8	53250.7	53251.5	58101	2.46+09	3.423
1.0-03	PR	4.34	606.2	3235.0	3841.2	979	5.97+05	1.616
3.0-04	PR	7.75	1320.8	2019.9	3340.7	386	1.04+05	1.882
2.0-04	PR	9.31	1661.3	1570.1	3231.4	263	5.71+04	2.072
3.0-04	MD	5.02	505.4	594.4	1099.8	142	8.05+03	1.682
2.0-04	MD	5.42	576.2	555.1	1131.3	125	3.62+03	1.756
3.0-04	rev.	5.13	551.2	494.0	1045.2	129	5.42+05	1.587

Table 8: Results for Problem 'Bridge', ($n = 366462$)

τ	perm.	Mflops$_{pr}$	Mflops$_{it}$	Mflops$_{tot}$	OPS$_{tot}$
Jacobi	PR	1.5	25.8	25.8	0.975+12
3.0-02	PR	3.7	22.4	22.2	0.316+12
2.0-02	PR	3.7	22.2	21.6	0.270+12
1.0-02	PR	3.5	21.5	19.8	0.260+12
7.0-03	PR	2.9	21.0	17.6	0.239+12
5.0-03	PR	2.7	21.4	15.6	0.240+12
3.0-03	PR	2.6	18.8	11.4	0.249+12

Table 10: Performance data for Problem 'Detail', $n = 584127$)

5 Conclusions

Problems arising in 'real-life' can be much more demanding on the numerical solution methods to be used than model type problems are. For instance, when solving ill-conditioned systems, direct methods are competitive with iterative even for fairly large sized problems. When the size of the problems increases, the direct methods loose eventually due to the high demand of memory resources and due to fast increase of computational complexity, in particular for the factorization. However, the possibility to use computers with memory capacity which is several orders of magnitude larger than that available say, some 10 years ago, has shifted considerably the border of the matrix sizes up to which a direct solution method can be used.

The winner among the direct methods for the larger sized problems turned out to be the nested dissection method. The tests show quite substantial differences in performance of the different methods using the software described earlier. The quotient minimum degree method requires about a factor two more time for factorization and also for solution, except for the 'Bridge' problem, where the difference was smaller. However, it required then a very large computer time for the reordering. An important observation is that for all methods, the solution time was just a small fraction (several hundreds smaller) of the factorization time. This indicates that for problems with a sufficient number of right hand sides, where the initial factorization cost can be amortized over the different solves, direct methods will win over iterative methods.

Given a fixed number of right hand sides in an iterative method of incomplete factorization type, one can adjust the threshold parameter to reduce the solution cost per system. Besides saving much in memory, the incomplete factorization method requires namely a substantially smaller time for construction of the preconditioner than the full factorization (direct) method does. One can therefore expect that the preconditioned iterative solution method will be more efficient also in such situations. Nevertheless, the timings for different values of the threshold parameter showed that the iteration time did not reduce monotonically with decreasing values of this parameter, indicating that one must choose very small values of it to come closer to a direct method in solution time. In such a case there will be only little savings in memory.

Therefore, other approaches must be taken to construct more efficient iterative solution methods, also for the case of several right hand sides.

In Part II of this study such methods are presented and tested on the same set of problems, as in Part I. In these methods, a coarse mesh solver is used as a preconditioner (for the Schur complement reduced methods) or as a part of an approximate inverse in the ASP method. Clearly, thereby it can be advantageous to use a direct solver or an accurate incomplete factorization method for the coarse mesh, as the size of this is substantially smaller than the global system. Again, in the choice between the incomplete factorization and the direct solution method, the savings in memory indicate that the choice of the former is to be recommended where memory limitations are of concern. Also, the incomplete factorization method showed robustness with respect to problem parameters, i.e., it is comparable with the behaviour of the direct solution method in this sense.

If many outer iterations are required, typically in the order of hundreds, then actually the direct method may be recommendable as a coarse mesh solver, as the initial factorization cost can then be amortized over those many iterations, assuming that the high memory demand can be met. This also indicates that a deflation method, which requires exact solution of the inner systems can be competitive with the ASP method even for quite large size of the inner (coarse mesh) system. On the other hand, if the number of outer iterations is of order of a few tenths as it is for the Schur complement based method and, hence, there occur few coarse mesh solutions, then it is more efficient to use the incomplete factorization preconditioned solver as a coarse mesh (inner iteration) solver.

Thus, Part I of this study has given important insight in how to construct still more efficient preconditioners for the conjugate gradient method, and such issues are discussed in the following Part II.

Acknowledgements

The work reported in this paper has been supported by The Netherlands Foundation for Technical Research (STW), grant NNS.4683 *High Performance Computing of problems in structural Engineering*. The help provided by ir. A. de Boer, Dr.ir. M. Hendriks and Dr.ir. P. Nauta in defining the benchmark test problems was invaluable for the progress reported here.

References

[1] M. A. Ajiz and A. Jennings. A robust incomplete Cholesky-conjugate gradient algorithm. *Int. J. Numer. Methods Engrg.*, 20, 949-966, 1984.

[2] P. R. Amestoy, T. A. Davis and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matr. Anal. Appl.*, 17, 886-905, 1996.

[3] C. Ashcraft and J.W.H. Liu. Robust ordering of sparse matrices using multisection. *SIAM J. Matrix Anal. Appl.*, 19, 816-832, 1998.

[4] O. Axelsson. A class of iterative methods for finite element equations. *Computer Meth. Appl. Mech. Engrg.*, 9, 123-137, 1976.

[5] O. Axelsson. *Iterative solution methods.* Cambridge University Press, Cambridge, 1994.

[6] O. Axelsson. On iterative solvers in structural mechanics, separate displacement orderings and mixed variable methods, *Mathematics and Computers in Simulation*, 50 (1999), 11-30.

[7] Axelsson, O., Lindskog, G. On the rate of convergence of the preconditioned gradient methods. Numer. Math. 48 (5), (1986) 499–523.

[8] F.F. Campos and N.R.C. Birkett. An efficient solver for multi-right-hand-side linear systems based on the CCCG(η) method with applications to implicit time-dependent partial differential equations. *SIAM J. Sci. Comput.*, 19, 126-138, 1998.

[9] E. Cuthill, J. McKee. Reducing the bandwidth of sparse symmetric matrices. *Proc. 24th Nat. Conf. Assoc. Comput. Mach.*, 157-172, 1969.

[10] I. S. Duff, R. G. Grimes and J. G. Lewis. Sparse matrix test problems. *ACM Trans. Math. Software*, 15, 1-14, 1989.

[11] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman. Yale sparse matrix package I: The symmetric codes. *Int. J. Numer. Methods Engrg.*, 18, 1145-1151, 1982.

[12] J. A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal*, 10, 345-363, 1973.

[13] J.A. George and J.W.H. Liu. *Computer solution of large sparse positive definite systems*. Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[14] J. George and J.W.H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Rev.*, 31, 1-19, 1989.

[15] J. A. George, J.W.H. Liu and E.G. Ng. User's guide for SPARSPAK: Waterloo Sparse Linear Equation Equations Package. Tech. report CS78-30 (revised), Department of Computer Sciences, University of Waterloo, Waterloo, ON, Canada, 1980.

[16] M.T. Jones and P.E. Plassman. An improved incomplete Cholesky factorization. *ACM Trans. Math. Software*, 21, 5-17, 1995.

[17] I. E. Kaporin. Explicitly preconditioned conjugate gradient method for the solution of unsymmetric linear systems. *Intern. J. Computer Math.* 40, 169-187, 1992.

[18] I. E. Kaporin. New convergence results and preconditioning strategies for the conjugate gradient method. *J. Numer. Linear Algebra Appls.* 1, 179-210, 1994.

[19] I. E. Kaporin. High quality preconditioning of a general symmetric positive definite matrix based on its $U^TU + U^TR + R^TU$-decomposition. *Numer. Linear Algebra Appls.* 5, 483-509, 1998.

[20] J.W.H. Liu, A. H. Sherman. Comparative analysis of the Cuthill-McKee and the reverse Cuthill-Mckee ordering algorithms for sparse matrices. *SIAM J. Numer. Anal.*, 13, 198-213, 1975.

[21] T. A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Math. Comput.* 34, 473-497, 1980.

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

[23] N. Munksgaard. Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients. *ACM Trans. Math. Software*, 6, 206-219, 1980.

[24] E. G. Ng and B. W. Peyton. Block sparse Cholesky algorithms on advanced uniprocessor computer. *SIAM J. Sci. Comput.*, 14, 1034-1056, 1993.

[25] Y. Notay. On the convergence rate of the conjugate gradients in presence of rounding errors. *Numer. Math.* 65, 301-317, 1993.

[26] A. Padiy, O. Axelsson and B. Polman. Generalized augmented matrix preconditioning approach and its application to iterative solution of ill-conditioned algebraic systems, *submitted to SIAM Journal of Matrix Anal. and Appl., report 9922, University of Nijmegen*, May, 1999.

[27] E. Rothberg and A. Gupta. Fast sparse matrix factorization on modern workstation - exploiting the memory hierarchy. *ACM Trans. Math. Software*, 17, 313-334, 1991.

[28] E. Rothberg and S.C. Eisenstat. Node selection strategies for bottom-up sparse matrix ordering. *SIAM J. Matr. Anal. Appl.*, 19, 682-695, 1998.

[29] A. H. Sherman. On the efficient solution of sparse systems of linear and nonlinear equations. Report No.46, Dept. of Computer Science, Yale University, 1975.

[30] W.F. Tinney and J.W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *J. Proc. IEEE*. 55, 1801-1809, 1967.

[31] M. Tismenetsky. A new preconditioning technique for solving large sparse linear systems. *Linear Algebra Appls.* 154-156. 331-353, 1991.

Part II

Iterative solution methods of Schur complement and related types and parallel implementations

6 Introduction

As we have seen in the first part, direct solution methods are quite competitive with iterative solution methods, in particular for $2D$ problems, even of quite large size. The asymptotic growth with increasing problem size, in particular for $3D$ problems is very large, however. This means that iterative methods, which have a lower asymptotic growth, will eventually be less expensive, even for the extremely ill-conditioned problems we consider in this study. Normally, we use the parameter free conjugate gradient method as iterative solution method. The choice of preconditioner is fundamental for the efficiency of this method. A natural choice of preconditioners are incomplete factorization methods, based on some drop tolerance parameter, as they can be computed to be arbitrary close to the direct solver, but can save much fill-in, and hence computer memory, factorization time and solution time. As we have seen, the IC-2 method is robust with respect to various problem parameters, such as aspect ratios and irregular meshes and compete well with direct solution methods. However, the asymptotic growth of the computational complexity of this methods with increasing problem size is typically $O(N^{1+\frac{1}{d}})$ where N is the degrees of freedom and d ($d = 2$ or $d = 3$) is the space dimension of the boundary value problem. Also the factorization cost is still fairly high and is furthermore sensitive to the choice of the drop tolerance (threshold) parameters in the methods. The numerical tests indicate a nonmonotone dependence of the computer time as a function of the decreasing values of this parameter and it must be chosen very small in order to come to the performance of a direct solver.

Therefore we consider now, in Part II of this study, preconditioners which are both robust as IC-2 but which may have a smaller asymptotic growth with problem size.

A good choice is thereby methods related to Schur complement solvers. Schur complement solvers have been used in direct solution methods to enable the reduction of several substructures to an equation for the reduced system corresponding to the interface or "glue" of the global structure. However, forming the Schur complement system can be expensive. Clearly, Schur complement based methods can be used with iterative methods also. Thereby we note that the method now works on vectors of much smaller dimension than for the global system. Consider then a matrix partitioned in two-by-two block form $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ to solve $A \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$, where we assume that the order of A_{22} is much smaller than the order of A_{11}. Since A is s.p.d., A_{11} as well as the Schur complement matrix $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$ are also s.p.d.

In an iterative method, the reduced system $Sx_2 = f_2 - A_{21}A_{11}^{-1}f_1$ can be solved without forming the matrix S, as only the action of S are needed. Hereby it can be efficient to compute the actions of A_{11}^{-1} by an iterative solver also as, normally, A_{11} is much better conditioned than the global matrix. However, the method will be too expensive unless an efficient preconditioner is used for S. An efficient such preconditioner is a coarse mesh matrix for the same problem. As it turned out, however, the total cost is expensive as the number of outer iterations multiply the number of inner iterations used for A_{11}. Therefore, the preconditioner for S must be further improved. Such an improvement can be based on writing the inverse of the Schur complement in the form

$$S^{-1} = A_{22}^{-1} + A_{22}^{-1} A_{21} S_1^{-1} A_{12} A_{22}^{-1}, \tag{6.1}$$

which follows e.g. from the Sherman-Morrison formula. Here $S_1 = A_{11} - A_{12} A_{22}^{-1} A_{21}$. In general, A_{22} is a diagonal matrix, which enables the formation of S_1 with relatively small expense. In fact, S_1 can be written as a sum of element matrices, and each of those can be efficiently preconditioned, to form an element by element preconditioner for S_1. The approximation of the inverse as a sum of terms as in (6.1) is an efficient, general approach which we will now demonstrate.

We can construct either a block approximate factorization of A or approximate the inverse of A on additive form. The approaches are related to each other as the following shows. First, the block matrix factorization of A is

$$A = \begin{bmatrix} A_{11} & 0 \\ A_{21} & S \end{bmatrix} \begin{bmatrix} I_1 & A_{11}^{-1} A_{12} \\ 0 & I_2 \end{bmatrix}$$

where I_1, I_2 denote the unit matrices of proper order. For its inverse, it holds

$$\begin{aligned} A^{-1} &= \begin{bmatrix} I_1 & -A_{11}^{-1} A_{12} \\ 0 & I_2 \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ -S^{-1} A_{21} A_{11}^{-1} & S^{-1} \end{bmatrix} \\ &= \begin{bmatrix} A_{11}^{-1} + A_{11}^{-1} A_{12} S^{-1} A_{21} A_{11}^{-1} & -A_{11}^{-1} A_{12} S^{-1} \\ -S^{-1} A_{21} A_{11}^{-1} & S^{-1} \end{bmatrix} \end{aligned}$$

or

$$A^{-1} = \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} -A_{11}^{-1} A_{12} \\ I_2 \end{bmatrix} S^{-1} \left[-A_{21} A_{11}^{-1}, I_2 \right].$$

A straightforward computation reveals that $A_{\widetilde{V}} \equiv \widetilde{V}^T A \widetilde{V} = S$ and, hence,

$$A^{-1} = \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & 0 \end{bmatrix} + \widetilde{V} (\widetilde{V}^T A \widetilde{V})^{-1} \widetilde{V}^T = \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & 0 \end{bmatrix} + \widetilde{V} A_{\widetilde{V}}^{-1} \widetilde{V}^T,$$

where

$$\widetilde{V} = \begin{bmatrix} -A_{11}^{-1} A_{12} \\ I_2 \end{bmatrix}.$$

Let $M_{11} \simeq A_{11}$ be an approximation of A_{11} (for which linear systems are simpler to solve than for A_{11}) and let $G_1 \simeq A_{11}^{-1}$ be a sparse approximate inverse. Possibly $G_1 = M_{11}^{-1}$. Then

$$M = \begin{bmatrix} M_{11} & 0 \\ A_{21} & B_2 \end{bmatrix} \begin{bmatrix} I_1 & M_{11}^{-1} A_{12} \\ 0 & I_2 \end{bmatrix}$$

is a preconditioner to A and

$$B = \begin{bmatrix} G_1 & 0 \\ 0 & 0 \end{bmatrix} + V B_2^{-1} V^T$$

is an approximate inverse, where $V = \begin{bmatrix} -G_1 A_{12} \\ I_2 \end{bmatrix}$ and B_2 is an approximation of S. If $B_2 = V^T \widetilde{A} V$, where $\widetilde{A} = \begin{bmatrix} G_1^{-1} & A_{12} \\ A_{21} & A_2 \end{bmatrix}$, then

$$
\begin{aligned}
B &= \begin{bmatrix} G_1 & 0 \\ 0 & 0 \end{bmatrix} + V(V^T \widetilde{A} V)^{-1} V^T \\
&= \begin{bmatrix} G_1 & 0 \\ 0 & 0 \end{bmatrix} + V S(\widetilde{A}) V^T,
\end{aligned}
$$

where $S(\widetilde{A}) = A_2 - A_{21} G_1 A_{12}$. If $M_1 = G_1^{-1}$, then in this case

$$B = M^{-1}.$$

Hence a convergence estimate for one method, say the block factorization method, can be directly applied for the other (say the approximate inverse method) method as well.

A related method is the augmented subspace preconditioning (ASP) method which has the form

$$A^{-1} = M^{-1} + \sigma V B_V^{-1} V^T,$$

where M is a preconditioner to $A_V = V^T A V$. Further, V is a rectangular $m \times n$ matrix, where normally $m \ll n$. The best choice of column vectors in V are vectors which span (approximately) the subspace of eigenvectors corresponding to the smallest eigenvalues of A. Finally, σ is a scaling parameter chosen to move the smallest eigenvalues to the cluster of bigger eigenvalues.

In this study, we compare methods ASP, Schur complement preconditioned solvers and presents also some parallel solution methods for the set of benchmark problems defined in Part I of this study.

When implementing the solution methods on a parallel computer platform, it is well known that methods based on factorization are not very efficient. This is due to the many recursive calls involved in the method and the purely sequential nature of the forward and backward triangular solves, which require much communication. It is also difficult to get a good load balance of the work among the available processors. Therefore, in Section 10 the RIC2S method has been adjusted for overlapping blocks to increase the performance. The ASP method can be implemented reasonably well in parallel when the matrix-vector multiplications and the actions of the matrix M^{-1} are parallelizable. However, this method involves also an inner solver which uses the RIC2S method itself which, as we have seen, is less parallelizable. On the other hand, since in the context of the ASP method, the inner (coarse mesh) preconditioner takes place on a coarse mesh it has therefore less influence on the global computer time.

Similarly, for the Schur complement solver, all matrix-vector multiplications of the outer solver can be implemented efficiently. The two possible bottlenecks for an efficient parallel implementation are the preconditioning step for the Schur complement system and the preconditioning step when solving the inner system. The best numerical performance for the defect-correction (Schur complement) methods is achieved when both above mentioned preconditioners are incomplete RIC2S factorizations. The parallel implementation of this numerical scheme is far from trivial, however, and has not been done within this study. The problem to find a good compromise between numerically efficient methods and methods with good parallel performance remains to some extend unresolved when real-life problems are tuckled. A reach source of information on solving large scale elasticity problems in parallel is [41] where various preconditioning techniques are tested within a object oriented parallel Finite Element package.

The remainder of this paper is organized as follows. Section 7 describes the Augmented Subspace Iteration (ASP) method, which is a general and powerful technique to approach problems where the ill-posedness is due to extremely small eigenvalues of the matrix of the discrete system of algebraic equations. In Section 8 a Schur complement based inner-outer iterative solution scheme is discussed and numerically tested. There, a special domain decomposition type ordering strategy of the mesh nodes is introduced which leads to a structure of the corresponding stiffness matrix which is relatively easily handled for parallel implementations. Section 9 discusses a special version of the ASP method suitable for problems with many right hand sides. Sections 10 and 11 describe two parallel solution methods. The first one is a block-diagonal overlapping Jacobi preconditioned CG, where the blocks are solved by the RIC2S method and the amount and the shape of the overlap is determined in order to minimize the so-called K-condition number of the preconditioned matrix. The second one is nonoverlapping Block-Jacobi preconditioned CG method which utilizes the block structure of the matrix introduced by the domain decomposition orderings described in Section 8. In Section 12 alternative solution techniques are listed. The so-called IBBLU method is mentioned in particular detail, as a general technique applicable also for nonsymmetric systems. Conclusions are given in Section 13.

