RC 22811, June 6, 2003
Computer Science/Mathematics

IBM Research Report

The Effects of Nonsymmetric Matrix Permutations and Scalings in Semiconductor Device and Circuit Simulation

Olaf Schenk
Computer Science Department
University of Basel
Basel, Switzerland

Stefan Röllin
Integrated Systems Laboratory
Swiss Federal Institute of Technology and Innovation (ETH)
Zurich, Switzerland

Anshul Gupta
Mathematical Sciences Department
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598, USA

Research Division
Almaden · Austin · China · Delhi · Haifa · Tokyo · Watson · Zurich

The Effects of Nonsymmetric Matrix Permutations and Scalings in Semiconductor Device and Circuit Simulation

Olaf Schenk, *Member, IEEE,* Stefan Röllin, and Anshul Gupta, *Member, IEEE*

***Abstract*— The solution of large sparse unsymmetric linear systems is a critical and challenging component of semiconductor device and circuit simulations. The time for a simulation is often dominated by this part. The sparse solver is expected to balance different, and often conflicting requirements. Reliability, a low memory-footprint, and a short solution time are a few of these demands. Currently, no black-box solver exists that can satisfy all criteria. The linear systems from both simulations can be highly ill-conditioned and therefore quite challenging for direct and iterative methods. In this paper, it is shown that nonsymmetric permutations and scalings aimed at placing large entries on the diagonal greatly enhance the reliability of both direct and preconditioned iterative solvers. The numerical experiments indicate that the overall solution strategy is both reliable and very cost effective for unsymmetric linear systems arising in semiconductor device and circuit simulations.**

***Index Terms*— Semiconductor device simulation, circuit simulation, sparse linear solvers, numerical linear algebra, sparse unsymmetric matrices, preconditioning**

I. Introduction

In 1950, Van Roosbroeck [1] introduced the drift-diffusion equations, which are the commonly used model in semiconductor device simulation. The drift-diffusion equations are a system of three coupled, nonlinear partial differential equations (PDEs), which describe the relation between the electrostatic potential and the flux of the charge carriers in a semiconductor device. The coupling between the different equations is highly nonlinear and implies numerical difficulties. More sophisticated models have evolved since the beginnings of semiconductor device simulation to describe the increasingly complex devices.

Different discretizations are used to solve the drift-diffusion equations numerically. The Scharfetter-Gummel box method and finite element discretizations are both used in modern semiconductor device simulators [2], [3]. Regardless of the discretization, a set of nonlinear equations must be solved. Two different approaches are used to solve these nonlinear equations [4]. The Gummel iteration can be seen as a block-Gauss-Seidel iteration on the nonlinear level. The three sets of variables electrostatic potential, electron and hole density are solved in turn, by using the Poisson and continuity equations. A major drawback of this method is the deterioration of the convergence, if the couplings between the variables become too strong. Often, the only possible way is to solve the nonlinear equations simultaneously with the Newton method. The resulting linear systems $Ax = b$, where A is an unsymmetric sparse $n \times n$ matrix, are three times larger than with the Gummel method, highly ill-conditioned and significantly more demanding.

In circuit simulation often a series of linear systems has to be solved. For example, in transient analysis, a differential algebraic equation (DAE) leads in each time-step to a system of nonlinear equations, usually to be solved with the Newton method resulting in very sparse unsymmetric linear systems with a lot of zeros on the diagonal in each matrix.

There are two main approaches to solving the unsymmetric sparse linear systems from both the Gummel and Newton method and the matrices from circuit simulation. The first approach is more conservative and uses sparse direct solver technology. In the last few years algorithmic improvements [5], [6], [7], [8], [9], [10] alone have reduced the time for the direct solution of unsymmetric sparse systems of linear equations by almost one or two order of magnitude. Remarkable progress has been made in the increase of reliability, parallelization and consistent high performance is now achieved for a wide range of computing architectures. As a result, a number of sparse direct solver packages for solving such systems are available [7], [11], [12] and it is now common to solve these unsymmetric sparse linear systems of equations with a direct method that might have been considered impractically large until recently.