7 The augmented subspace approximate inverse preconditioning

7.1 Description of the ASP method

In severely ill-conditioned problems the ill-conditioning is often caused by a relatively small group of extremely small eigenvalues. One way to tackle the slow convergence of iterative methods resulting form these small eigenvalues is to deflate (see [42]) the corresponding eigenmodes by using a projector

$$B = I - V(V^tAV)^{-1}V^tA$$

as a right preconditioner where the span of the columns of V should include the “bad” eigenmodes. The disadvantage of this approach is that one has to solve the system with $A_V := V^tAV$ exactly in each iteration. To avoid this problem we use an idea suggested in [10], instead of deflating the small eigenmodes we try to shift them to the vicinity of the largest eigenvalues by using a preconditioner of the form

$$B = I + \sigma V {B_V}^{-1} V^t$$

where B_V is an easily invertible approximation of A_V. In this form we do not have to solve exactly systems with A_V which allows us a wider range of choices for V. A further advantage is that the method can be easily combined with a second preconditioner M which may improve the whole spectrum but in particular bounds the largest eigenvalues:

$$B = M^{-1} + \sigma V {B_V}^{-1} V^t$$

For a detailed analysis of this method see [49]. The main result in the current context is the following

THEOREM 7.1 *Let* $B = M^{-1} + \sigma V {B_V}^{-1} V^t$, $\sigma = \lambda_{\max}(M^{-1}A)/\lambda_{\max}({B_V}^{-1}A_V)$. *Assume that* $\{(\lambda_i, \mathbf{v}_i)\}_{i=1}^n$ *is an ordered set of eigenpairs of* $M^{-1}A$ *such that* $\lambda_1 \leq \cdots \leq \lambda_n$. *Let* $V_e = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$. *If* V *is such that the subspaces* $\mathcal{W} = (\mathrm{Im}\, A^{\frac{1}{2}} V)^{\perp}$ *and* $\mathcal{V}_e = \mathrm{Im}\, A^{\frac{1}{2}} V_e$ *satisfy the condition*

$$\cos(\mathcal{W}, \mathcal{V}_e) = \sup_{\substack{\mathbf{x} \in \mathcal{W} \\ \mathbf{y} \in \mathcal{V}_e}} \frac{\mathbf{x}^T \mathbf{y}}{(\mathbf{x}^T \mathbf{x})^{\frac{1}{2}} (\mathbf{y}^T \mathbf{y})^{\frac{1}{2}}} \leq \gamma < 1$$

then the minimal eigenvalue of BA *is bounded as*

$$\lambda_{\min}(BA) \geq \max\left\{\lambda_1, (1-\gamma) \cdot \min\left\{\frac{\lambda_{\max}(M^{-1}A)}{\kappa(B_V^{-1}A_V)}, \lambda_{m+1}\right\}\right\}. \tag{7.1}$$

The maximal eigenvalue of BA *is bounded as*

$$\lambda_{\max}(BA) \leq 2\lambda_{\max}(M^{-1}A) \tag{7.2}$$

for any choice of V *and* B_V.

For a proof see [49].

In the present context we choose for V^t a restriction operator so that the method can be viewed as an approximate subspace correction method. Let A_H be the matrix corresponding to the initial coarse mesh and let A_h be the matrix corresponding to the final fine mesh which is constructed by uniformly refining the coarse mesh one or more times. For V^t we then choose the standard trilinear restriction operator from the fine to the coarse mesh. As an approximation to $A_V = V^t A V$ we choose B_V as the RIC2S incomplete factorization, described in Section 4, of A_H. Depending on the boundary conditions and whether the mesh is axiparallel A_H will be equal to A_V or otherwise a close approximation. In the numerical experiments we have always scaled the matrix A_h to unit diagonal. For the fine mesh preconditioner M we either use I or again a RIC2S incomplete factorization but now of A_h and typically we use a larger value of the threshold parameter τ on the fine mesh than on the coarse.

The parameter σ can be easily estimated, for $\lambda_{\max}(M^{-1}A)$ we can simply use 2 (note that A_h is diagonally scaled) and $\lambda_{\max}({B_V}^{-1}A_V)$ is approximated by a few Lanczos iterations.

The computational cost of one application of the preconditioner is $2^d N$ for the prolongation and for the restriction where d is the space dimension, in our case $d = 3$, plus one solve with the RIC2S factors of the coarse mesh matrix and possibly a solve with the RIC2S factors of the fine mesh matrix. For 3D problems typically in case of $M = I$ one PCG iteration is about 25% more expensive than an unpreconditioned CG iteration and if $M = RIC2S$ it only is about 50 to 60% more expensive.

7.2 Numerical Experiments

In this section we present the numerical tests for the 3D benchmark problems.

These numerical tests are performed on a different computer, SUN Ultra Enterprise 10000 with 24 UltraSparc 330 MHz processors and 12 GB of memory, and the timing results are not directly comparable with the other numerical experiments. Although the processors of SUN Enterprise 10000 are twice faster than those of the SUN Enterprise 3000 computer, about a factor two is gained only in the construction of the preconditioner (t_{prep}). The solution time (t_{it}), however, turns out to be even slightly larger due to slow memory and cache mishits.

In all tests we have used a standard Preconditioned Conjugate Gradient method with starting vector $x_0 = 0$ and stopping criterion $\|r_k\| \leq 10^{-8}\|r_0\|$. As preconditioner we have used the Augmented Subspace approximate inverse Preconditioner of the form

$$B = M^{-1} + \sigma V^t {B_V}^{-1} V.$$

For B_V we always use the RIC2S factorization of A_H with $\tau_1 = 0.01$, $\tau_2 = 0.0001$, see section 4. For M we use I or the RIC2S factorization of A_h with $\tau_1 = 0.03$, $\tau_2 = 0.0009$. This leads to a rather rough approximation of A_h but it is anticipated that this will be more than compensated for by the subspace approximation. In the first table we present the results for all 3-dimensional test problems in the case that $H = 2h$, i.e., the coarse mesh is once uniformly refined to get the fine mesh. In the second table we have used $H = 4h$, so that the dimension of the fine mesh matrix is 64 times larger than the dimension of the coarse matrix.

For σ we use the computed estimate as described in the previous section if $M = RIC2S$, in the case that $M = I$ we use twice the computed estimate. This is based on the expectation that $\lambda_{\max}(A_h) > \lambda_{\max}(M^{-1}A_h)$, which is confirmed by the numerical experiments. We have included tables 13 and 14 to show that the dependence on σ is not severe and tables 15 and 16 to show how the method behaves if we increase the gap between coarse and fine mesh.

In the tables flops stands for flops per iteration per unknown. The NoPrec results correspond to the unpreconditioned CG and are included for comparison only.

7.3 Conclusions

The results show that the augmented subspace approach via a coarse mesh is so efficient in reducing the condition number that it suffices to use a simple scaling of A to reduce the largest eigenvalues, i.e. we can take $M = I$. With a few exceptions, the computing time for this version was always less than for the RIC2S version. This holds even when we use a fairly coarse mesh, determined by the ratio $H/h = 4$. It is further observed that this combination can be implemented efficiently on various parallel platforms. A further observation is that the flops per unknown didn't vary much between the different problems but the number of iterations did vary significantly. It may be anticipated that some block diagonal preconditioning matrix M can further improve the current method.

8 Schur complement based solvers

Let A be partitioned in two by two block form $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ with blocks of orders n_1, n_2. We consider some properties of the Schur complement system S for A,

$$S = A_{22} - A_{21}A_{11}^{-1}A_{12}$$

M	size	t_{prep}	t_{it}	t_{perit}	t_{tot}	iter.	flops	true res.
Bricks								
NoPrec	135	0	0.0039	0.00018	0.0039	22	38.1	1.85e-09
I	135	0.0026	0.0072	0.00023	0.0098	31	58.6	1.95e-08
RIC2S	135	0.011	0.0062	0.00039	0.017	16	81.6	3.56e-08
NoPrec	675	0	0.052	0.0012	0.052	45	54.8	5.92e-09
I	675	0.011	0.060	0.0015	0.071	39	77.6	2.60e-08
RIC2S	675	0.13	0.057	0.0024	0.19	24	104	3.03e-08
NoPrec	4131	0	0.78	0.0088	0.78	89	67.8	6.84e-09
I	4131	0.28	0.55	0.011	0.84	50	92.1	3.94e-08
RIC2S	4131	1.64	0.79	0.021	2.43	38	120	1.56e-08
NoPrec	28611	0	33.072	0.18072	33.1	183	76	8.95e-09
I	28611	7.01	16.7	0.23	23.7	73	101	2.82e-08
RIC2S	28611	22.5	19.6	0.32	42.1	61	129	2.29e-08
NoPrec	212355	0	644.4	1.76	644.4	366	80.8	9.96e-09
I	212355	103.7	286.4	2.23	390.1	128	106	2.48e-08
RIC2S	212355	248.9	388.0	3.27	637.0	119	135	1.71e-08
Soil								
NoPrec	2187	0	0.58	0.0040	0.58	143	61.8	9.62e-09
I	2187	0.083	0.57	0.0052	0.65	109	86.4	2.20e-08
RIC2S	2187	0.79	0.40	0.0086	1.18	46	114	1.12e-08
NoPrec	14739	0	20.9	0.072	20.9	290	72.5	9.02e-09
I	14739	3.13	14.5	0.098	17.6	147	97.4	3.54e-08
RIC2S	14739	10.9	11.1	0.16	22.0	70	127	2.39e-08
NoPrec	107811	0	463.4	0.79	463.4	588	78.9	9.59e-09
I	107811	49.9	203.6	0.99315	253.5	205	104	3.07e-08
RIC2S	107811	127.9	181.2	1.4978	309.2	121	135	1.86e-08
NoPrec	823875	0	9230.3	7.7	9230.3	1189	82.3	9.70e-09
I	823875	608.7	3080.9	9.8	3689.5	313	108	2.97e-08
RIC2S	823875	1311.6	3643.1	15.05	4954.7	242	139	1.96e-08
Detail								
I	79065	11.8	972.9	0.57	984.6	1713	103	6.09e-08
RIC2S	79065	54.9	916.5	0.88	971.4	1036	132	5.27e-08
I	584127	576.3	18511	8.0	19087	2312	107	9.69e-08
RIC2S	584127	1565.8	21553	13.4	23119	1605	136	6.81e-08
Sluis								
I	12957	0.98	24.5	0.068	25.46	362	89.6	4.70e-08
RIC2S	12957	5.09	18.532	0.10901	23.6	170	115	4.84e-08
I	87159	14.6	495.2	0.72	509.8	689	98	6.29e-08
RIC2S	87159	53.3	354.6	1.06	407.8	333	125	4.56e-08
I	633579	188.8	8364.4	7.0	8553.3	1194	103	5.37e-08
RIC2S	633579	529.9	7419.6	10.12	7949.5	733	131	4.28e-08

Table 11: SUN Enterprise 10000: $H = 2h$

M	size	t_{prep}	t_{it}	t_{perit}	t_{tot}	iter.	flops/iter	true res.
Bricks								
NoPrec	675	0	0.052	0.0012	0.052	45	54.8	5.92e-09
I	675	0.0028	0.074	0.0014	0.077	52	71.7	2.87e-08
RIC2S	675	0.12	0.063	0.0022	0.19	28	98	2.42e-08
NoPrec	4131	0	0.78	0.0088	0.78	89	67.8	6.84e-09
I	4131	0.013	0.73	0.010	0.75	71	84.9	3.52e-08
RIC2S	4131	1.33	0.71	0.016	2.04	44	112	2.17e-08
NoPrec	28611	0	33.07	0.18	33.07	183	76	8.95e-09
I	28611	0.29	20.1	0.21	20.39	98	93.3	3.67e-08
RIC2S	28611	15.6	20.3	0.30	35.8	68	122	1.89e-08
NoPrec	212355	0	644.4	1.76	644.4	366	80.8	9.96e-09
I	212355	7.17	266.6	2.0	273.8	133	98	3.56e-08
RIC2S	212355	149.3	308.7	3.0	458.0	103	127	2.59e-08
Soil								
NoPrec	14739	0	20.9	0.072	20.88	290	72.5	9.02e-09
I	14739	0.090	17.68	0.084	17.77	210	89.8	1.97e-08
RIC2S	14739	7.80	12.84	0.15	20.63	88	120	1.61e-08
NoPrec	107811	0	463.4	0.79	463.43	588	78.9	9.59e-09
I	107811	2.82	238.1	0.90	240.9	265	96.1	3.88e-08
RIC2S	107811	81.14	177.11	1.42	258.25	125	127	2.65e-08
NoPrec	823875	0	9230.3	7.76	9230.3	1189	82.3	9.70e-09
I	823875	51.20	3187.2	8.78	3238.5	363	99.5	3.14e-08
RIC2S	823875	743.3	2706.9	13.7	3450.2	198	131	2.04e-08
Detail								
I	584127	11.97	19433	5.71	19445	3406	98.8	8.54e-08
RIC2S	584127	421.8	15740	8.40	16162	1874	128	5.69e-08
Sluis								
I	87159	0.997	543.4	0.67	544.4	811	91.3	5.79e-08
RIC2S	87159	38.29	354.2	1.01	392.5	350	118	4.05e-08
I	633579	14.89	9122.5	6.35	9137.4	1436	96.7	7.27e-08
RIC2S	633579	356.58	6071.1	9.46	6427.6	642	124	5.63e-08

Table 12: SUN Enterprise 10000: $H = 4h$

Problem: ***Soil*** $H = 2h$								
Outer	ASP	t_{prep}	t_{it}	t_{perit}	t_{tot}	iter.	flops	true res.
I	-	0	9230.3	7.76	9230.3	1189	82.3	9.70e-09
I	4σ	592.31	3278.2	9.76	3870.5	336	108	3.70e-08
I	2σ	608.68	3080.9	9.84	3689.5	313	108	2.97e-08
I	σ	595.43	3339.2	9.79	3934.6	341	108	2.39e-08
I	$\sigma/2$	594.2	4120.1	9.79	4714.3	421	108	1.91e-08
RIC2S	4σ	1299.5	3714.6	14.62	5014.1	254	139	2.87e-08
RIC2S	2σ	1296.9	3526.8	14.76	4823.7	239	139	2.49e-08
RIC2S	σ	1311.6	3643.1	15.05	4954.7	242	139	1.96e-08
RIC2S	$\sigma/2$	1307.9	3926.3	14.93	5234.2	263	139	1.75e-08

Table 13: SUN Enterprise 10000: *soil*, size=823875

Problem: ***Soil*** $H = 4h$								
Outer	ASP	t_{prep}	t_{it}	t_{perit}	t_{tot}	iter.	flops	true res.
I	-	0	9230.3	7.76	9230.3	1189	82.3	9.70e-09
I	4σ	50.05	3658.5	8.69	3708.5	421	99.5	4.12e-08
I	2σ	51.21	3187.2	8.78	3238.5	363	99.5	3.14e-08
I	σ	52.68	3138.9	8.79	3191.6	357	99.5	2.73e-08
I	$\sigma/2$	50.00	3494.2	8.80	3544.2	397	99.5	2.15e-08
RIC2S	4σ	742.08	3596.8	13.62	4338.9	264	131	2.99e-08
RIC2S	2σ	745.35	2990.5	13.59	3735.8	220	131	2.75e-08
RIC2S	σ	743.32	2706.9	13.67	3450.2	198	131	2.04e-08
RIC2S	$\sigma/2$	743.23	2757.7	13.65	3501.0	202	131	1.72e-08

Table 14: SUN Enterprise 10000: *Soil*, size=823875

Soil, size=823875								
M	H/h	t_{prep}	t_{it}	t_{perit}	t_{tot}	iter.	flops/iter	true res.
NoPrec		0	9230.3	7.76	9230.3	1189	82.3	9.70e-09
I	16	0.24	6463.8	8.76	6464	738	98.4	1.92e-08
RIC2S	16	692.97	4146.1	13.78	4839.1	301	130	1.49e-08
I	8	2.95	4283.3	8.71	4286.3	492	98.5	4.06e-08
RIC2S	8	696.01	3093.1	13.57	3789.1	228	130	2.16e-08
I	4	51.21	3187.2	8.78	3238.5	363	99.5	3.14e-08
RIC2S	4	743.3	2706.9	13.67	3450.2	198	131	2.04e-08
I	2	608.68	3080.9	9.84	3689.5	313	108	2.97e-08
RIC2S	2	1311.6	3643.1	15.05	4954.7	242	139	1.96e-08

Table 15: SUN Enterprise 10000, size=823875

Bricks, size=212355								
M	H/h	t_{prep}	t_{it}	t_{perit}	t_{tot}	iter.	flops/iter	true res.
NoPrec		0	644.4	1.76	644.4	366	80.8	9.96e-09
I	16	0.052	461.29	2.01	461.34	230	96.8	4.23e-08
RIC2S	16	145.66	438.41	3.04	584.08	144	126	2.50e-08
I	8	0.33	357.53	2.01	357.85	178	97	4.54e-08
RIC2S	8	145.39	364.38	3.04	509.77	120	126	2.20e-08
I	4	7.17	266.63	2.00	273.8	133	98	3.56e-08
RIC2S	4	149.34	308.68	3.00	458.02	103	127	2.59e-08
I	2	103.66	286.42	2.24	390.08	128	106	2.48e-08
RIC2S	2	248.95	388.03	3.26	636.98	119	135	1.71e-08

Table 16: SUN Enterprise 10000

of relevance when solving the corresponding reduced system

$$S\mathbf{x}_2 = \mathbf{f}_2 - A_{21}A_{11}^{-1}\mathbf{f}_1$$

where $A\begin{bmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{bmatrix} = \begin{bmatrix}\mathbf{f}_1\\ \mathbf{f}_2\end{bmatrix}$.
The next Theorem is well known (see e.g. [3]).

THEOREM 8.1 *Let A be Hermitian positive definite, A_{11} be nonsingular and $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ be a partitioning of x consistent with the partitioning of A. Then*

(a) $\mathbf{x}^* A\mathbf{x} \geq \mathbf{x}_2^* S\mathbf{x}_2$.

(b) For any $\mathbf{x}_2$, $\min_{\mathbf{x}_1} \mathbf{x}^* A\mathbf{x} = \mathbf{x}_2^* S\mathbf{x}_2$.

(c) $\min_{\mathbf{x}_1,\mathbf{x}_2\neq 0} \mathbf{x}^* A\mathbf{x} = \min_{\mathbf{x}_2\neq 0} \mathbf{x}_2^* S\mathbf{x}_2$.

Proof. Clearly A_{11}, A_{22} are positive definite. The statements follow directly from the Cholesky decomposition,

$$A = \begin{bmatrix} 0 & 0 \\ 0 & S \end{bmatrix} + Q^*Q, \text{ where } Q = \left[A_{11}^{\frac{1}{2}}, A_{11}^{-\frac{1}{2}}A_{12}\right] \qquad \square$$

Part (b) shows that S is positive definite. Consider now the condition number of S when A is a finite element matrix partitioned according to the set of (2) vertex nodes forming a coarse mesh (Ω_H) and the set of edge and interior nodes, respectively. Consider now the condition number of S. There exists a transformation matrix $J = \begin{bmatrix} I_1 & J_{12} \\ 0 & I_2 \end{bmatrix}$, where J_{12} is an interpolation matrix from some coarse to the added fine node points for which it holds

$$\widehat{A} = J^T AJ$$

where $\widehat{A} = \begin{bmatrix} \widehat{A}_{11} & \widehat{A}_{12} \\ \widehat{A}_{21} & A_H \end{bmatrix}$ is the corresponding hierarchical basis function matrix. Note that $\widehat{A}_{11} = A_{11}$ and $A_H = [J_{12}^T, I_2]A \begin{bmatrix} J_{12} \\ I_2 \end{bmatrix}$. The corresponding Schur complement matrix for $\widehat{A}$ is

$$\widehat{S} = A_H - \widehat{A}_{12}A_{11}^{-1}\widehat{A}_{12}$$

and an elementary computation (see e.g. [53] and [12]) shows that $S = \widehat{S}$.