Nevertheless, in large three-dimensional simulations with more than 100K grid nodes, the memory requirements of direct methods as well as the time for the factorization may be too high. Therefore, a second approach, namely, preconditioned Krylov subspace methods, is often employed to solve these systems. This iterative approach has smaller memory requirements and often smaller CPU time requirements than a direct method. However, an iterative method may not converge to the solution in some cases where a direct method is capable of finding the solution.

Various iterative methods are known and have been used in semiconductor device and circuit simulations: CGS [13], GMRES(m) [14], BICGSTAB [15] and others. A good preconditioner is mandatory to achieve satisfactory convergence

Manuscript submitted to IEEE TCAD. This work was supported by the Swiss Commission of Technology and Innovation under contract number 5648.1, the IBM T.J. Watson Research Center and the Strategic Excellence Projects of the Swiss Federal Institute of Technology Zurich

O. Schenk is with the Computer Science Department of the University of Basel, Basel, Switzerland. S. Röllin is with Integrated Systems Laboratory of the Swiss Federal Institute of Technology and Innovation (ETH), Zurich, Switzerland. Anshul Gupta is affiliated with the Mathematical Department of the IBM T.J. Watson Research Center, Yorktown, USA.

rates with these methods. There have been several attempts to use preconditioned Krylov subspace methods with different incomplete LU-factorizations (ILU) in this context, but in general the results have been far from satisfactory for non-trivial semiconductor device simulations. The primary concern of the device and circuit engineers is the lack of robustness of the iterative solver.

It is well known that iterative methods work well when the coefficient matrix is, at least to some degree, diagonally dominant or well conditioned. Matrices with these properties arise e.g. from the discretization of second-order, elliptic partial differential equations and simple preconditioning techniques such as ILU(0) are usually reliable under these circumstances and deliver good convergence rates. In contrast, the preconditioners are often unstable and the convergence deteriorates when the coefficient matrix has zeros on the diagonal, and or is highly unsymmetric. Furthermore, preconditioned Krylov subspace methods rarely converge well when the off-diagonal values of the coefficient matrix are an order of magnitude larger than the diagonal entries. This is often the case in semiconductor device simulation and circuit simulations. These systems still pose a challenge for most preconditioned Krylov subspace solvers.

In [16] Olschowka and Neumaier introduce new permutations and scaling strategies for Gaussian elimination to avoid extensive pivoting strategies. The goal is to transform the coefficient matrix A with diagonal scaling matrices D_r and D_c and a permutation matrix P_r so as to obtain an equivalent system with a matrix $D_r P_r A D_c$ that is better scaled and more diagonally dominant. This preprocessing has a beneficial impact on the accuracy of the solver and it also reduces the need for partial pivoting, thereby speeding up the factorization process. These and other heuristics have been further developed and implemented by Duff and Koster [17] and Gupta and Ying [18]. Evidence of the usefulness of this preprocessing in connection with sparse direct solvers has been provided in [10], [19].

For iterative methods, simple techniques like Jacobi or Gauss-Seidel converge more quickly, if the diagonal entry is large relative to the off-diagonals in its row or columns. Additionally, for diagonal preconditioning or incomplete LU factorizations, it is intuitively evident that large diagonals should be beneficial. The contribution of the paper is to carry out an extensive systematic experimental study of the use of nonsymmetric matrix permutations and scalings primary in the context of preconditioned Krylov subspace solvers for semiconductor device and circuit simulation matrices. These nonsymmetric permutations and scalings alter the spectrum distribution of the coefficient matrix, thus they are able to accelerate the convergence and the speedup especially of preconditioned iterative methods. A number of different iterative preconditioners are considered (ILU types and sparse approximate inverse). It is shown that this preprocessing has a stabilizing effect on the computation of the preconditioner and it results in an iterative method of high quality in terms of convergence rates and reliability.