Now it is known (see [3]) that A_H is spectrally equivalent to $\widehat{S}$, i.e. also to S, and there holds

$$\mathbf{x}_2^T A_H \mathbf{x}_2 \geq \mathbf{x}_2^T \widehat{S} \mathbf{x}_2 \geq (1-\gamma^2)\mathbf{x}_2^T A_H \mathbf{x}_2, \text{for all } \mathbf{x}_2 \in \mathbb{R}^{n_2}. \tag{8.1}$$

Here

$$\gamma = \sup_{\mathbf{x}_1,\mathbf{x}_2} \frac{\mathbf{x}_1 \widehat{A}_{12}\mathbf{x}_2}{\{\mathbf{x}_1^T A_{11}\mathbf{x}_1\}^{\frac{1}{2}}\{\mathbf{x}_2^T A_H \mathbf{x}_2\}^{\frac{1}{2}}},$$

the constant in the strengthened CBS inequality. It follows from (8.1) that

$$\text{cond}(S) = \text{cond}(\widehat{S}) \leq \frac{1}{1-\gamma^2}\text{cond}(A_H).$$

Here, for 2nd order boundary value problems, it holds that

$$\text{cond}(A_H) \simeq \left(\frac{h}{H}\right)^2 \text{cond}(A),$$

so the Schur complement matrix has a smaller condition number than A,

$$\text{cond}(S) < \text{cond}(A)$$

if $\gamma^2 \lesssim 1 - \left(\frac{h}{H}\right)^2$.
The following estimates hold for $\frac{1}{1-\gamma^2}$ (see e.g. [54]),

$$\frac{1}{1-\gamma^2} = \begin{cases} O\left(\log\left(1+\frac{H}{h}\right)\right), & \text{in } 2D \\ O\left(\frac{H}{h}\right), & \text{in } 3D \end{cases} \tag{8.2}$$

Hence, asymptotically as $\frac{h}{H} \to 0$, the condition number $\text{cond}(S) \ll \text{cond}(A)$.

8.1 Description of the method

Estimate (8.2) indicates that if we have at our disposal two (nested) meshes, $\mathcal{T}_H$ and $\mathcal{T}_h$, with corresponding discretization matrices A_H and $A_h(= A)$, where

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{array}{l} \}\mathcal{T}_h/\mathcal{T}_H \\ \}\mathcal{T}_H, \end{array} \tag{8.3}$$

then solving the Schur complement system $S = S_H$ with respect to nodes in $\mathcal{T}_H$, preconditioned by A_H is a promising solution strategy which guarantees a fast rate of convergence.

The method described below is based on the assumption that some coarse mesh $\mathcal{T}_H$ is given, this mesh is then uniformly refined a few times to obtain a fine mesh $\mathcal{T}_h$ and the aim is to solve the arising discrete algebraic system on the fine mesh, $A\mathbf{x}_h = \mathbf{f}_h$.

Let us for theoretical purposes write the following elementary computational procedure.

The solution of $A\mathbf{x} = \mathbf{f}$, which can be rewritten in the following block factorized manner,

$$A\mathbf{x} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} A_{11} & 0 \\ A_{21} & S \end{bmatrix} \begin{bmatrix} I_1 & A_{11}^{-1}A_{12} \\ 0 & I_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix},$$

and assuming that we have an efficient procedure to invert A_{11}, is replaced by

$$\begin{array}{lll} \text{(1) Solve} & S\mathbf{x}_2 & = \mathbf{f}_2 - A_{21}A_{11}^{-1}\mathbf{f}_1 \\ \text{(2) Recover} & \mathbf{x}_1 & = A_{11}^{-1}(\mathbf{f}_1 - A_{12}\mathbf{x}_2). \end{array} \tag{8.4}$$

Step (1) in (8.4) is done using some iterative procedure. Here the Schur complement matrix S is never formed exactly, thus, its action on a vector involves one solution with A_{11}. This solution procedure, as presented in (8.4), requires $it_{Schur} + 2$ solutions with A_{11}, where it_{Schur} is the number of iterations to complete step (1) (8.4). Since step (1) is of inner-outer solution type, which corresponds to having a variable system matrix S, a generalized preconditioned CG-type of method has to be used.

The computationally most heavy part in (8.4) is to solve systems with A_{11}, which is assumed to be very large. Systems with A_{11} can be solved in various ways and the methods used in this study will be discussed in the sequel.

REMARK 8.1 One disadvantage of approach (8.4) is that it does not work directly on the residual of the whole system but only on the residual of the reduced Schur complement system. Furthermore, due to the fact that S is not formed exactly, only the iterative residual is subject to control during the iteration. Thus, as also some numerical experiments show, for some problems the iterative residual might diverge from the true one and this can be detected a posteriori when recovering the true residual after the iterative stopping criterion has been met.

In both steps (1) and (2) in (8.4) it is presumed that systems with A_{11} are solved accurately.

Based on the observation that it can be a waste of computing effort to solve the inner systems with A_{11} exactly, one can use the following defect-correction based approach.

We consider $A\mathbf{x} = \mathbf{f}$ with a matrix A, partitioned as in (8.3), namely $A_{11}\mathbf{x}_1 + A_{12}\mathbf{x}_2 = \mathbf{f}_1$, $A_{21}\mathbf{x}_1 + A_{22}\mathbf{x}_2 = \mathbf{f}_2$, and the *defects* (residuals), corresponding to some current approximate solution $\mathbf{x}^{(k)} = [\mathbf{x}_1^{(k)}, \mathbf{x}_2^{(k)}]^T$,

$$\mathbf{d}_1^{(k)} = \mathbf{f}_1 - A_{11}\mathbf{x}_1^{(k)} - A_{12}\mathbf{x}_2^{(k)}, \tag{8.5}$$

$$\mathbf{d}_2^{(k)} = \mathbf{f}_2 - A_{21}\mathbf{x}_1^{(k)} - A_{22}\mathbf{x}_2^{(k)}, \tag{8.6}$$

and the Schur complement matrix S, computed with respect to A_{11}. Then, the following algorithm is suggested.

ALGORITHM 8.1 [Defect-correction scheme] Let an initial approximation $\mathbf{x}^{(0)}$ be given.

For $k = 0, 1, 2, \cdots$ until convergence,

(dc1) *Solve accurately by iteration* $A_{11}\mathbf{x}_1^{(k)} = \mathbf{f}_1 - A_{12}\mathbf{x}_2^{(k)}$

The stopping criterion is $\|rhs_{inner}\| < \varepsilon_2$

(dc2) *Compute* $\mathbf{d}_2^{(k)} = \mathbf{f}_2 - A_{22}\mathbf{x}_2^{(k)} - A_{21}\mathbf{x}_1^{(k)}$

(dc3) *Check whether* $\|\mathbf{d}_2^{(k)}\| < \varepsilon_1\|\mathbf{f}\|$

If the criterion is satisfied, stop; otherwise continue.

(dc4) *Solve* $S\delta\mathbf{x}_2^{(k)} = \mathbf{d}_2^{(k)}$

Above is solved iteratively with inexact actions of S due to solving inner systems with A_{11} less accurately than in (dc1). The stopping criterion when solving inner systems with A_{11}, which are part of the action of S is

$\sqrt{(\widetilde{\mathbf{r}}_{inner}^{(k)}, \widetilde{\mathbf{r}}_{inner}^{(k)})} < \varepsilon_3\|rhs_{inner}\|$

(dc5) *Update* $\mathbf{x}_2^{(k+1)} = \mathbf{x}_2^{(k)} + \delta\mathbf{x}_2^{(k)}$

(dc6) *Repeat (dc1).*

The background considerations for the above described defect-correction procedure are the following. As the matrix block A_{11} is assumed to be (very) large, we must reduce the costs to solve systems with it, for instance by regulating the accuracy by which it is solved to the size of the current residual. In other words, we will not solve the systems with A_{11} to a high relative accuracy. However, at some point in the solution process we must compute an accurate action of A_{11}^{-1} (at least once), otherwise the iterative solution will converge to the solution of a perturbed system. The proposed defect-correction algorithm serves the latter purpose, namely, the correction equation for $\delta\mathbf{x}_2^{(k)}$ in step (*dc4*) is solved by a (generalized) conjugate gradient method using inexact actions of S, i.e., solving systems with A_{11} to a lower accuracy, but the defect $\mathbf{d}_2^{(k)}$ in step (*dc2*) is based on an accurately computed $\mathbf{x}_1^{(k)}$.

Suppose the system in (*dc1*) is solved exactly, then the intermediate defect $\widetilde{\mathbf{d}}_2^{(k)} = \mathbf{f}_2 - A_{21}\mathbf{x}_1^{(k)} - A_{11}\mathbf{x}_2^{(k+1)}$ would be zero. If, in addition, the system in (*dc4*) is solved exactly, then

$$S\mathbf{x}_2^{(k+1)} + A_{21}A_{11}^{-1}A_{12}\mathbf{x}_1^{(k)} = \mathbf{f}_2 - A_{21}\mathbf{x}_1^{(k+1)}$$

or

$$S\mathbf{x}_2^{(k+1)} = \mathbf{f}_2 - A_{21}A_{11}^{-1}\mathbf{f}_1,$$

i.e., we would have solved the problem exactly. However, numerical tests have shown that solving systems with A_{11} to a quite low accuracy leads to too many defect-correction steps for more difficult problems, which means no actual gain in the overall computational complexity.

The above computational scheme requires in total $no(dc) + 1$ accurate solutions with the matrix A_{11}, $no(dc) * av(Schur)$ inaccurate solutions with A_{11} and $no(dc) * av(Schur)$ solutions with the coarse mesh matrix, used as a preconditioner for the Schur complement. Here, $no(dc)$ is the number of defect-correction steps performed and $av(Schur)$ is the average number of Schur iterations required per defect correction.

8.2 Solving systems with A_H and A_{11}

Numerical experiments have shown that in the case of very ill-conditioned problems, as the test problems considered in this study, one needs a very robust and efficient solver for the coarse mesh matrix A_H. In all the numerical results systems with A_H are solved by a RIC2S-preconditioned CG method (cf. [31]) to a relatively high accuracy. We note, that since A_H is smaller than A_h, one can permit a smaller factorization drop tolerance parameter τ and use a high quality factorization without increasing excessively the total computational time. The stopping criterion for the Schur preconditioning step is indicated in Table 22.

As to the inner solution method when solving systems with A_{11}, various methods have been tested.

(i) The best performing inner method turns out to be another RIC2S-preconditioned CG method. Since the matrix A_{11} is much better conditioned than the whole matrix A, it suffices to use not very accurate factorization of it as a preconditioner (larger τ).

(ii) The inner system can be solved by a block-diagonal preconditioned CG, where the diagonal blocks correspond to the *edge* and *face* blocks in 2D and to the *edge*, *face* and *body* blocks introduced by the special domain decomposition ordering, described in Section 8.3. This preconditioner is much easier to parallelize. compared to any kind of incomplete factorization preconditioner. Its numerical efficiency, however, is not satisfactory already for the relatively better conditioned 2D test problem 'Dam', as is seen from the results presented in Table 20.

(iii) Various versions of block-incomplete factorizations have been tested, described in [8] (cf. [44], [45], [46]). However, the construction of such type of preconditioners requires M-matrices and is not directly applicable to the elasticity matrices we are dealing with. Thus, first some spectrally equivalent transformations have to be done to the A_{11} block and then construct an incomplete factorization based on that one. Practice has shown that this approach does not lead to numerically more efficient preconditioners, compared to the other choices.

(iv) The numerically most efficient preconditioner, tested for the 2D problems is to solve the inner system by another Schur complement based solution procedure, utilizing the block structure resulting from the special DD orderings, described in Section 8.3. In this case there is the extra problem how to precondition the *second level* Schur complement. Furthermore, although this method reveals the best convergence rate, it is computationally very heavy and leads to the largest computing times.

For the serial tests presented in this paper the variant (i) was used.

8.3 A domain decomposition oriented matrix partitioning

To fix some notations, let V_H and V_h be the finite element spaces, corresponding to $\mathcal{T}_H$ and $\mathcal{T}_h$, where $V_H \subset V_h \subset H_0^1$. There is a nodal set, associated with each of V_H and V_h, depending on the choice of finite element basis functions. To keep the presentation simple, in what follows it is assumed that V_H and V_h consist of continuous piecewise linear polynomials.

Let denote by $\mathcal{A}(.,.)$ the corresponding system of variational equations for the linear elasticity problem. The stiffness matrix K, corresponding to the finest refinement level, is computed from $\mathcal{A}(.,.)$ and the standard nodal basis of V_h., i.e., $K = [K_{ij}]_{i,j=1}^n$, $K_{ij} = \mathcal{A}(\varphi_i, \varphi_j)$ and $\{\varphi_j\}_{j=1}^n$ are the standard nodal basis functions. The following simple idea how to order the nodes during the refinement procedure enables us to achieve "for free" a structure of the stiffness matrix which is prone to be handled easily in parallel environment.

We describe the idea in 2D. Let refer the initial (given) triangulation $\mathcal{T}_H$ of Ω to as *the very coarse*. Let it consist of a set of nodes N_{H} (n_0 of them) and a set of edges E_{H}. Let refine this triangulation once and add the new points to the set N_{H}, i.e., let the coarse mesh points be those corresponding to $\mathcal{T}_{\frac{H}{2}}$. Figure 8 illustrates an initial mesh (bold lines), which has then been uniformly refined once. Points 1 to 6 are regarded as "coarse".

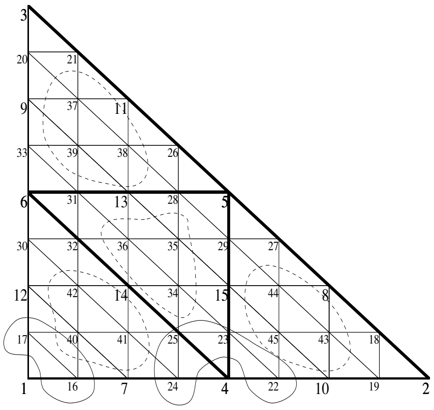
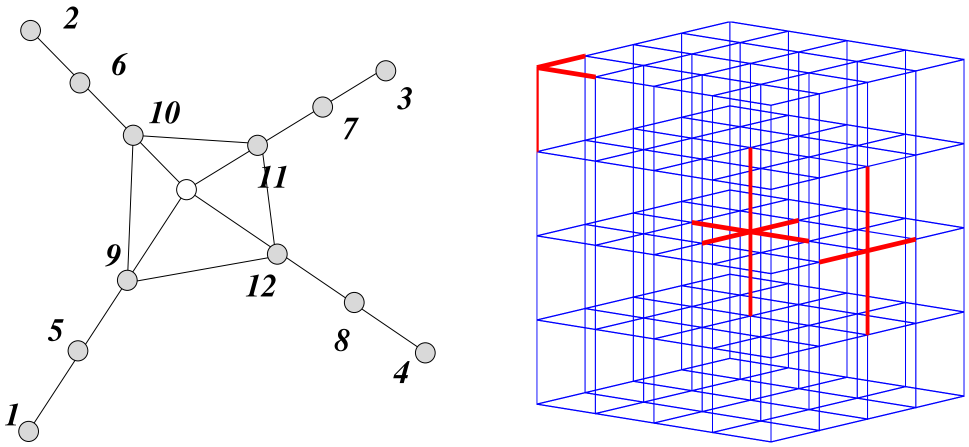


Figure 8: The node numbering strategy

We note that the points in the middle of each edge $e \in E_{\mathrm{H}}$ subdivide the coarse edges into n_0 groups around the points from N_{H}. Then, we perform another $\ell - 1$ refinement steps. During the refinement procedure all new points which appear on the coarse edges are numbered consecutively within the group, the corresponding coarse edge belongs to (illustrated in Figure 8). Needless to mention, the interior points within each coarse level subdomain are numbered also consecutively. This procedure can be seen as splitting the interface nodes into two classes (coarse/vertex nodes and edge nodes), and leads to a fine mesh stiffness matrix of the form

$$K = \begin{bmatrix} K_{ii} & K_{ie} & K_{ic} \\ K_{ei} & K_{ee} & K_{ec} \\ K_{ci} & K_{ce} & K_{cc} \end{bmatrix} \begin{array}{l} \text{i}nterior\ (face)\ nodes \\ \text{e}dge\ nodes \\ \text{c}oarse\ (vertex)\ nodes \end{array} \tag{8.7}$$

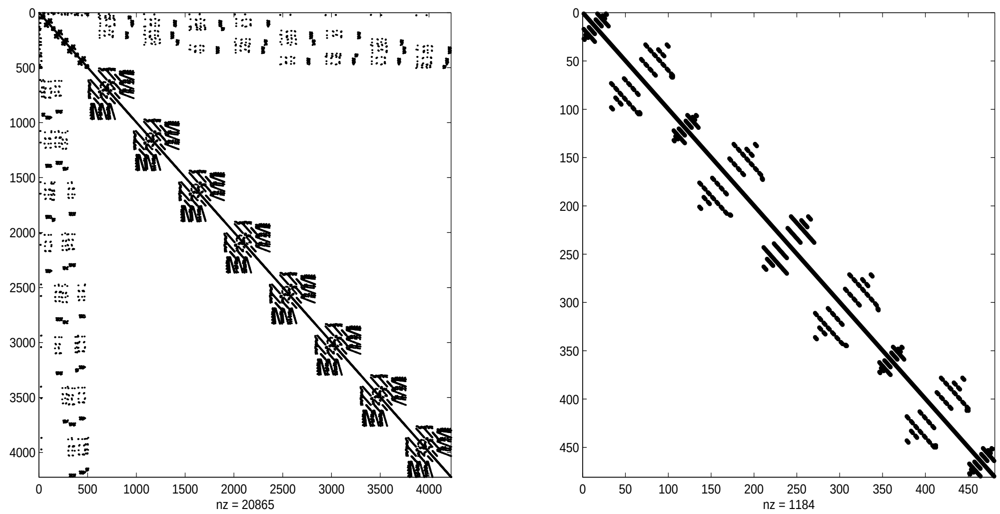
where the block K_{cc} is diagonal and, furthermore, both diagonal blocks K_{ee} and K_{ii} are also block-diagonal. The stiffness matrix structure in 2D is illustrated in Figures 10(a) and 10(b).