The paper is organized as follows. In Section 2, basic properties of different algorithms based on [17] for computing nonsymmetric matrix permutations and scalings are described. Section 3 briefly reviews current sparse direct solvers and iterative Krylov subspace methods that are routinely used in large semiconductor device and circuit simulations. Computational experience for the algorithms applied to matrices from device simulation and circuit simulation and the effect of the nonsymmetric ordering and scaling applied to direct and iterative methods are presented in Section 4. Finally, in Section 5, the conclusions are presented.

II. Nonsymmetric Permutations

This section gives an introduction to the known techniques to find nonsymmetric permutations, which try to maximize the elements on the diagonal of the matrix. For a deeper understanding, we refer the reader to the original papers of Olschowka and Neumaier [16], Duff and Koster [9], [17] and Benzi, Haws, and Tuma [20].

Matrices with zeros on the diagonal can cause problems for both direct and iterative methods (for the latter the creation of the preconditioner can fail). In some fields like chemical engineering or circuit simulation a lot of zeros happen to be on the diagonal. The matrices originating from device simulation usually, but not always, have zero free diagonals. A remedy is to permute the rows of the matrix, such that only nonzero elements remain on the diagonal. By solving a combinatorial problem, we find a permutation with the desired properties.

Let $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ be a general matrix. The nonzero elements of A define a set $\mathcal{S} = \{(i,j) : a_{ij} \neq 0\}$ of ordered pairs of row and column indices. A subset $\mathcal{M} \subset \mathcal{S}$ is called a matching or a transversal, if every row index i and every column index j appears at most once. A matching $\mathcal{M}$ is called perfect, if the cardinality is equal n. For a nonsingular matrix at least one perfect matching exists and can be found with well known algorithms. With a perfect matching, it is possible to define a permutation matrix $P_r = (p_{ij})$ with:

$$p_{ij} = \begin{cases} 1 & (j,i) \in \mathcal{M} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

As a consequence, the matrix $P_r A$ has nonzero elements on its diagonal, which improves the robustness of both direct and sparse methods. This method only takes the nonzero structure of the matrix into account. There are other approaches which maximize the diagonal values in some sense. One possibility is to look for a matrix P_r, such that the product of the diagonal values of $P_r A$ is maximal. In other words, a permutation σ has to be found, which maximizes:

$$\prod_{i=1}^{n} |a_{\sigma(i)i}|. \tag{2}$$

This maximization problem is solved indirectly. We first reformulate it by defining a matrix $C = (c_{ij})$ with

$$c_{ij} = \begin{cases} \log a_i - \log |a_{ij}| & a_{ij} \neq 0 \\ \infty & \text{otherwise,} \end{cases} \tag{3}$$

where $a_i = \max_j |a_{ij}|$, i.e. the maximum element in row i of matrix A. A permutation σ, which minimizes the sum

$$\sum_{i=1}^{n} c_{\sigma(i)i}. \tag{4}$$

also maximizes the product (2).

The minimization problem is known as (linear sum) assignment problem or bipartite weighted matching problem in combinatorial optimization. The problem is solved by a sparse variant of the Kuhn-Munkres algorithm. The complexity is $O(n^3)$ for full $n \times n$ matrices and $O(n\tau \log n)$ for sparse matrices. For matrices, whose associated graph fulfill special requirements, this bound can be reduced further to $O(n^\alpha(\tau + n \log n)$ with $\alpha < 1$. All graphs arising from finite-difference or finite element discretizations meet the conditions [21]. As before, we finally get a perfect matching, which in turn defines a nonsymmetric permutation. In the literature this permutation is called MPD, which stands for "maximize product on diagonal".