(a) A wire basket in 2D

(b) Wire baskets on the 3D unit cube

Figure 9:

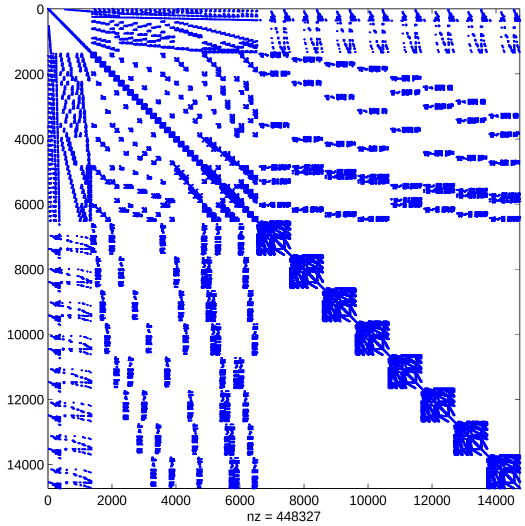


(a) The nonzero structure of the complete finest level stiffness matrix

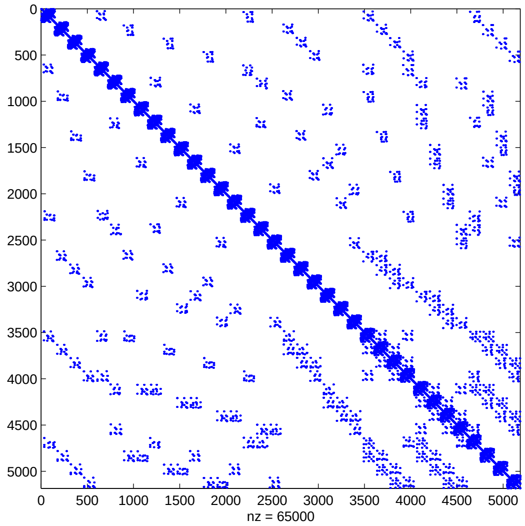
(b) The nonzero structure of the ”edge” matrix $\widetilde{K}_{ee}$

Figure 10: Example of the structure of a 2D stiffness matrix: very few elements in the initial coarse triangulation (8) and 3 levels of refinement.

We point out one advantage of this ordering strategy, namely, it handles in a uniform way problems with more than one unknown per grid point, such as for the elasticity problem. Having more degrees of freedom per mesh point does not change the matrix structure, but only makes the blocks denser.



(a) The nonzero structure of the complete finest level stiffness matrix



(b) The nonzero structure of the "face" matrix $\widetilde{K}_{ff}$

Figure 11: Example of the structure of a 3D stiffness matrix: Improved node ordering

In 3D the interface points are subdivided into *coarse*, *edge* and *face* nodes and the matrix K becomes of the form

$$K = \begin{bmatrix} K_{ii} & K_{if} & K_{ie} & K_{ic} \\ K_{fi} & K_{ff} & K_{fe} & K_{fc} \\ K_{ei} & K_{ef} & K_{ee} & K_{ec} \\ K_{ci} & K_{cf} & K_{ce} & K_{cc} \end{bmatrix} \begin{array}{l} \text{i}nterior\ (volume,\ body)\ nodes \\ \text{f}ace\ nodes \\ \text{e}dge\ nodes \\ \text{c}oarse\ (vertex)\ nodes \end{array} \tag{8.8}$$

Here, the submatrices K_{cc}, K_{ee} and K_{ii} enjoy the block-diagonal structure which does not hold for K_{ff} anymore (Figure 11(a)-11(b)). When constructing a preconditioner, however, one can regain the block-diagonal structure by some proper compensation of the entries appearing out of the desired sparsity pattern.

The blocks in K_{ee} correspond to structures, called *wire-baskets*. They are illustrated in Figures 9(a) and 9(b). Under the particular node-numbering shown in Figure 9(a) these blocks have a structure which is suitable for multi-frontal elimination and can be utilized when balancing the overall computational complexity of the solution methods to be used.

Table 17 illustrates the block structure of the test problem matrices, corresponding to different H/h ratios.

8.4 Numerical experiments

We present in this section the numerical results when the defect-correction procedure (Algorithm 8.1) is applied to the Schur-complement based solver, described in Section 8.1, as well as the numerical performance of other solvers for comparison purposes.

			Blocks							
	Fine	Coarse	Coarse		Edge		Face		Body	
$H/2h$	dof	dof	no	size	no	min/max	no	size	no	size
Dam										
2	3474	922	421	2	129	4/14	204	6		
4	13474	922	421	2	129	12/42	204	42		
2	13474	3474	1737	2	461	4/14	816	6		
8	53058	922	461	2	129	28/98	204	210		
4	53058	3474	1737	2	461	12/42	816	42		
2	53058	13474	6737	2	1737	4/14	3264	6		
16	210562	922	461	2	129	60/210	204	930		
8	210562	3474	1737	2	461	28/98	816	210		
4	210562	13474	6737	2	1737	12/42	3264	36		
2	210562	53059	26529	2	6737	4/14	13056	6		
8	838914	13474	6737	2	1737	28/98	3264	210		
4	838914	53058	26529	2	6737	12/42	13056	42		
Bridge										
2	23838	6270	3135	2	861	4/16	1412	6		
4	92862	6270	3135	2	861	12/48	1412	42		
2	92862	23838	11919	2	3135	4/16	5648	6		
8	366462	6270	3135	2	861	28/112	1412	210		
4	366462	23838	11919	2	3135	12/48	5648	42		
2	366462	92862	46431	2	11919	4/16	22592	6		
8	1455870	23838	11919	2	3135	28/112	5648	210		
4	1455870	92862	46431	2	11919	12/48	22592	42		
2	1455870	366462	183231	2	46431	4/16	90368	6		
Bricks										
2	675	135	45	3	12	18/27	11	12	2	78
4	4131	135	45	3	12	72/111	11	84	2	1026
2	4131	675	225	3	45	18/54	68	12	16	78
8	28611	135	45	3	12	180/279	11	516	2	10122
4	28611	675	225	3	45	72/234	68	84	16	1026
2	28611	4131	1377	3	225	18/54	464	12	128	78
8	212355	675	225	3	45	180/594	68	516	16	10122
4	212355	4131	1377	3	225	72/234	464	84	128	1026
2	212355	28611	9537	3	1377	18/54	3392	12	1024	78
Soil										
2	14739	2187	729	3	125	18/54	240	12	64	78
4	107811	2187	729	3	125	72/234	240	84	64	1026
2	107811	14739	4913	3	729	18/54	1728	12	512	78
4	823875	14739		3						
2	823875	107811	35937	3	4913	18/54	13056	12	4096	78
Sluice										
2	87159	12957	4319	3	724	18/54	1436	12	372	78
4	633579	12957	4319	3	724	72/234	1436	84	372	1026
Detail										
2	584127	11484	26355	3	3828	18/54	9350	12	2800	78

Table 17: Structure of the matrices due to applying the DD ordering

Problem size	Time to generate stiff.matr	Solution methods						Total time (sec)	
		N.D.	E.B.E.						
		sol.time	iter.	sol.time	relat.res.	true res.		N.D.	E.B.E.
Bridge									
1722	4.48	0.80	904	15.73	0.98e-8	0.84e-2		5.18	10.21
6262	17.21	4.58	>2000	-	0.11e-2	0.48e+3		21.79	90.59
23790	73.71	51.73	-	-	-	-		125.96	-
92638	307.19	783.88	-	-	-	-		1091.07	-
Sluice									
2172	2.94	1.19	758	21.27	0.93e-8	0.61e-3		4.13	24.26
11379	16.63	41.39	1475	328.78	0.82e-8	0.12e-2		58.02	343.11
74469	132.53	2808.89	>2000	-	0.33e-1	0.30e+2		3087.16	5098.11
234057	442.2	$\approx$ 28200	-	-	-	-		$\approx$ 28600	-
Detail									
42615	82.88	1125.47	>2000	-	0.74e+9	0.56e+3		1208.35	3070.56

Table 18: Results for the ND and EBE commercial FE package solvers

8.4.1 Performance of solvers from a commercial finite element package on some of the benchmarks

Table 18 shows the solution times required to solve some of the benchmark problems considered in this study. Two solution methods are tested - a direct solver based on the nested dissection strategy and an Element-By-Element (EBE) preconditioned CG method. The choice of the EBE to be included in the finite element package is that it is universally applicable to any FE application. One can see, however that its numerical inefficiency is significant for the ill-conditioned test problems and it reveals very slow convergence rate. Comparing the numerical results, one finds a significant difference in the performance of the nested dissection direct solver and the nested dissection algorithm used in Section 3 of Part I. One of the reasons is that this solver has to be seen as a part of a whole package, which uses a particular data base. It requires relatively less computer memory but this leads to slowing down its overall performance.

Despite the big differences in the environment and the data structure used, we find the comparisons with a popular commercial FE software useful in order to be able to evaluate the true advantages of the variety of iterative solution techniques tested in this study.

8.4.2 Pure Schur complement solver

To answer the question how much better a defect-correction solution scheme performs compared to a straightforwardly implemented Schur complement solver (8.4), we include the numerical results presented in Table 19.

$H/2h$	Fine dof	Coarse dof	Schur iter	Coarse iter(av.)	Inner iter(av.)	Time prec	Time solution	Time total	True resid.
Bricks									
2	675	135	17	17	17	0.13	1.19	1.32	3.20e-10
4	4131	135	20	17	45	2.31	40.72	43.31	1.30e-09
2	4131	675	21	31	22	1.83	19.32	21.15	2.30e-10
4	28611	675	26	31	63	22.68	577.46	600.14	1.09e-09
Soil									
2	14739	2187	30	26	15	100.20	125.60	225.79	6.90e-11
2	14739	2187	30	26	35	20.55	168.86	189.41	6.95e-11
4	107811	2187	47	26	36	2153.53	3800.59	5954.12	7.71e-11
2	107811	14739	31	54	40	341.74	1912.93	2254.67	6.49e-11

Table 19: 3D test problems: pure Schur complement solver, RIC2S-PCG for the coarse mesh solver and for the inner (K_{11}) solver

As expected (compare with Table 24), the defect-correction scheme is computationally more efficient.

8.4.3 Block-diagonally preconditioned CG as an inner solver

The simplest inner solver which still can be efficient is a block-diagonally preconditioned CG with a preconditioner

$$B_{11} = \begin{bmatrix} K_{ii} & 0 \\ 0 & K_{ee} \end{bmatrix} \quad or \quad B_{11} = \begin{bmatrix} K_{ii} & 0 & 0 \\ 0 & K_{ff} & 0 \\ 0 & 0 & K_{ee} \end{bmatrix}. \tag{8.9}$$

The blocks for the block-diagonally preconditioned CG are those formed naturally by the chosen ordering strategy during the refinement process. Table 17 illustrates the structure of the matrix and the diagonal blocks used to form the preconditioning matrix. All the blocks despite the so-called *body-blocks* in 3D, which correspond to the interior nodes per subdomain, are solved by a direct method. The body-blocks are replaced by their RIC2S factorization with $\tau = 0.05$. Table 20 illustrates the behaviour of this variant on the 'Dam' test problem. The coarse mesh solver is a RIC2S-preconditioned CG, as for the experiments reported in Table 23. The numerical inefficiency of the block-diagonally preconditioned inner solver compared to the RIC2S-preconditioned inner solver reveals in larger number of inner iterations performed and larger solution time.

8.4.4 Schur complement based solution method combined with a defect-correction procedure

As is clear from Algorithm 8.1, an important issue is how to balance the the stopping tolerances ε_1, ε_2 and ε_3, so that the finally computed $\mathbf{x}^{(k)}$ is within an a priori prescribed accuracy. Table 22 shows the stopping criteria which have been used in the performed numerical experiments.

Tables 23 and 24 show the numerical behaviour of the defect-correction scheme. There, both the coarse and the inner systems are solved by RIC2S-preconditioned CG. The values of the

$H/$ $2h$	Fine dof	Schur iter(av.)	Coarse iter(av.)	Av.inner iter. accur.	Av.inner iter. inaccur.	Time prec.	Time solution	Time total	True res. (relat.)
Dam									
2	3474	2/9	10	59	37	0.1239	2.908	3.032	1.126e-12
4	13474	2/12	10	109	68	0.39	93.15	93.54	1.748e-11
2	13474	2/9	17	68	41	1.409	23.53	24.94	1.558e-12
4	53058	2/13	17	128	78	2.542	656.1	658.7	6.19e-12
2	53058	2/9	31	74	42	9.341	158.1	167.5	3.668e-12
4	210562	2/14	31	137	79	13.8	2930	2943	1.538e-11

Table 20: Block-diagonal PCG as an inner solver

coarse and inner τ are given in brackets next to the problem identifier. As already mentioned, since the coarse mesh can be chosen much smaller than the fine, one can permit very good quality of the RIC2S factorization for the coarse system by choosing smaller threshold parameter τ. Also, utilizing the fact that the inner system is much better conditioned that the whole system, there it suffices with larger values of τ. Except for the largest 'Sluice' problem ($n = 633579$) one observes very reasonable number of inner iterations performed.

Figures 12, 13 and 14 present, whenever possible, a graphical comparison of the methods tested on one and the same computer. Under 'Best direct' (dotted lines) one finds the best results obtained by the direct solvers, reported in Table 1, Part I of this study. The best results from Table 7, Part I, are named 'RIC2S' (solid lines). The results for the defect-correction scheme are depicted by dashed lines under the name 'Schur'. Whenever results from the direct method from the commercial FE package are available, they are plotted by dot-dashed lines, named 'Direct (package)'. On some of the figures the timing results reported in Table 21 for a block-diagonally preconditioned CG method (applied to the fine system) are also included under the name 'Block-diag'.

9 Solution of matrix problems with several right hand sides

We describe several possible solution approaches for solving a linear system $A\mathbf{x} = \mathbf{f}$ with many right hand sides (MRHS) $\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_m$. It will be useful to distinguish the following two formulations of the MRHS (multiple right hand side) problem.

a) All right hand side vectors are available simultaneously. It means that we can consider the MRHS problem as a matrix equation

$$AX = F, \quad X = (\mathbf{x}_1|\mathbf{x}_2|...|\mathbf{x}_m)\,, \quad F = (\mathbf{f}_1|\mathbf{f}_2|, ..., |\mathbf{f}_m)\,. \tag{9.1}$$

b) The right hand side vectors are obtained consecutively, one by one, and we could consider only one single linear system at time,

$$A\mathbf{x}_1 = \mathbf{f}_1, \quad A\mathbf{x}_2 = \mathbf{f}_2, \quad ..., \quad A\mathbf{x}_m = \mathbf{f}_m. \tag{9.2}$$

Obviously, the case (a) looks most advantageous, but in both cases the MRHS problem could be solved more efficiently than if we solve m linear systems apart. We describe several solution

$H/2h$	Fine dof	Coarse dof	Iterations	Time prec	Time solution	Time total	True resid.	time per it
Dam								
2	3474	922	464	0.01	4.48	4.49	1.129e-08	0.01
4	13474	922	722	0.27	49.22	49.50	1.589e-08	0.07
2	13474	3474	938	0.06	42.53	42.49	1.151e-08	0.05
8	53058	922	1061	19.23	855.86	875.09	2.553e-08	0.81
4	53058	3474	1455	1.07	417.90	418.97	1.412e-08	0.29
2	53058	13474	1887	0.24	375.29	375.53	1.097e-08	0.20
8	210562	3474	2143	80.30	7057.10	7137.39	2.132e-08	3.29
4	210562	13474	2929	4.40	3472.97	3477.37	1.334e-08	1.19
2	210562	53059	3790	0.94	3129.10	3130.04	1.080e-08	0.83
Bridge								
2	23838	6270	8991	0.01	751.08	751.18	9.775e-08	0.08
4	92862	6270	14573	1.82	7214.78	7216.60	5.240e-07	0.50
2	92862	23838						
Bricks								
2	675	135	74	0.02	0.24	0.27	8.60e-11	0.01
4	4131	135	123	1.05	5.49	6.52	1.09e-11	0.05
2	4131	675	147	0.20	5.20	5.40	9.26e-11	0.04
8	28611	135	221	36.36	122.35	158.70	1.22e-10	0.55
4	28611	675	240	9.34	85.78	95.12	1.21e-10	0.36
2	28611	4131	288	1.71	81.73	83.44	1.05e-10	0.28
8	212355	675	426	266.70	1776.58	2043.27	1.24e-10	4.17
4	212355	4131	464	75.04	1322.70	1397.74	1.24e-10	2.85
2	212355	28611	563	13.19	1291.48	1304.67	9.68e-11	2.29
Soil								
2	14739	2187	375	0.97	53.93	54.91	9.57e-11	0.14
4	107811	2187	587	131.00	938.61	1069.62	1.15e-10	1.60
2	107811	14739	750	8.07	881.04	889.11	9.96e-11	1.18
Sluice								
2	87159	12957	1928	5.40	1683.36	1688.76	1.23e-10	0.873
4	633579	12957	>4000			>28560	-	≈

Table 21: Block-diagonal PCG with blocks obtained by the special DD ordering

Solver	Stopping crit.	ε	Remark
def.-cor.	$\|\|\mathbf{d}_2^{(k)}\|\| < \varepsilon\|\|\mathbf{f}\|\|$	10^{-8}	See *(dc2), (dc3)*, nonzero initial guess
K_{coars}	$\|\|\mathbf{r}_{coars}^{(k)}\|\| \leq \varepsilon\|\|\mathbf{r}_{coars}^{(0)}\|\|$	10^{-8}	relative, zero initial guess
K_{11} accurate	$\|\|\mathbf{r}_{inner}^{(k)}\|\| \leq \varepsilon$	10^{-10}	absolute, zero initial guess
K_{11} inaccurate	$\|\|\mathbf{r}_{inner}^{(k)}\|\| \leq \varepsilon\|\|\mathbf{r}_{inner}^{(0)}\|\|$	10^{-8}	relative, zero initial guess

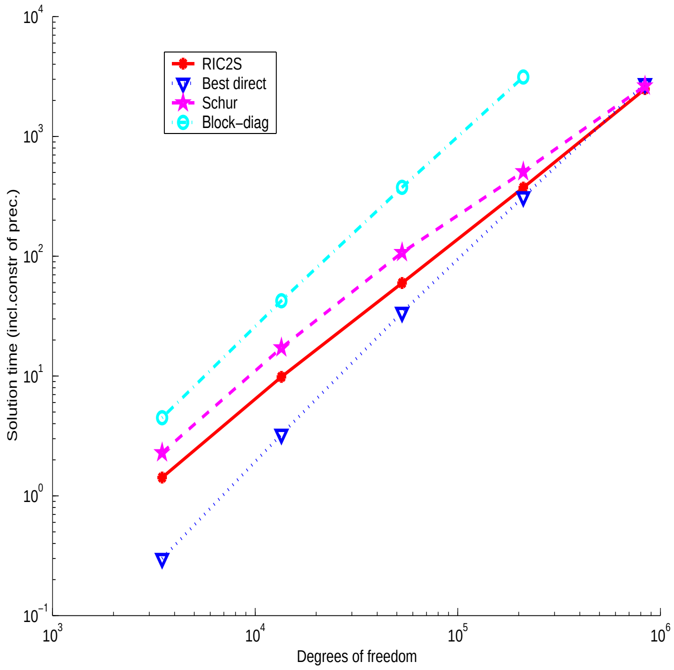
Table 22: Stopping criteria for the defect-correction scheme