In the solution of the assignment problem, two vectors $u = (u_i)$ and $v = (v_i)$ are generated, which can be used to scale the matrix. These vectors have the property, that they fulfill the following equations:

$$u_i + v_j = c_{ij} \qquad (i,j) \in \mathcal{M}, \tag{5}$$

$$u_i + v_j \leq c_{ij} \qquad \text{otherwise.} \tag{6}$$

Two diagonal matrices D_r and D_c are defined through

$$D_r = \text{diag}(d_1^c, d_2^c, \ldots, d_n^c), \qquad d_j^c = \exp(v_j)/a_j, \tag{7}$$

$$D_c = \text{diag}(d_1^r, d_2^r, \ldots, d_n^r), \qquad d_i^r = \exp(u_i). \tag{8}$$

With the equations (5) and (6), it can be shown, that the scaled and permuted matrix $A_1 = P_r D_r A D_c$ is an I-matrix, for which holds:

$$|a_{ii}^1| = 1, \tag{9}$$

$$|a_{ij}^1| \leq 1. \tag{10}$$

Olschowka and Neumaier [16] introduced these scalings and permutation for reducing pivoting in Gaussian elimination of full matrices. We use the abbreviation MPS for these scalings and the permutation, which stands for "maximize product on diagonal with scalings".

The linear assignment problem can also be used to maximize the sum of the diagonal elements. Instead of (3) the matrix C is defined in the following way:

$$c_{ij} = \begin{cases} a_i - |a_{ij}| & a_{ij} \neq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

In contrary to the maximization of the product (2), it is not possible to derive scalings with the same properties from the linear assignment algorithm in this case. The acronym for "maximize sum of diagonals" is MSD.

Finding a bottleneck transversal is a further possibility to maximize the diagonal elements in some extent. Instead of looking at all diagonal values, a permutation σ is searched, which maximizes the smallest element on the diagonal, i.e. we maximize the expression

$$\min_i |a_{\sigma(i)i}|. \tag{12}$$

Two different approaches are known to achieve this goal. One of them uses a slightly modified variant of the assignment problem. The other defines matrices A_ϵ, where entries with $|a_{ij}| \leq \epsilon$ from the original matrix are dropped. A matching for A_ϵ is then searched. Interval nesting is used to find the optimal ϵ and thus the desired permutation. These methods do not generate unique permutations and are sensitive to a prior scaling of the matrix. A major drawback is that only the smallest values on the diagonal is regarded, which was already reported in [9]. In the section with the numerical results, we use the letters BT for the bottleneck transversal.

III. Solvers for Sparse Linear Systems of Equations

In this section the algorithms and strategies that are used in the direct and preconditioned iterative linear solvers in the numerical experiments are discussed.

A. Sparse direct solver technology

Figure 1 outlines the PARDISO approach [12] to solve an unsymmetric sparse linear system of equations. According to [10], it is very beneficial to precede the ordering by performing a nonsymmetric permutation to place large entries on the diagonal and then to scale the matrix so that the diagonal entries are equal to one. Therefore, in step (1) the row permutation matrix, P_r, is chosen so as to maximize the absolute value of the product of the diagonal entries in $P_r A$. The code used to perform the permutations is taken from MC64, a set of Fortran routines that are included in HSL (formerly known as Harwell Subroutine Library). Further details on the algorithms and implementations are provided in [9]. The diagonal scaling matrices, D_r and D_c, are selected so that the diagonal entries of $A_1 = D_r P_r A D_c$ are 1 in absolute value and its off-diagonal entries are less than or equal to 1 in absolute value.

In step (2) any symmetric fill-reducing ordering can be computed based on the structure of $P_r A + A^T P_r^T$, e.g. minimum degree or nested dissection. All experiments reported in this paper with PARDISO were conducted with a nested dissection algorithm [22].