$H/$	Fine	Schur	Coarse	Av.inner iter.		Time			True res.
$2h$	dof	iter(av.)	iter(av.)	accur.	inaccur.	prec.	solution	total	(relat.)
Dam: $(\tau_{coars} = 3e-3, \tau_{K_{11}} = 3e-3)$									
2	3474	2/9	10	8	5	0.3714	2.018	2.39	1.084e-12
4	13474	2/12	10	11	7	4.229	17.31	21.54	1.741e-11
2	13474	2/9	17	8	5	3.283	13.93	17.21	1.264e-12
8	53058	2/15	10	16	10	34.5	139.2	173.7	1.656e-11
4	53058	2/13	17	11	7	19.96	87.29	107.3	4.645e-12
2	53058	2/9	31	8	5	19.05	92.22	111.3	2.244e-12
8	210562	2/17	17	17	10	150	657.5	807.5	9.451e-12
4	210562	2/14	31	11	7	90.25	417.3	507.6	6.65e-12
2	210562	2/9	64	8	5	96.31	720.8	817.1	5.302e-12
4	838914	2/14	64	12	7	408.47	2227.48	2635.94	1.87e-11
Bridge $(\tau_{coars} = 3e-4, \tau_{K_{11}} = 3e-3)$									
2	23838	2/8	16	10	5	2.053	22.69	24.74	7.27e-08
4	92862	2/13	16	13	7	15.57	139.8	155.4	8.381e-08
2	92862	3/8	40	11	6	59.99	339.2	399.2	2.08e-08
8	366462	3/16	16	20	10	122.1	1431	1553	1.042e-07
4	366462	3/13	40	13	7	131.5	1148	1280	9.182e-08
2	366462	3/8	78	11	5	438.4	2437	2876	8.418e-08
4	1455870	2/13	78	14	7	799.3	4409	5208	1.055e-06
2	1455870	3/8	161	11	6	2792	21179	23971	3.398e-07

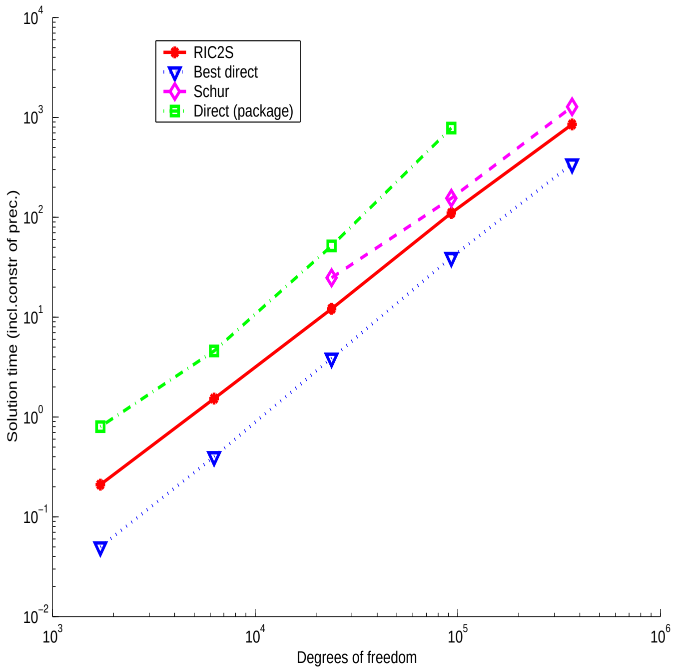
Table 23: Solution of 2D problems using defect-correction

$H/$ $2h$	Fine dof	Schur iter(av.)	Coarse iter(av.)	Av.inner iter. accur.	Av.inner iter. inaccur.	Time prec.	Time solution	Time total	True res. (relat.)
Brick ($\tau_{coars} = 3e-2, \tau_{K_{11}} = 3e-2$)									
2	675	2/9	13	22	9	0.1297	0.8281	0.9578	3.629e-09
4	4131	2/12	13	56	24	2.311	28.35	30.66	8.361e-09
2	4131	2/10	25	29	11	1.734	11.64	13.37	2.896e-09
8	28611	3/14	13	136	62	23.49	1020	1043	9.432e-13
4	28611	2/14	25	82	33	22.12	359.5	381.6	4.736e-09
2	28611	2/11	57	34	13	18.14	152.8	171	7.385e-10
4	212355	2/15	57	94	37	189.9	3559	3749	3.1e-09
2	212355	2/11	121	36	13	162.5	1772	1934	7.303e-10
Soil ($\tau_{coars} = 3e-2, \tau_{K_{11}} = 3e-2$)									
2	14739	2/12	57	43	18	9.93	96.5	106.4	4.873e-09
4	107811	2/20	57	109	46	111.8	2773	2884	6.25e-09
2	107811	2/13	125	50	20	94.09	1256	1350	2.573e-09
2	823875	1/13	149	25	15	2925	7057	9982*	2.881e-6
Soil ($\tau_{coars} = 1e-2, \tau_{K_{11}} = 1e-2$)									
2	14739	2/12	30	24	10	42.32	72.12	114.4	4.625e-09
4	107811	2/20	30	59	25	707.3	1971	2678	5.999e-09
2	107811	2/13	68	27	11	505.2	930.2	1435	2.576e-09
2	823875	1/13	149	26	15	2925	7057	9982	2.881e-06
Sluice ($\tau_{coars} = 1e-2, \tau_{K_{11}} = 1e-2$)									
2	87159	2/15	64	30	19	178.16	962.42	1140.58	7.442e-08
4	633579	2/21	195	211	103	980	35987	36967	3.042e-06
2	633579	2/16	173	47	29	2125	15631	17756	6.589e-08
Detail ($\tau_{coars} = 5e-4, \tau_{K_{11}} = 2e-2$)									
2	584127	2/11	145	59	30	4143	18625	22768	9.779e-06

Table 24: Numerical solution of 3D problems using defect-correction

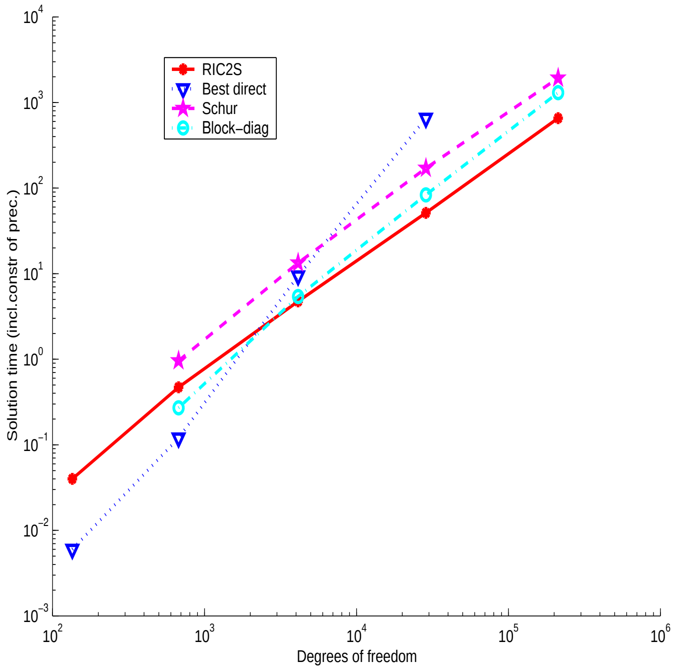


(a) Dam

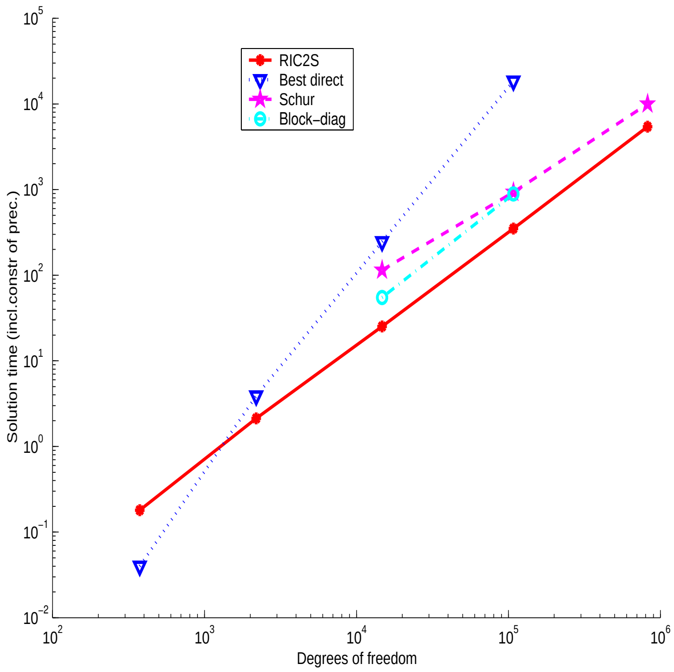


(b) Bridge

Figure 12:

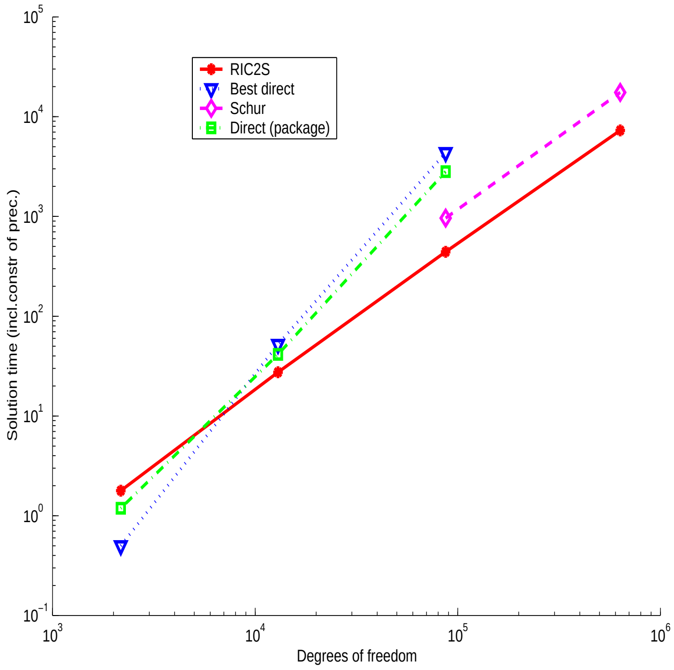


(a) Bricks

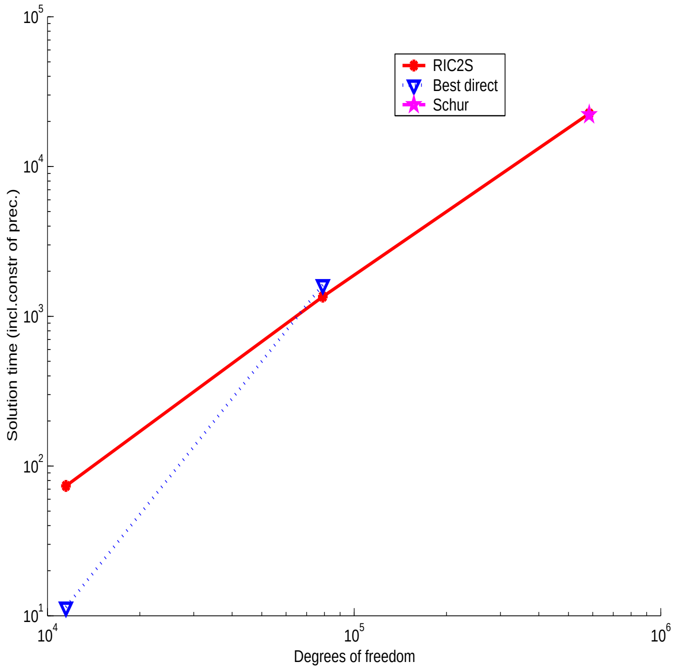


(b) Soil

Figure 13:



(a) Sluice



(b) Detail

Figure 14:

	Bridge $n = 366462$	Dam $n = 838914$	Bricks $n = 28611$	Soil $n = 107811$	Detail $n = 79065$	Sluice $n = 871592$
Method	ND	ND	ND	ND	RCM	ND
T_{tot}, sec.	342.1	2755.0	654.9	18464.9	1625.0	4367
T_{solv}, sec.	5.43	17.16	2.134	27.695	7.61	10.285
$\frac{T_{solv}}{T_{tot}}$	0.0159	0.0062	0.0033	0.0015	0.0047	0.0024

Table 25: The ratio of the time for solution of a factorized system to the total solution time

methods where the gain is twofold. First, the cost of a preparation step which is done once is amortized over many systems. Second, by solving the first systems one can retrieve some useful information about the matrix A, which can be used to accelerate the solution of the remaining systems.

Direct solution methods look as an ideal solution tool for solving the MRHS problem. At least three advantages brought in by the MRHS formulation should be noted here.

First, compared to the solution time, the relative cost for the triangular decomposition of the matrix A, i.e the preparation step $PAP^T = LL^T$, which consist of finding a proper ordering, symbolic and numerical factorizations, is no longer so high even for a small number m of systems to be solved. In Table 25 we display the ratio $c_s = \frac{T_{solv}}{T_{tot}}$, where T_{solv} denotes the time for solving one factorized linear system, and T_{tot} - the total solution time. The results in the table are for the largest problems from the test matrix collection, which has been solved by direct methods. It is seen that the factorization takes almost all the time while the solution of the factorized system itself requires only 1% or less of the total time. Hence, the relative cost of solving m linear systems

$$\frac{T_{tot}^{(m)}}{T_{tot}} = \frac{T_{tot} + (m-1)T_{solv}}{T_{tot}} < 1 + c_s m$$

is growing linearly with m and with a very small coefficient c_s. In other words, the relative cost of solving one of the m systems is increasing as $\frac{1}{m} + c_s$, i.e. for small values of m it is almost proportional to the inverse of m.

Second, if the factorization is done once and the Cholesky factors of the matrix A are saved there is no difference if the right hand sides are available simultaneously or consecutively. We note only, that by rewriting the triangular solution program in a special form, able to handle several right hand sides at once, one could achieve a higher Mflop rate, since internal overhead expense of the solver is also amortized over many solutions, and therefore the overall solution time is reduced.

The third and last observation concerns the implementation of direct methods on parallel computers. Necessary to say, it is much harder to implement a direct algorithm on a parallel computer than an iterative one, in particular to achieve a balanced distribution of computation among processors is especially difficult. The presence of many solution vectors, e.g. enlarging the amount of independent computational tasks per processor could greatly facilitate the scheduling task.

However the use of direct methods remains impractical when the size of the problem is very large. Although in the case of MRHS problem the time to factorize and solve is no longer the restrictive parameter, the amount of memory required remains the main constraint.

Possibly the most popular iterative approach, applicable to the MRHS problem with all right hand side vectors available in advance, is the Block CG algorithm (see [48]), where all vectors in X, e.g. $n \times m$ matrix X, are iterated simultaneously. The block algorithm is rather similar to the traditional one. The same formulae are used, for example, iterations are updated in the same way (compare with the algorithm from Section 4)

$$X_{k+1} = X_k + P_k \alpha_k \tag{9.4}$$

but now X_{k+1}, X_k, P_k are $n \times m$ matrices, and α_k is also an $m \times m$ matrix. Since this algorithm provides the minimization property on each residual vector, and the order of the subspace over which this minimization is done on each iteration increase by m (instead of 1), the convergence of the block method is better. Formal analysis shows that the number of iterations required is governed by the effective condition number $\frac{\lambda_n}{\lambda_m}$. This is favorable when the first $m-1$ eigenvalues forms a cluster, close to the origin, i.e. when $\frac{\lambda_n}{\lambda_m} << \frac{\lambda_n}{\lambda_1}$. On the other hand, the cost of each iteration, as one can see from (9.4), becomes now proportional to $m^2 n$. Unlike direct methods, the cost for each of the m systems grows like m.

Due to space limitations we can not consider various deflation algorithms [17], [19], [42], [50], [39], which, in a few words, aspire to obtain information about eigenvalues and corresponding eigensubspaces and to eliminate the corresponding component in the error vector. Different approaches are possible here. One such method, the Augmented Subspace Preconditioning method has been considered in Section 7. The application of the ASP method to MRHS problem is straightforward. Here we mention one special application of the ASP method, which concern not only MRHS problems, but is of practical importance even when only a single elasticity problem has to be solved. Usually an end-user is interested in the solution not of the linear systems itself, but of the actual discretized problem with some desired accuracy. Many factors influence the resulting accuracy. The quality of discretization and the mesh resolution are the main control parameters. That is why it is natural to solve really large problems on sequence of meshes.

Assume that a linear system $A_h \mathbf{x}_h = \mathbf{f}_h$ should be solved. The subscript h means that this problem is posed on some fine mesh Ω_h. Besides, let a corresponding linear system $A_H \mathbf{x}_H = \mathbf{f}_H$ on a coarse grid Ω_H be available too. Assume also that we can perform one more additional operation, namely an interpolation from the coarse Ω_H to the fine grid Ω_h. In general it is not a cumbersome and costly operation. Then we can find a solution $\mathbf{x}_H$ on the coarse grid at a low price and interpolate $\mathbf{x}_H$ to the fine grid. We can use the interpolation vector $v_h = Interpolate(x_H)$ as a vector V for the ASP method. One can pose m linear systems on the fine grid and get in a such way a $n \times m$ matrix $V_h^{(m)}$. When the ratio H/h is increased, the relative cost on operations on the coarse mesh decreases. Such a matrix V, being used in the ASP iterative process, inserts additional information on the solution, so one can expect a faster convergence of the iterative process. [To be continued.]

10 Parallel solution of s.p.d. linear systems based on overlapped domain decomposition with an inexact RIC2S subdomain solver

In this section, we present test results for the parallel iterative solver proposed in [32, 33] obtained for the set of large scale s.p.d. linear systems arising in FE modeling of linear elasticity problems. The solver is based on overlapped domain decomposition with inexact RIC2S solution over subdomains. We demonstrate the dependence of the parallel performance of the solver on the size of the linear system as well as on the number of processors.

10.1 Method description

In the present section we consider a combined preconditioning strategy for the Conjugate Gradient algorithm intended to achieve good parallelism while providing low iteration costs and fast convergence. The algorithm can be considered as a proper combination of preconditioning strategies described in [30, 31]. The coefficient matrix of the linear system to be solved is split into nearly equal overlapping blocks (the number of which is typically equal to the number of processors available) and then subjected to the Block version of the two-sided Incomplete Inverse Cholesky preconditioning [30]. After rewriting the preconditioner in additive form, each of its block is then approximated by its 2nd order Incomplete Cholesky factorization [31], which greatly reduces the preconditioning costs while almost does not impair its quality.

As a result, most eigenvalues of the preconditioned matrix become highly clustered around unity. Such an eigenvalue distribution ensures low number of the CG iterations needed to achieve the convergence even if the spectral condition number of the preconditioned matrix is still rather large (cf. [5] for details).

Some remarks on the choice of parallel CG preconditioning. Let the linear algebraic system

$$Ax = \mathbf{b} \tag{10.1}$$

with a s.p.d. sparse $n \times n$ matrix A be solved by the PCG method [2] with some preconditioner H. As is well known, the choice of H plays a crucial role, especially when aiming at good convergence, robustness and good parallelization properties of the resulting solution algorithm. One very efficient and robust preconditioner for serial computations turns out to be the RIC2S method, described in Section 4.1. The RIC2S preconditioner possesses the property that the so-called K-condition number of the preconditioned system $H^{-1}A$ is minimized. In the search of an efficient parallel preconditioner with similar properties, several preconditioning strategies have been analyzed with respect to the latter criterium, see [27, 28, 30, 31]. The preconditioning discussed below is actually based on a proper combination of the Block Incomplete Inverse Cholesky (BIIC) preconditioning [27, 29, 30] and the robust Incomplete Cholesky 2nd order (RIC2S) preconditioning [31].

Block explicit preconditioner. Here we briefly describe an approximate version of the Block Incomplete Inverse Cholesky (BIIC) preconditioning algorithm [27, 29, 30]. Let A be reordered and split in some $s \times s$ block form similar as for the Block Jacobi preconditioning, namely, the t-th diagonal block of the symmetrically reordered matrix has dimension n_t and $n_1 + \cdots + n_s = n$. Here $t = 1, ..., s$, and s is the block dimension of A. For the t-th diagonal block, we define a "basic" index set as $\{k_{t-1} + 1, ..., k_t\}$, where $k_{t-1} = n_1 + ... + n_{t-1},\ \ k_0 = 0,\ \ k_s = n$, and introduce an "overlapping" index sets as $\{j_t(1), ..., j_t(m_t - n_t)\}, \quad j_t(p) \leq k_{t-1}$. For each t, the latter index set typically includes those indices not greater than k_t that are the most "essentially" connected to the "basic" index set, e.g. in the sense of the sparse matrix graph adjacency relations. Here $m_t \geq n_t$ and, obviously, $m_1 = n_1$, i.e. at least the first overlapping set is empty. The BIIC-RIC2S preconditioner H (which will be further referenced to as BIIC2) can be represented in the following additive form:

$$H(\tau) = \sum_{t=1}^{s} V_t U_t^{-1} \begin{bmatrix} 0 & 0 \\ 0 & I_{n_t} \end{bmatrix} U_t^{-T} V_t^T,$$

where V_t are rectangular matrices composed of n-dimensional unit vectors e_j as follows:

$$V_t = [e_{j_t(1)}|...|e_{j_t(m_t - n_t)}|e_{k_{t-1}+1}|...|e_{k_t}], \quad t = 1, \ldots, s.$$

Each upper triangular matrix U_t is the (in general, approximate) right Cholesky factor constructed for the t-th "extended" diagonal $m_t \times m_t$ submatrix $V_t^T A V_t$, that is,

$$V_t^T A V_t = U_t^T U_t + U_t^T R_t(\tau) + R_t(\tau)^T U_t, \tag{10.2}$$

(cf. (1.2) of Section 4) where $R_t(\tau)$ is a strictly upper triangular error matrix (i.e., it contains only relatively small entries). The recurrences for the calculation of RIC2S factorization can easily be obtained from (10.2), e.g., when the sparsity patterns of U and R do not have coinciding nonzero positions and their nonzero elements are subjected to the conditions $|U_{i,j}| \geq \tau$ and $|R_{i,j}| < \tau$, respectively, $i < j$. Here $0 < \tau \ll 1$ is the drop tolerance parameter determining the approximation quality of H. Note that with $\tau = 0$ we get $R_t(\tau) = 0$ so that the exact Cholesky factors are used in the construction of the preconditioner $H(0)$, and we obtain the BIIC preconditioner from [27, 29, 30]. The existence and correctness of these RIC2S decompositions are guaranteed for any s.p.d. matrix, since any principal submatrix of A is also s.p.d.. Some modifications of this approach involving less computational effort were already described in Section 4 (cf. [31]). Note that here we have also used $\tau_2 = \tau_1^2$, but $\tau_3 = 0$ since the submatrices are typically considerably better conditioned than the matrix A itself, and therefore no diagonal shift is required.

It was shown in [27, 29, 30] that the BIIC preconditioner $H(0)$ possesses, in a certain sense, the K-optimality property. Moreover, using the properties of the RIC2S and BIIC decompositions one can prove cf. [32, 33] that

$$\log K(H(\tau)A) \leq \log K(H(0)A) + c_0 \tau^2,$$

where c_0 is not very large even for ill-conditioned matrices. Since, by [28, 30], the number of PCG iterations can be bounded from above by $\log_2 K + \log_2 \varepsilon^{-1}$, the latter estimate shows that for some reasonably small RIC2S dropping tolerance τ such BIIC2 hybrid construction will give nearly the same rate of the PCG convergence as for the K-optimal BIIC preconditioner $H(0)$. At the same time BIIC2 involves essentially smaller costs for the evaluation, storage, and the use of the preconditioner as compared to the "exact" BIIC one.

REMARK 10.1 By referring our preconditioning to as “overlapping DD” we only indicate the structural similarity with some other approaches. This similarity is motivated by some non-algebraic considerations related to certain splittings of the underlying physical problem as well as the similar parallel properties of these algorithms.

10.2 Description of the matrix splitting strategy

We do not use any knowledge of the real topology of the domain, so the decomposition of the domain is performed purely algebraically using the sparse matrix graph only. Most of ill-conditioned problems which are investigated in the present report are formulated on very stretched domains, so the simplest method for block splitting turns out to be quite successful because of the narrow bandwidth of the corresponding discretization matrix.

The global bandwidth reduction is performed initially to the whole matrix by Reversed Cuthill-McKee algorithm (RCM), then the node set is divided into s approximately equal “subdomains” (or blocks), and finally the bandwidth inside each of those blocks is again reduced by RCM. The overlap is obtained using sparsity structure of the q-th degree of the coefficient matrix, i.e. the sparsity structure of A^q (see overlap size parameter q in Tables 26 and 27).

More complicated matrix graph splitting techniques (as in the public-domain graph partitioning package METIS [34]) could provide a somewhat better splitting. Experiments with medium size test problems demonstrate that with this technique it is possible to obtain 5-10% reduction of the total solution time. However, the use of METIS for the largest problems was not presently possible due to its excessive memory requirements.

10.3 Parallel implementation

The above described mathematical techniques are implemented in a portable software written in a simplified message-passing style using the MPI-like interface to the low-level communication library.

Let us assume that the linear system (10.1) is solved on a parallel computer having N_{PE} processor elements (PEs) with distributed memory. Let us also assume that $s = N_{\mathrm{PE}}$, i.e. the number of processors coincides with the number of blocks to which the original matrix A is split, and the i-th block corresponds to i-th PE, $i = 1, ..., s$.

The proposed algorithm can be implemented as follows. Perform the RIC2S factorization (see Subsection 4.3) of the local submatrix $V_i^T A V_i$ at the local i-th PE. No data exchanges are required at this stage.

The PCG iterations stage involves the following types of operations:

(i) matrix-vector multiplication with the coefficient matrix A,

(ii) matrix-vector multiplication with the preconditioner $\tilde{H}$, and

(iii) inner product.

Three vector update operations are also needed on each iteration, but they do not require interprocessor communications with our distribution of data, where t-th processor stores the “basic” vector components $k_{t-1} + 1, \ldots, k_t$ and the rows of the matrix A with the same numbers as well as the corresponding block of preconditioner. This implies that all the vector components needed for a vector update are already available at the required PE.

Matrix-vector product. The multiplication of a matrix by a vector in parallel environment is a well investigated problem. It can be presented as the following three-stage algorithm:

(i.1) for any PE requiring some data which are local for the PE, place the required components of the vector to a local data buffer and send them to PEs which require them;

(i.2) receive the required data from the other PEs;

(i.3) multiply the local matrix coefficient data by the vector gathered.

The resulting components of the vector are located at the PE which compute them.

Preconditioner by a vector product. Since the preconditioner is in already factored form, one action if it on a vector can be regarded as a matrix-vector product. The parallel implementation of matrix-vector multiplication by the preconditioner is qualitatively similar to that of A and can be presented as an analogous algorithm. The differences are the following:

(ii.1) the data exchange topology is based on the geometry with overlap;

(ii.2) the type of operation with the local data (local triangular system solutions instead of the local matrix by vector product);

(ii.3) after the first global synchronization, two successive triangular system solutions with U_i^T and U_i are performed, and the second global synchronization operation is required after these local computations.

Inner product. Two inner products are required per each PCG iteration. This is one global exchange MPI-like operation consisting of the following local ones:

(iii.1) the partial inner product for the local part of the vector is computed at each PE;

(iii.2) the scalars obtained are sent to the root PE;

(iii.3) the final scalar product is computed on the root PE;

(iii.4) the final scalar product is sent from the root PE to the other PEs.

Properties of the parallel realization.

(1) The total number of global exchange initialization operations does not depend on the number of blocks and the size of the linear system and is equal to 5 per each PCG iteration.

(2) After completion of the global data exchange, all the computations can be performed on each PE without any additional synchronizations.

(3) There is no serial part in the code implementing the proposed algorithm.

(4) The computations are well balanced if the sizes of the local submatrices are approximately equal.

(5) The communication costs are not large as compared to the arithmetic costs.

10.4 Numerical experiments

In this section, we present numerical results obtained on an eight-processor SUN Enterprise 3000 computer with UltraSparc CPU at 170MHz and compare some BIIC2 preconditioning tuning options. The size of the core memory available for this particular tests was only 1 GB. However, our implementation fitted well this memory size even for the largest problems, with the only exception for the largest Soil problem (823875 degrees of freedom).

The “nearly best” performance results (with respect to the total solution time for a single right hand side) for 8 PEs used are presented in Table 26 for all problems. For more detailed information on the test matrices see Table 1 from Section 4. The BIIC2 tuning options are given in brackets nest to the problem identifier, where the value of the threshold parameter τ as well as the overlap size parameter q are given. Note that the value of τ_1 is taken to be the same as in Section 4 for the sequential RIC2S algorithm. The column "density" shows the quantity $\sum_{t=1}^{s} nz(U_t)/nz(D_A+U_A)$, $s = N_{\rm PE}$, which indicates the (relative) amount of memory needed for the preconditioning. The column "iter." indicates the number of PCG iterations performed. In the next columns the time per one PCG iteration and the wall clock total solution time in seconds are given (the latter includes the construction of the BIIC2 preconditioner and the PCG iterations, but does not include the additional time for scaling, reordering, and computation of the final true residual that is usually not greater than 1% of the total solution time). It should be noted that the wall clock time is always greater than the pure central processor time usually reported for the sequential version of a solver. The column $S_{\rm eff}$ presents the efficient speedup calculated by the formula $S_{\rm eff} = N_{\rm PE}\cdot(t_{\rm total}-t_{\rm mpi})/t_{\rm total}$, where $t_{\rm mpi}$ is the time spend for MPI data exchanges, global synchronizations and all other overheads (delays) in computations. The column S_1 presents the so-called relative speedup. The relative speedup is the ratio $t_{\rm PE}/t_1$, thus the time for the parallel program executed on PE processors, divided by the time for the same program executed on one processor. Column $S_{\rm RIC2S}$ is a measure of the amount of redundancy caused by the parallel implementation and is computed as $t_1/t_{\rm best}$, where t_1 is the total solution time of BIIC2 on one processor and $t_{\rm best}$ is the sequential solution times of the RIC2S-CG (PR) from Table 7 of Section 4.

The iterations start with a zero initial guess $x_0 = 0$, so that $r_0 = b$, and performed until the stopping criterion

$$||r_k|| \le \varepsilon ||r_0||, \qquad \varepsilon = 10^{-8},$$

is satisfied. The actual value of the relative residual norm $||b - Ax_k||/||b||$ is given in the last column as "true res.".

Table 27 presents results for the dependence of the BIIC2 method on parameter tuning. The column "overlap" show the value $\sum_{t=1}^{s} m_t/n$ specifying the total amount of overlap. The table is split into four parts. The first part shows the behavior of the BIIC2 method when increasing the number of the blocks (s). It is seen that the preconditioning quality is only slightly dependent on the number of blocks (equal to the number of PEs) and shows the very good scalability of the method. The second part of the table shows the dependence of the BIIC2 performance on the overlap size parameter q. It is seen that this parameter must only be not too small (say, $q \ge 6$) in order to provide for a good performance, while its further increase cannot change the total solution time essentially. The third part of the table shows that, as expected, the dependence of the total solution time on the dropping threshold parameter τ is not very crucial: over 200% variations of τ change $t_{\rm total}$ only by less than 20%. The fourth part of the table is a comparison with other methods, namely, Point Jacobi preconditioning (PJ), the Block Jacobi (BJ) and the Overlapped Block Jacobi preconditioning (OBJ). The best result in each series is marked by the sign "$^{(*)}$". These results demonstrate the advantages of the BIIC2 preconditioning as compared with the other methods having approximately the same parallel properties.

n	density	iter.	t_{iter}	t_{total}	S_{eff}	S_1	S_{RIC2S}	true res.
Bridge ($\tau = 3 \cdot 10^{-4}$, $q = 40$)								
1722	10.84	14	0.01	0.11	4.53	2.09	2.81	6.69-9
6270	9.98	27	0.02	0.72	5.27	2.86	3.75	4.31-9
23838	9.43	62	0.09	6.48	5.82	3.17	3.91	9.75-9
92862	8.54	152	0.30	53.93	6.46	3.92	4.22	1.24-8
366462	7.92	378	1.11	534.33	6.76	6.04	5.17	5.43-8
Dam ($\tau = 3 \cdot 10^{-3}$, $q = 10$)								
3474	5.19	32	0.01	0.27	4.67	4.22	5.25	8.59-9
13474	4.13	56	0.02	1.59	5.91	5.49	6.18	7.74-9
53058	3.40	109	0.08	11.45	6.54	5.21	5.22	8.03-9
210562	3.05	214	0.30	76.01	7.20	5.52	4.92	6.69-9
838914	2.86	400	1.23	577.27	7.41	5.96	4.30	8.96-9
Brick ($\tau = 3 \cdot 10^{-2}$, $q = 1$)								
135	1.86	19	0.002	0.03	2.23	1.33	1.33	5.61-9
675	1.04	35	0.003	0.08	3.25	3.62	5.87	8.80-9
4131	0.68	73	0.01	0.80	4.62	5.11	5.92	6.15-9
28611	0.53	153	0.07	10.93	6.44	5.01	4.85	7.00-9
212355	0.46	290	0.51	151.15	7.08	5.26	4.34	9.90-9
Soil ($\tau = 3 \cdot 10^{-2}$, $q = 3$)								
375	2.16	22	0.002	0.05	3.05	2.40	3.60	9.09-9
2187	1.54	50	0.01	0.41	4.19	4.34	5.19	3.46-9
14739	1.03	118	0.04	5.90	5.91	4.50	4.27	9.89-9
107811	0.71	258	0.31	81.71	6.77	4.99	4.29	9.79-9
823875	—	—	—	——	—	—	—	——
Detail ($\tau = 2 \cdot 10^{-2}$, $q = 1$)								
11484	0.60	514	0.02	11.31	6.98	7.76	6.50	7.57-9
79065	0.52	1415	0.16	240.02	7.35	6.76	5.63	1.09-8
584127	0.48	3286	1.32	4482.18	7.46	6.65	5.03	2.86-8
Sluice ($\tau = 2 \cdot 10^{-2}$, $q = 1$)								
2172	0.72	81	0.004	0.30	4.35	4.70	5.93	7.79-9
12957	0.57	183	0.02	4.65	6.12	5.84	5.89	5.83-9
87159	0.48	551	0.17	100.92	7.12	5.33	4.38	8.33-9
633579	0.44	1137	1.46	1782.42	7.42	5.61	3.76	9.55-9

Table 26: Matrix data, tuned BIIC2 parameters, and PCG results for the test problems, $p = 8$

s	q	τ	density	overlap	$C(\tilde{H}A)$	iter.	t_{total}	Remark
1	10	3.0-3	2.70	0.00	5.38+4	357	3441.36	
2	10	3.0-3	2.76	0.03	5.04+4	390	2023.69	
3	10	3.0-3	2.72	0.03	4.43+4	398	1371.41	
4	10	3.0-3	2.78	0.05	5.43+4	406	1075.14	
5	10	3.0-3	2.80	0.06	5.62+4	424	928.49	
6	10	3.0-3	2.82	0.07	5.52+4	421	766.40	
7	10	3.0-3	2.84	0.08	5.56+4	425	663.27	
8	10	3.0-3	2.86	0.10	5.86+4	425	577.27	(*)
8	0	3.0-3	2.56	0.00	1.00+5	771	903.87	
8	2	3.0-3	2.61	0.02	7.16+4	552	696.82	
8	4	3.0-3	2.68	0.04	6.56+4	491	634.88	
8	6	3.0-3	2.74	0.06	6.24+4	464	624.95	
8	8	3.0-3	2.80	0.08	6.01+4	441	613.44	
8	10	3.0-3	2.86	0.10	5.86+4	425	577.27	(*)
8	12	3.0-3	2.91	0.12	5.79+4	415	626.14	
8	14	3.0-3	2.96	0.14	5.70+4	406	612.71	
8	16	3.0-3	3.01	0.16	5.64+4	402	631.98	
8	18	3.0-3	3.07	0.17	5.59+4	400	629.71	
8	20	3.0-3	3.12	0.19	5.56+4	398	647.33	
8	10	1.0-3	4.50	0.10	2.16+4	285	690.29	
8	10	2.0-3	3.38	0.10	4.02+4	364	661.85	
8	10	3.0-3	2.86	0.10	5.86+4	425	577.27	(*)
8	10	4.0-3	2.54	0.10	7.62+4	482	690.66	
8	10	5.0-3	2.33	0.10	9.26+4	529	713.73	
8	0	3.0-3	0.00	0.00	3.94+7	11198	4182.84	PJ
8	0	3.0-3	2.56	0.00	1.00+5	771	956.42	BJ
8	10	3.0-3	3.18	0.19	1.45+5	642	1091.38	OBJ
8	10	3.0-3	2.86	0.10	5.86+4	425	577.27	(*) BIIC2

Table 27: Parameters tuning for test problem Dam ($n = 838914$), $s = N_{\text{PE}}$

11 Parallel implementation of the nonoverlapping block - diagonal preconditioned CG, based on a domain decomposition ordering strategy

This section describes the parallel implementation of a standard block-diagonally preconditioned CG method. The only difference is that the diagonal blocks are obtained after applying the domain decomposition ordering strategy described in Section 8.3. As one can see from the results for the serial case, this preconditioning technique performs reasonably for some of the benchmark problems (for example, 'Brick', Table 21) but is in general numerically inefficient. The only reason to implement it in parallel is to study the communication complexity of the matrix structures induced by this particular domain decomposition ordering and to eventually use the programming patterns to implement the more efficient Schur complement solver discussed in Section 8.1. The method is implemented only for 2D structures. The ideology however is directly applicable for the 3D case. The program implementation uses Fortran 77 and MPI standards.

11.1 Parallel implementation issues

In this implementation we consider that the finite elements of the given coarse mesh are (fairly equally) distributed with no overlap among the available processors. As is well known, in terms of mesh nodes this means that some nodes and edges will be shared among neighbouring processors. While the neighbouring processors through an edge are always two, the neighbouring processors through a node (vertex) can be more than 2. To fix terminology, we will refer to *shared edges* and *cross nodes*.

We recall the standard notion of *replicated* and *distributed* vectors (or matrices) in the context of distributed memory parallel environment. A vector is called *distributed* whenever the true value at its entries has to be recovered by summing up the corresponding values from all neighbouring processors. A vector is called *replicated* whenever the values corresponding to cross nodes are the same on all neighbouring processors.

Since the assembly of the stiffness matrix is performed locally (by each processor on its local mesh in parallel), the result is that the stiffness matrix and the right-hand-side vector are distributed.

With this in mind, assuming that we start with an initial guess, which is replicated, it is easily checked that in the preconditioned CG algorithm, Algorithm 11.1, all steps but (5) are consistent and do not require communications.

The preconditioning step (5) is the only place where some communications are required since, in order to preserve consistency, the resulting vector has to be replicated. In our case the preconditioner is block-diagonal, and is replicated in advance. The solution of a system with a replicated matrix and a distributed vector is indeed a distributed vector and has to be communicated before the next CG step (6) takes place.