Like other modern sparse factorization codes [7], [11], PARDISO relies heavily on supernodes to efficiently utilize the memory hierarchies in the hardware. There are two main approaches in building these supernodes. In the first approach, consecutive rows and columns with the same and exactly identical structure in the factors L and U are treated as one supernode. These supernodes are so crucial to high performance in sparse matrix factorization that the criterion for the inclusion of rows and columns in the same supernode can be relaxed [23] to increase the size of the supernodes. This is the second approach and it is called supernode amalgamation. In this approach consecutive rows and columns with nearly the same but not identical structures are included in the same supernode, and artificial nonzero entries with a numerical value of 0 are added to maintain identical row and column structures for all members of a supernode. The rationale is that the slight increase in the number of nonzeros and floating-point operations involved in the factorization can be compensated by a higher factorization speed. Both approaches are possible

(1) Row/column equilibration $A_1 \leftarrow D_r \cdot P_r \cdot A \cdot D_c$, where D_r and D_c are diagonal matrices and P_r is a row permutation that maximizes the magnitude of the diagonal entries.
(2) Find a symmetric permutation P_{fill} to preserve sparsity: $A_2 \leftarrow P_{fill} \cdot A_1 \cdot P_{fill}^T$ and based on $\hat{A} = A_1 + A_1^T$.
(3) Level-3 BLAS factorization $A_2 = Q_r L U Q_c$ with diagonal block supernode pivoting permutations Q_r and Q_c. The growth of diagonal elements is controlled with:
if $(|l_{ii}| < \epsilon \cdot \|A_2\|_\infty)$ **then**
set $l_{ii} = sign(l_{ii}) \cdot \epsilon \cdot \|A_2\|_\infty$
endif
(4) Solve $Ax = b$ using the block L and U factors, the permutation matrices $P_{fill}, P_r, D_r, D_c, Q_r$ and Q_c and iterative refinement.

Fig. 1. Pseudo-code of the complete block diagonal supernode pivoting algorithm for general unsymmetric sparse matrices.

in PARDISO and the first approach has been used in the remainder of the paper.

An interchange among the rows and columns of a supernode, referred to as complete block diagonal supernode pivoting, has no effect on the overall fill-in and this is the mechanism for finding a suitable pivot in PARDISO. However, there is no guarantee that the numerical factorization algorithm would always succeed in finding a suitable pivot within the supernode block. When the algorithm reaches a point where it cannot factor the supernode based on the previously described supernode pivoting, it uses a pivot perturbation strategy similar to [10]. The magnitude of the potential pivot is tested against a constant threshold of $\alpha = \epsilon \cdot \|A_2\|_\infty$, where ϵ is the machine precision and $\|A_2\|_\infty$ is the ∞-norm of the scaled and permuted matrix A_2.

Therefore, in step (3), any tiny pivots encountered during elimination are set to $\mathrm{sign}(l_{ii}) \cdot \epsilon \cdot \|A_2\|_\infty$ — this trades off some numerical stability for the ability to keep pivots from getting too small. Although many failures could render the factorization well-defined but essentially useless, in practice it is observed that the diagonal elements are rarely modified for the large class of matrices that has been used in the numerical experiments. The result of this pivoting approach is that the factorization is, in general, not exact and iterative refinement may be needed in step (4). Furthermore, when there are a small number of pivot failures, they corrupt only a low dimensional subspace and each perturbation is a $\mathrm{rank} - 1$ update of A_2, so iterative refinement can compensate for such corruption with only a few extra iterations.

B. Incomplete LU factorizations and approximate inverses

Iterative methods are usually combined with preconditioners to improve the convergence rates. Especially for ill-conditioned matrices, iterative methods fail without the application of a preconditioner. We briefly discuss some of the most common preconditioners. For a deeper understanding we refer the reader to [24].