ALGORITHM 11.1 [Parallel CG] Assume A and $\mathbf{b}$ are distributed and an initial guess $\mathbf{x}^{(0)}$ is given, which is replicated.

$$
\begin{array}{lrcl}
 & \mathbf{g}^{(0)} & = & \mathbf{b} - A\mathbf{x}^{(0)} \\
 & \mathbf{h} & = & [D]\mathbf{g}^{(0)} \\
 & \mathbf{d}^{(0)} & = & -\mathbf{h} \\
 & \delta_0 & = & (\mathbf{g}^{(0)}, \mathbf{d}^{(0)}) \\
\multicolumn{4}{l}{\textit{For } k = 0, 1, \cdots \textit{ until convergence}} \\
(1) & \mathbf{h} & = & A\mathbf{d}^{(k)} \\
(2) & \tau & = & \delta_0/(\mathbf{h}, \mathbf{d}^{(k)}) \\
(3) & \mathbf{x}^{(k+1)} & = & \mathbf{x}^{(k)} + \tau\mathbf{d}^{(k)} \\
(4) & \mathbf{g}^{(k+1)} & = & \mathbf{g}^{(k)} + \tau\mathbf{h} \\
(5) & \mathbf{h} & = & [D]\mathbf{g}^{(k+1)} \\
(6) & \delta_1 & = & (\mathbf{g}^{(k+1)}, \mathbf{h}) \\
(7) & \beta & = & \delta_1/\delta_0, \delta_0 = \delta_1 \\
(8) & \mathbf{d}^{(k+1)} & = & \mathbf{h} + \beta\mathbf{d}^{(k)}
\end{array}
$$

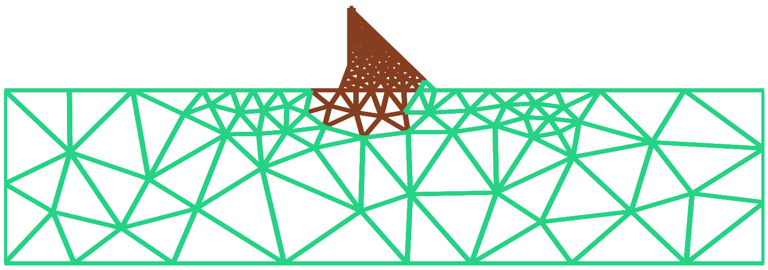
11.2 Numerical experiments

The parallel implementation includes all steps of the solution process, namely, parallel mesh refinement, assembly of the distributed stiffness matrix, factorization of the blocks for the preconditioner and solution of the system by a so-preconditioned CG. The initial (very coarse mesh) mesh is read from a file and it is assumed that the faces are relatively equally split into connected nonoverlapping subdomains. For test problems 'Dam' and 'Bridge' the initial mesh distribution for eight processors is depicted in Figures 15 and 16.

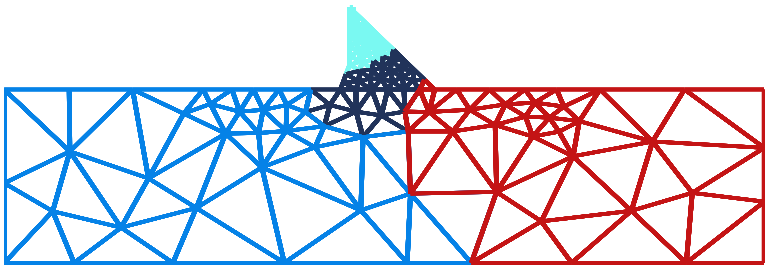
For instance, for Problem 'Dam', in case of eight processors, the initial amount of finite elements per processor is 26, 25, 25, 25, 26, 25, 26, 26, correspondingly. For Problem 'Bridge', it is 174, 170, 184, 176, 158, 182, 184, 184. In general the initial mesh partitioning can be done by some optimized dedicated software but this issue falls out of the scope of the present study.

Despite of the global communications required to compute the scalar products during each CG iteration, the construction and the action of the preconditioner on a vector are the only steps which require communication. Recall that the blocks in the block-diagonal preconditioner in 2D are of three different types - *coarse*, *edge* and *face* blocks. The face blocks correspond to the interior nodes per subdomain and are local per construction. The coarse and edge blocks, however are troublesome since some of them are shared between more than one processor. Further, while the size of the coarse blocks is fixed (2×2 in this case), the size for the edge blocks depends on the initial coarse mesh and varies especially for irregular meshes. As a consequence, the parts of the shared edge blocks in different processors are of different size.

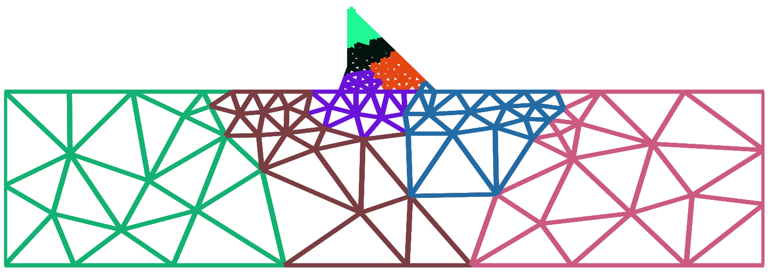
In this application each shared coarse or edge block is gathered to its complete size and replicated in all neighbours which share that block. This is done only once and is counted in the construction phase. Then each block is factorized locally and in parallel. The shared blocks are factorized by each of the sharing processors. The second step requiring communication is the preconditioning step in each CG iteration. Then a system with a block-diagonal matrix is solved where the blocks are replicated but the vector is distributed. The result is also in a distributed



(a) 2 processors



(b) 4 processors

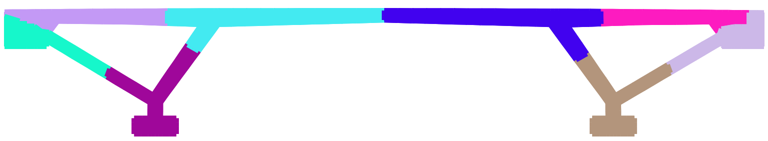


(c) 8 processors

Figure 15: Mesh partitioning for Problem ’Dam’

(a) 2 processors

(b) 4 processors



(c) 8 processors

Figure 16: Mesh partitioning for Problem 'Bridge'

$H/2h$	Fine dof	Iter.	Parallel time 1	2	4	8	Speedup 2	4	8
Dam									
4	13474	722	50.02	27.53	16.64	11.91	1.8	3.0	4.2
2	13474	938	47.64	25.97	16.87	13.94	1.8	2.8	3.4
8	53058	1061	886.99	454.16	243.17	138.45	1.9	3.6	6.4
4	53058	1455	414.00	218.94	120.79	77.70	1.9	3.4	5.3
2	53058	1887	413.39	218.91	124.08	85.84	1.9	3.3	4.8
8	210562	2143	7086.99	3657.96	1924.50	1057.65	1.9	3.7	6.7
4	210562	2929	3358.50	1758.50	957.50	574.55	1.9	3.5	5.8
2	210562	3790	3464.41	1903.15	1053.26	628.27	1.8	3.3	5.5
4	838914	5880	-	-	7716.79	4657.51	-	-	-
Bridge									
2	23838	8843	832.80	413.74	232.95	139.56	2.0	3.6	6.0
4	92862	14367	7110.49	3570.56	1887.57	1028.70	2.0	3.8	6.9
2	92862	18545	7275.89	3539.31	1927.49	1083.72	2.0	3.8	6.7

Table 28: Nonoverlapping block-Jacobi: 2D tests

form and it is replicated in order to preserve consistency in the parallel CG as described in Algorithm 11.1.

Table 11.2 contains the numerical tests performed for problems 'Dam' and 'Bridge'. The columns under "Parallel time' show the total time required for the construction of the preconditioner and for the solution of the system. The achieved relative reduction of the true residual is of the same order as for all other numerical experiments ($O(10^{-8})$). The three columns showing the achieved relative speedup for two, four and eight processors show good scalability, similar to the speedup reported in Table 26.

A separate observation is that the block-diagonal preconditioner used here is less efficient than the block-diagonal preconditioner (BJ) obtained after applying the recursive Cuthill-McKee algorithm, as reported in Table 27.

12 Other approaches

12.1 Incomplete LU-factorization algorithm based on block bordering

This algorithm, referred to as IBBLU, uses a specified block partitioning $A = (A_{ij})_{i,j=1}^n$ of the original matrix A of order $N \geq n$ and computes approximate triangular factors L and U for A in explicit-implicit block form. Namely, if $L = (L_{ij})_{i,j=1}^n$ and $U = (U_{ij})_{i,j=1}^n$ are the block partitionings of L and U induced by the block partitioning of A, then the off-diagonal blocks L_{ij}, $i > j$, and U_{ij}, $i < j$, are found and stored explicitly, whereas instead of the diagonal blocks L_{ii} and U_{ii} (which are themselves lower and upper triangular matrices, respectively) the IBBLU algorithm computes and stores their sparse inverses. Due to this, the solution of linear systems with the matrices L and U, needed when $\tilde{A} = LU$ is used as a preconditioner for A, reduces to multiplying the off-diagonal blocks L_{ij}, U_{ij}, $i \neq j$, and the known approximate inverses L_{ii}^{-1} and U_{ii}^{-1} of the diagonal blocks by vectors. Such a partial explicitness of this preconditioning is obviously beneficial on parallel computing platforms mainly due to the reduction of the recurrence length from N to n. The block format also enhances the sustained flop performance of RISC chips. Furthermore, the block format provides for a considerable reduction of the arithmetic costs for constructing the IBBLU preconditioners as compared to the conventional incomplete LU preconditioners of the same density.

It should be mentioned that, even in the symmetric positive definite (s.p.d.) case, the IBBLU algorithm can in principle break down at the stage of construction either because a certain principal submatrix of the current approximate Schur complement is singular (or highly ill-conditioned), in which case the factorized sparse approximate inverse with the sparsity pattern prescribed cannot be constructed, or because the last diagonal entry of its inverse is (close to) zero, which means that the preconditioner would be (almost) singular. In such cases, one could use, for instance, additive diagonal corrections, as is common in constructing incomplete LU-factorizations, or other more specific tricks. However, in our practice these situations never occurred so far neither in the symmetric nor in the unsymmetric cases. Another important issue is that the IBBLU preconditioning does not preserve the positive definiteness of the original s.p.d. matrix. The arising negative eigenvalues can easily be detected in the course of constructing an IBBLU preconditioner, and one could restore positive definiteness if wanted. But all tricks forcing positive definiteness are known to deteriorate the resulting preconditioning quality. For this reason, we allow the IBBLU preconditioners of s.p.d. matrices to have several negative eigenvalues and apply the GMRES algorithm instead of the CG iteration.

12.1.1 Description of the IBBLU algorithm

The algorithm is applicable also for nonsymmetric matrices and is described in its general form here. Let $\mathcal{A} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$ be a nonsingular matrix of order $N \geq 2$ with square diagonal blocks that allows for a factorization

$$\mathcal{A} = \mathcal{L}\,\mathcal{U}, \tag{12.1}$$

where the matrices $\mathcal{L}$ and $\mathcal{U}$ are lower and upper triangular, respectively. Assuming that a triangular factorization $A = LU$ of the first diagonal block of $\mathcal{A}$ exists, one can write the factorization (12.1) in the form

$$\mathcal{L}\,\mathcal{U} = \begin{bmatrix} L & 0 \\ CU^{-1} & F^{-1} \end{bmatrix} \begin{bmatrix} U & L^{-1}B \\ 0 & G^{-1} \end{bmatrix}, \tag{12.2}$$

where F is lower triangular, G is upper triangular, and

$$GF = (D - CU^{-1}L^{-1}B)^{-1} = (D - CA^{-1}B)^{-1}$$

is an UL-factorization of the inverse to the Schur complement $S = D - CA^{-1}B$. In this case, the matrices F and G will be referred to as inverse triangular factors of S.

In the case where the triangular factors L and U are known only approximately, i.e., $A \approx \tilde{L}\tilde{U}$, one can construct an approximate triangular factorization of the matrix $\mathcal{A}$ in the following form, similar to (12.1):

$$\tilde{\mathcal{A}} = \tilde{\mathcal{L}}\tilde{\mathcal{U}} = \begin{bmatrix} \tilde{L} & 0 \\ X & \tilde{F}^{-1} \end{bmatrix} \begin{bmatrix} \tilde{U} & Y \\ 0 & \tilde{G}^{-1} \end{bmatrix}, \tag{12.3}$$

where $X = \Omega(C\tilde{U}^{-1})$, $Y = \Omega'(\tilde{L}^{-1}B)$ and $\tilde{G}\tilde{F} = \tilde{\Omega}\,[(D - XY)^{-1}]$. Here and below, $\Omega(\cdot)$ and $\Omega'(\cdot)$ denote certain sparse approximations of the matrices in brackets, whereas $\tilde{\Omega}$ is a sparse approximation of the corresponding inverse matrix in the form of a UL-factorization.

Thus, given $\tilde{L}$ and $\tilde{U}$, in order to compute $\tilde{\mathcal{L}}$ and $\tilde{\mathcal{U}}$ in accordance with (12.3), one first needs to compute the matrix products $\tilde{L}^{-1}B$ and $C\tilde{U}^{-1}$ and some sparse approximations Y and X of the former matrices. Also one must determine sparse approximations $\tilde{G}$ and $\tilde{F}$ of the inverse triangular factors of the matrix $D - XY$. To this end, one can use either the FSAI approach, see [35], or the AINV-A algorithm, see [37]. Note that, in the latter case, one does not need to compute the approximate Schur complement $D - XY$ explicitly. On the other hand, when using the FSAI algorithm, which requires that the sparsity patterns of both G and F be prescribed a priori, one does not need to compute all of the entries of the product XY, but only a part of them depending on the sparsity patterns.

Now, using the above considerations concerning the two-by-two block case, we give a general description of the algorithm for constructing approximate or, as is common to say, incomplete LU-factorizations of block matrices. This algorithm is based on block bordering and will be referred to as the IBBLU algorithm.

Let $A = (A_{ij})_{i,j=1}^{n}$, $N \geq n \geq 1$, be a block partitioned matrix with square diagonal blocks A_{ii} of order m_i, $i = 1, \ldots, n$. Define the sequence $A^{(k)}$, $k = 1, \ldots, n$, of principal submatrices of the matrix A as follows:

$$A^{(1)} = A_{11}; \quad A^{(k)} = \begin{bmatrix} A^{(k-1)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{kk} \end{bmatrix}, \quad k = 2, \ldots, n. \tag{12.4}$$

Obviously, $A = A^{(n)}$.

ALGORITHM 12.1 (THE IBBLU ALGORITHM)

(1) Compute lower and upper sparse triangular matrices F_1 and G_1, respectively, such that $G_1F_1 = \tilde{\Omega}_1\left[A_{11}^{-1}\right]$ and set $L^{(1)} = F_1^{-1}; \quad U^{(1)} = G_1^{-1}$.

(2) For $k = 2, \ldots, n$, compute

$$X^{(k)} = \Omega_k\left(A_{21}^{(k)}U^{(k-1)^{-1}}\right); \quad Y^{(k)} = \Omega_k'\left(L^{(k-1)^{-1}}A_{12}^{(k)}\right); \tag{12.5}$$

G_k and F_k such that

$$G_kF_k = \tilde{\Omega}_k\left[(A_{kk} - X^{(k)}Y^{(k)})^{-1}\right] \tag{12.6}$$

and set

$$L^{(k)} = \begin{bmatrix} L^{(k-1)} & 0 \\ X^{(k)} & F_k^{-1} \end{bmatrix}, \qquad U^{(k)} = \begin{bmatrix} U^{(k-1)} & Y^{(k)} \\ 0 & G_k^{-1} \end{bmatrix}. \tag{12.7}$$

(3) Set

$$L = L^{(n)}; \quad U = U^{(n)}; \quad \tilde{A} = LU. \tag{12.8}$$

Next we indicate some specific features of the IBBLU algorithm, which distinguish the latter from other incomplete factorization methods.

• The IBBLU algorithm produces a pointwise incomplete LU-factorization of the original matrix A.

• This pointwise factorization is computed based on a prescribed block partitioning of A.

• The triangular factors L and U are represented in explicit-implicit block form. To be more precise, let $L = (L_{ij})_{i,j=1}^{n}$ be the block partitioning of L induced by the given block partitioning of A. Then the off-diagonal blocks $L_{ij}, \quad i > j$, of the matrix L are stored explicitly, whereas its diagonal blocks (which are themselves lower triangular matrices) are not explicitly available but represented by their sparse inverses: $L_{ii}^{-1} = F_i, \quad i = 1, \ldots, n$. The same holds for the upper triangular factor U. In view of this, the application of the matrix $\tilde{A} = LU$ as a preconditioner for A, which reduces to the solution of linear systems with the matrices L and U, is free of matrix inversion and requires only multiplication of the blocks L_{ij} and $U_{ij},\ i \neq j$, and F_i and $G_i,\ i = 1, \ldots, n$, by vectors.

The above-mentioned features make the IBBLU algorithm different from traditional methods of pointwise incomplete LU-factorization as well as from incomplete block factorization schemes. For instance, in the case where the original matrix $A = (A_{ij})_{i,j=1}^{n}$ is block tridiagonal, i.e., $A_{ij} = 0,\ |i - j| > 1$, the residual matrix $A - LU$, where L and U are computed by applying the IBBLU algorithm, is not in general block diagonal as in the case of incomplete block factorizations (see, e.g., [38]). For this reason, we do not recommend to apply the IBBLU algorithm for block tridiagonal matrices and emphasize that this approach has mainly been designed for unstructured sparse matrices.

It should also be pointed out that in the extreme case $n = 1$ when the original matrix is treated as a single block of order N, the IBBLU algorithm reduces to the algorithm for constructing sparse approximations to the inverse triangular factors and provides an explicit preconditioner for A in factored form.

12.1.2 Construction of sparse approximations to inverse triangular factors

In order to compute sparse approximations F_k and G_k to the inverse triangular factors of the approximate Schur complement $A_{kk} - X^{(k)}Y^{(k)},\ 1 \le k \le n$, we consider the application of the FSAI method see [35], which is briefly recalled below in its simplest form.

Let $H = (h_{ij})$ be a given nonsingular matrix. Fix a lower triangular sparsity pattern S_L and an upper triangular sparsity pattern S_U such that

$$\begin{aligned} &\{(i,j)\ :\ i < j\} \subseteq S_L \subseteq \{(i,j)\ :\ 1 \le i \neq j \le m\}; \\ &\{(i,j)\ :\ i > j\} \subseteq S_U \subseteq \{(i,j)\ :\ 1 \le i \neq j \le m\}. \end{aligned} \tag{12.9}$$

Construct the lower triangular matrix $F = (f_{ij})_{i,j=1}^{m}$ and the upper triangular matrix $G' = (g'_{ij})_{i,j=1}^{m}$ in accordance with the formulas

$$\begin{array}{lclcl} f_{ij} & = & 0, & (i,j) \in S_L; & g'_{ij} & = & 0, & (i,j) \in S_U; \\ \{FH\}_{ij} & = & \delta_{ij}, & (i,j) \notin S_L; & \{HG'\}_{ij} & = & \delta_{ij}, & (i,j) \notin S_U, \end{array} \tag{12.10}$$

where δ_{ij} is the Kronecker symbol. Thus, the nonzero entries of the ith row of the matrix F and the nonzero entries of the ith column of the matrix G' form the last row of the inverse to $H[R_i]$ and the last column of the inverse to $H[C_i]$, respectively, where

$$R_i = \{j \ : \ (i,j) \notin S_L, \ 1 \le j \le i\}, \ C_i = \{j \ : \ (j,i) \notin S_U, \ 1 \le j \le i\}, \ i = 1, \dots, m.$$

It is assumed that all the principal submatrices $H[R_i]$ and $H[C_i]$, $i = 1, \dots, m$, of the matrix H are nonsingular, which ensures that the matrices G' and F are correctly and uniquely determined, which holds for positive definite matrices.

Set $D = \mathrm{diag}\,(FHG')$ and (assuming that D is invertible) $G = G'D^{-1}$. Then, obviously, $\mathrm{diag}\,(FHG) = I$. Note that if $S_U = (S_L)^T$, then from equalities (12.10) it follows that $D = \mathrm{diag}\,(F) = \mathrm{diag}\,(G')$. In the latter case, the matrices F and G' are singular or nonsingular simultaneously, and their nonsingularity amounts to the nonsingularity of D. Taking into account that $f_{ii} = g'_{ii}$ is the last diagonal entry of the inverse to the submatrix $H[R_i]$, $i = 1, \dots, m$, one can guarantee that F, G', and D are nonsingular, provided that the matrix H is either diagonally dominant (or, more generally, an H-matrix) or positive definite.

In application to computing approximate inverse triangular factors F_k and G_k of the approximate Schur complements $A_{kk} - X^{(k)}Y^{(k)}$ in the IBBLU algorithm, we take $S_U^{(k)} = (S_L^{(k)})^T$, and the lower triangular sparsity patterns $S_L^{(k)}$, $k = 1, \dots, n$, are selected by using one of the following two alternative strategies. The first one consists in choosing $S_L^{(k)}$ as the sparsity pattern of the lower triangular part of the matrix $(A_{kk})^r$, where $r = 1, 2, 3$. Here, it can happen that not all of the entries of the product $X^{(k)}Y^{(k)}$ are actually needed, in which case those unnecessary entries are not computed. In accordance with the second strategy, $S_L^{(k)}$ coincides with the sparsity pattern of the lower triangular part of the matrix $\left(A_{kk} - X^{(k)}Y^{(k)}\right)^r$, $r = 1, 2, 3, 4$. In this case, all the entries of the matrix $A_{kk} - X^{(k)}Y^{(k)}$ must be computed. The second strategy seems to be preferable when dealing with highly ill-conditioned matrices. However, for better conditioned matrices the first strategy can be more efficient.

12.1.3 Computation of the off-diagonal blocks of incomplete LU-factors

The off-diagonal blocks $X^{(k)}$ and $Y^{(k)}$ of the matrices $L^{(k)}$ and $U^{(k)}$ are computed, in accordance with (12.5), as sparse approximations to the matrices $A_{21}^{(k)}{U^{(k-1)}}^{-1}$ and ${L^{(k-1)}}^{-1}A_{12}^{(k)}$, respectively. In our implementation of the IBBLU algorithm, we compute the latter matrices explicitly by solving triangular linear systems. In order to sparsify them, we use the following straightforward strategy. For a fixed threshold parameter ε, $0 \le \varepsilon < 1$ (usually, we use $10^{-4} \le \varepsilon \le 10^{-1}$), we define the sets

$$\begin{aligned}
S_k(\varepsilon) &= \left\{(i,j),\ 1 \le i \le m_k,\ 1 \le j \le \sum_{l=1}^{k-1} m_l \ : \ \left|\{A_{21}^{(k)}{U^{(k-1)}}^{-1}\}_{ij}\right| < \varepsilon\right\}, \\
S'_k(\varepsilon) &= \left\{(i,j),\ 1 \le i \le \sum_{l=1}^{k-1} m_l,\ 1 \le j \le m_k, \ : \ \left|\{{L^{(k-1)}}^{-1}A_{12}^{(k)}\}_{ij}\right| < \varepsilon\right\}
\end{aligned}$$

and set

$$\{X^{(k)}\}_{ij} = \begin{cases} 0, & (i,j) \in S_k(\varepsilon) \\ \left\{A_{21}^{(k)} U^{(k-1)^{-1}}\right\}_{ij}, & (i,j) \notin S_k(\varepsilon) \end{cases}, \; 1 \le i \le m_k, \; 1 \le j \le \sum_{l=1}^{k-1} m_l;$$

$$\{Y^{(k)}\}_{ij} = \begin{cases} 0, & (i,j) \in S'_k(\varepsilon) \\ \left\{L^{(k-1)^{-1}} A_{12}^{(k)}\right\}_{ij}, & (i,j) \notin S'_k(\varepsilon) \end{cases}, \; 1 \le i \le \sum_{l=1}^{k-1} m_l, \; 1 \le j \le m_k.$$

Note that this absolute sparsification strategy is usually applied to Jacobi-scaled original matrices A, i.e., we always assume that $a_{ii} = 1, \quad i = 1, \ldots, N$.

When implementing the IBBLU algorithm, a posteriori filtration of the constructed triangular factors L and U can be applied, governed by some parameter $\delta,\ 0 \le \delta \le \varepsilon$. Since, by construction the blocks $X^{(k)}$ and $Y^{(k)},\ k = 2, \ldots, n$, contain no entries smaller than ε in modulus, the postfiltration with $\delta \le \varepsilon$ actually concerns only the inverses L_{ii}^{-1} and $U_{ii}^{-1},\ i = 1, \ldots, n$, to the diagonal blocks of L and U.

12.2 Numerical experiments

The IBBLU method is a quite general tool and its program implementation does not take any advantage of the symmetricity of the system matrix. Also it uses full GMRES as an iterative solver. Due to this reason it is more memory demanding and since the computer used for the rest of the methods to test has only 1 GB of memory, only some of the test problems could be tested. As an example, we compare it with the other solution methods tested in Table 29.

	Iterative				Direct			
Method	RIC2S-MD	RIC2S-PR	Def.cor.	IBBLU	RCM	ND	QMD	ND*
Time (sec)	110.5	227.7	155.4	340.0	45.82	39.54	171.84	783.88
Iter.	55	89	-	40	-	-	-	-

Table 29: Comparison results for Problem 'Bridge', $n = 92862$

Here 'ND*' denotes the time required for the nested dissection solver from the commercial FE package mentioned in Section 8.4.1.

Due to higher memory demands, the largest 3D test problems are tested with IBBLU on a INTEL Pentium II 450MHz workstation. We present these results for completeness (see Table 30).

Test Problem	Size	Density of prec.	Iter.	Total time (sec)
Bricks	212355	4.88	72	734.7
Soil	107811	3.37	79	274.8
Sluice	87159	4.43	92	284.8

Table 30: Performance of the IBBLU method for some 3D test problems

As an illustration, we include some experiments with the parallel implementation of IBBLU, which are performed on an IBM SP computer with POWER2 160 MHz nodes with 512MB of RAM per PE. Note that the SUN UltraSparc computer used for the parallel numerical tests presented in Sections 10 and 11 has faster CPUs (170 MHz). As is seen from Table 31, the method scales without deterioration of the convergence rate.

Number of processors	1	2	3	4
Factorization time (sec)	198	124	45	33
Iteration time (sec)	90	64	36	25
Total time (sec)	288	192	80	59
Number of iterations	55	56	56	57

Table 31: Parallel performance of the IBBLU method: Bridge (2D), $n = 92862$

13 Conclusions

The application of the RIC2S-PCG as a part of another solver, such as the defect-correction scheme or the ASP method, where substantially smaller or better conditioned matrices (compared to the fine level matrix) are factorized, is to be recommended. For such cases the right choice of the threshold parameter τ is of less importance for the overall solution procedure and overshoots or undershoots can not harm severely the global performance. The numerical results for both the ASP method and the Schur complement based solution method confirm the above.

Regarding the defect-correction solution procedure, based on the use of the coarse mesh matrix as a preconditioner for the Schur complement system. The experiments show that the defect-correction performs robustly and in all cases two defect-correction steps are enough to obtain the desired accuracy. However, although not many of them, the iterations required to solve the Schur complement system during each defect-correction step are to be reduced in order to shorten the solution time. For that to be achieved, one has to find still better preconditioner for the Schur complement system. For non M-matrices it turns to be difficult to extract extra information from the implicitly formed Schur complement. That is naturally overcome in the ASP method, which is also better parallelizable compared to all other methods considered.

The block structure imposed on the stiffness matrix as a result of the special node ordering has not been fully utilized so far. A parallel implementation of a nonoverlapping block-diagonally preconditioned CG is available, based on the blocks due to the latter ordering. Handling the shared 'wire-basket' blocks does not entail extra unnecessary commutnications and these structures can be used as building blocks for other implementations in the future.

Regarding the parallel implementation of BIIC2, both theoretical and experimental results show a remarkable efficiency of the algebraic domain decomposition based preconditionings for the cases when the coefficient matrix of the linear system is s.p.d. matrix but not an M-matrix and has very large condition number. The well known fact of increasing speedups, approaching the maximum speedup, with increasing problem size due to a relatively smaller portion of overhead is clearly seen. Despite a rather high computational cost needed by this new factorization, the gain obtained from the considerable reduction in the total costs of PCG iterations far outweighs this drawback, in particular when solving a set of linear equations with the same matrix but many different right hand sides. The parallel efficiency of the solver for multiple right hand sides will also increase due to the relative reduction of number of global synchronizations.

The numerical results presented in Table 26 demonstrate a reasonable speedup in comparison with the single processor runs of both BIIC2 and RIC2S algorithms. The larger is the size of the problem, the greater is the speedup obtained. The absence of slow down even for the smallest problems shows the very efficient usage of the parallel features of the computer system.

Acknowledgements

The work reported in this paper has been supported by The Netherlands Foundation for Technical Sciences (STW), grant NNS.4683 *High Performance Computing of problems in structural Engineering*.

References

[1] M. A. Ajiz and A. Jennings. A robust incomplete Cholesky-conjugate gradient algorithm. *Int. J. Numer. Methods Engrg.*, 20, 949-966, 1984.

[2] O. Axelsson. A class of iterative methods for finite element equations. *Computer Meth. Appl. Mech. Engrg.*, 9, 123-137, 1976.

[3] O. Axelsson. *Iterative solution methods.* Cambridge University Press, Cambridge, 1994.

[4] O. Axelsson and I. Gustafsson. Preconditioning and two level multigrid methods of arbitrary degree of approximation,*Math. Comp.*, Providence R.I., 40 (1983), 219–242.

[5] O. Axelsson and G. Lindskog. On the rate of convergence of the preconditioned gradient methods. Numer. Math. 48 no.5 (1986) 499–523

[6] O. Axelsson and M. Neytcheva. An iterative solution method for Schur complement systems with inexact inner solver. In O. Iliev, M. Kaschiev, S. Margenov, Bl. Sendov, P.S. Vassilevski eds., *Recent Advances in Numerical Methods and Applications II*, World Scientific, 1999, 795–803.

[7] O. Axelsson and M. Neytcheva. Fully Parallel Interface Domain Decomposition Method for Finite Element Elliptic Problems. First Progress report for STW Project NNS.4683 *High Performance Computing of problems in structural Engineering*.

[8] O. Axelsson and M. Neytcheva. Fully Parallel Interface Domain Decomposition Method for Finite Element Elliptic Problems. Second Progress report for STW Project NNS.4683 *High Performance Computing of problems in structural Engineering*.

[9] O. Axelsson and M. Neytcheva. Numerical experiments on benchmark problems. Third Progress report for STW Project NNS.4683 *High Performance Computing of problems in structural Engineering*.

[10] O. Axelsson, M. Neytcheva and B. Polman. An application of the bordering method to solve nearly singular systems, *Vestnik Moskovskogo Universiteta, Seria 15, Vychisl. Math. Cybern.*, No. 1, 3–25, 1996.

[11] O. Axelsson and B. Polman. Block preconditioning and domain decomposition methods II, *J. Computational and Applied Mathematics*, 24 (1988), 55 – 72.

[12] O. Axelsson and V. Vassilevski. Algebraic multilevel preconditioning methods I, *Numer. Math.*,56 (1989), 157-177.

[13] O. Axelsson and P. Vassilevski. A Two-level Stabilized Framework for Interface Domain Decomposition Preconditioners. In I.T.Dimov, Bl.Sendov and P.S.Vassilevski (eds), *Proceedings of the Third International Conference on Numerical Methods and Applications NM&A - $O(h^3)$*, August 21-26, 1994, Sofia, Bulgaria, 196–202.

[14] M. Benzi, R. Kouhia and M. Tuma. An assessment of some preconditioning techniques in shell problems. Technical Report LA-UR-97-3892, Los Alamos National Laboratory, Los Alamos, NM (1997)

[15] J. H. Bramble, J. E. Pasciak and A. H. Schatz. The construction of preconditioners for elliptic problems by substructuring I–IV, *Math. Comp.*. 47 (1986), 49 (1987), 51 (1988), 53 (1989), 103–134, 1–16, 415–430, 1–24.

[16] J. H. Bramble, J. E. Pasciak and A. T. Vassilev. Analysis of nonoverlapping domain decomposition algorithms with inexact solvers, *Math. Comp.*, 67 (1998), 1–19.

[17] K. Burrage, J. Erhel and B. Pole. A deflation technique for linear systems of equations. *SIAM J. Sci. Comput.*, 19, 1245-1260, 1998.

[18] F. F. Campos and N. R. C. Birkett. An efficient solver for multi-right-hand-side linear systems based on the CCCG(η) method with applications to implicit time-dependent partial differential equations. *SIAM J. Sci. Comput.*, 19, 126-138, 1998.

[19] A. Chapman and Y. Saad. Deflated and augmented Krylov subspace techniques. *Numerical Linear Algebra with Applcations*, 4, 43-66, 1997.

[20] M. Dryja, B. Smith and O. Widlund. Schwarz Analysis of Iterative Substructuring Algorithms for Elliptic Problems in Three Dimensions, *SIAM J. Numer. Anal.*, 31 (1994), 1662–1694.

[21] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman. Yale sparse matrix package I: The symmetric codes. *Int. J. Numer. Methods Engrg.*, 18, 1145-1151, 1982.

[22] M. R. Field. Optimizing a parallel conjugate gradient solver. *SIAM J. Sci. Comput.*, 19, 27-37, 1998.

[23] M. T. Jones and P. E. Plassman. An improved incomplete Cholesky factorization. *ACM Trans. Math. Software*, 21, 5-17, 1995.

[24] M. Géraldin, D. Coulon and J.-P. Delsemme. Parallelization of the SAMCEF finite element software through domain decomposition and FETI algorithm, *Int.J. Supercomputer Applic. High Performance Computing*, 11 (1997), 286–298.

[25] G. Haase and U. Langer. The efficient parallel solution of PDEs, *Computers Math. Applic.*, 31 (1996), 151–159.

[26] G. Haase, U. Langer, A. Meyer, and S. V. Nepomnyaschikh. Hierarchical extension operators and local multigrid methods in domain decomposition preconditioners, *East–West J. Numer. Math.* 2(1994), 173–193.

[27] I. E. Kaporin. On preconditioning for the conjugate gradient method when solving discrete analogues of differential problems. Differ. Uravn. 26 no.7 (1990) 1225–1236 (in Russian)

[28] I. E. Kaporin. Explicitly preconditioned conjugate gradient method for the solution of unsymmetric linear systems. *Intern. J. Computer Math.* 40, 169-187, 1992.

[29] I. E. Kaporin. Spectrum boundary estimation for two-side explicit preconditioning. Vestnik Mosk. Univ., ser. 15, Vychisl. Matem. Kibern. 2 (1993) 28–42 (in Russian)

[30] I. E. Kaporin. New convergence results and preconditioning strategies for the conjugate gradient method. *J. Numer. Linear Algebra Appls.* 1, 179-210, 1994.

[31] I. E. Kaporin. High quality preconditioning of a general symmetric positive definite matrix based on its $U^TU + U^TR + R^TU$-decomposition. *Numer. Linear Algebra Appls.* 5, 483-509, 1998.

[32] I. E. Kaporin and I. N. Konshin. Parallel solution of large sparse s.p.d. linear systems based on overlapping domain decomposition. In: Parallel Computing Technologies (Ed. V.Malyshkin), Lecture Notes in Computer Science, Vol. 1662, Springer-Verlag, Berlin – Heidelberg – New-York, (1999) 436–445

[33] I. E. Kaporin and I. N. Konshin. Parallel solution of symmetric positive definite linear systems based on overlapped domain decomposition. (submitted to Zh. Vych. Mat. Mat. Fiz., 2000)

[34] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning, Technical Report 98-036, Dept. Comp. Sci. Engrg., Army HPC Research Center, Univ. of Minnesota, MN (1998)

[35] L.Yu. Kolotilina and A.Yu. Yeremin. Factorized sparse approximate inverse preconditionings. I. Theory, *SIAM J. Matrix Anal. Appl.*, 14, 45-58 (1993).

[36] L.Yu. Kolotilina, A.A. Nikishin and A.Yu. Yeremin. *An Incomplete LU-Factorization Algorithm Based on Block Bordering*,

[37] S.A. Kharchenko, L.Yu. Kolotilina, A.A. Nikishin, and A.Yu. Yeremin. A robust AINV-type method for constructing sparse approximate inverse preconditioners in factored form, *Submitted to NLA*, July 1999.

[38] L.Yu. Kolotilina and A.Yu. Yeremin. Incomplete block factorizations as preconditioners for sparse SPD matrices, in *Recent Advances in Iterative Methods*, The IMA Volumes in Mathematics and its Applications, 60 (1994), pp.119-133.

[39] A.B. Kucherov and M.E. Kornienko. Conjugate gradient proejction methods with incomplete factorization for solving of finite-difference equations. In: Sparse Matrices. Numerical Methods and Algorithms, Moscow State University, 1988, pp. 80-98 (in Russian).

[40] C.-H. Lai, A. M. Cuffe and K. A. Pericleos. A defect equation approach for the coupling of subdomains in domain decomposition methods, *Computers Math. Applic.*, 35 (1998), 81–94.

[41] F.J. Lingen. Design of an object oriented Finite Element package for parallel computers, Ph.D. Thesis, Delft Technical University, 2000.

[42] R. Nicolaides. Deflation of conjugate gradients with application to boundary value problems, *SIAM J. Num. Anal.* 24, 355–365, 1987.

[43] Y. Notay. On the convergence rate of the conjugate gradients in presence of rounding errors. *Numer. Math.* 65, 301-317, 1993.

[44] Y. Notay. Optimal order preconditioning of finite difference matrices, Report GANMN 97-02, Universite Libre de Bruxelles, 1997.

[45] Y. Notay, Using approximate inverses in algebraic multilevel methods, *Numer. Math.*, 80 (1998), 397–417.

[46] Y. Notay. A multilevel block incomplete factorization preconditioning, Appl. Numer. Math. 31, No.2, 209-225 (1999).

[47] Y. Notay. Optimal V cycle algebraic multilevel preconditioning, To appear in *Num. Lin. Alg. Appl.*, 2000.

[48] D. O'Leary. The block conjugate gradient algorithm and related methods. *Linear Algebra Appl.*, 29, 293-322, 1980.

[49] A. Padiy, O. Axelsson and B. Polman. Generalized augmented matrix preconditioning approach and its application to iterative solution of ill-conditioned algebraic systems, *submitted to SIAM Journal of Matrix Anal. and Appl., report 9922, University of Nijmegen*, May, 1999.

[50] Y. Saad. Analysis of augmented Krylov subspace techniques. *SIAM J. Matrix Anal. Appl.*, 18, 435-449, 1997.

[51] P. Saint-Georges, G. Warzee, R. Beauwens and Y. Notay. High-performance PCG solvers for FEM structural analysis. (English) Int. J. Numer. Methods Eng. 39, No.8, 1313-1340 (1996).

[52] M. Tismenetsky. A new preconditioning technique for solving large sparse linear systems. *Linear Algebra Appls.* 154-156, 331-353, 1991.

[53] P. Vassilevski, Nearly optimal iterative methods for solving finite element equations based on multilevel splitting of the matrix. Report 1989-09, Institute for Scientific Computation, Univ. of Wyoming, Laramie.

[54] H. Yserentant. On the multi-level splitting of finite element spaces, *Numer. Math.* 49 (1986), 379-412.