Incomplete LU-factorizations: An often used class of preconditioners are incomplete LU-factorizations. In contrary to full Gaussian elimination, the factors L and U are not computed exactly, but some elements are disregarded during the elimination, which makes it more economical to compute, store and solve with. Several strategies have been proposed in the literature to determine which elements are kept and which are dropped. One of the simplest ideas is to keep those elements in L and U, whose corresponding values in the given matrix are nonzero. This version is called ILU(0). It was originally developed for 5-point or 7-point matrices originating from finite difference discretizations of PDEs. Advantages of ILU(0) are its simplicity and the memory requirements, which are known in advance. For some matrices this preconditioner gives quite good results. However, the quality of ILU(0) is often not enough for the convergence of iterative methods and more elaborate incomplete factorizations are necessary.

The concept of "level-of-fill" generalizes ILU(0). In this method, each element of L and U has an associated level lev_{ij} during the elimination. If an element a_{ij} is updated, its level is changed according to

$$\mathrm{lev}_{ij} := \min\{\mathrm{lev}_{ij}, \mathrm{lev}_{ik} + \mathrm{lev}_{kj} + 1\}. \tag{13}$$

The levels are initialized with

$$\mathrm{lev}_{ij} = \begin{cases} 0 & a_{ij} \neq 0 \quad \text{or} \quad i = j \\ \infty & \text{otherwise.} \end{cases} \tag{14}$$

Here, an element is dropped during the factorization, if its level becomes larger than a given threshold p. The motivation for this choice can be explained as follows. Let us suppose, that the elements of A satisfy $a_{ij} = \epsilon^{1+\mathrm{lev}_{ij}}$. Since the elements are updated during the factorization with the formula

$$a_{ij} := a_{ij} - a_{ik} a_{kj}, \tag{15}$$

the size of the new element is about $\epsilon^{\mathrm{lev}_{ij}}$, where lev_{ij} is the updated level according to (13). As a consequence, elements with a large level and thus with small magnitudes are dropped. In the literature this method is called ILU(p). The complexity of this preconditioner is higher than for ILU(0). In addition, the memory demand is not known until the computation is completed. Since it is solely based on the structure of the matrix and the numerical values of the matrix are not taken into account, the resulting preconditioning can be poor.

In the ILUT(ε, q) factorization, the dropping is based on the numerical values rather than the positions. Most incomplete factorizations are either row (or column) oriented. After a row has been computed in the ILUT(ε, q) factorization, all elements in this row of L and U smaller than the given tolerance ε are disregarded. In order to limit the size of the factors, only the q largest elements in each row are kept. For non-diagonally dominant and indefinite matrices, this preconditioner usually gives better results than ILU(p).

The combination of ILU(p) and ILUT(ε, q) leads to a factorization, which is not often mentioned in the literature. We call this method ILUPT(p, ε) in the remainder of this

document. An element is always kept in this factorization, if its level is zero. If the level is positive, the element is dropped if either the value is smaller than the given tolerance ε or its level exceeds p.

Incomplete LU-factorizations are often quite successful in accelerating iterative methods. Nevertheless, some severe problems may happen during the factorization. As in full Gaussian elimination, zero pivots can occur. A remedy is to use pivoting. Because of the underlying data structures, column pivoting is the best choice, if the incomplete factorization is computed row-wise. Not only zero pivots but also small pivots are a problem, since they lead to unstable and inaccurate factorizations. Another cause of inaccuracy is due to the dropping of elements. Each element that is dropped, makes the "error" of the factorization larger. Adapting the parameters may improve this situation.

A successful computation of an incomplete factorization does not guarantee the convergence of an iterative method. For indefinite matrices the factors are often far from being diagonal dominant. Even if the elements of L and U are within reasonable bounds, $\|L^{-1}\|$ and $\|U^{-1}\|$ can be arbitrarily large, which is a sign for unstable triangular solves, which in turn can deteriorate the convergence.

Approximate inverses: The product of the factors in incomplete Gaussian eliminations, directly approximates a given matrix A. Other preconditioners approximate the inverse of A. A major advantage of this approach is that the application of the preconditioner requires one or more matrix-vector products instead of triangular solves. Therefore, these preconditioners are well suited to parallel environments.

The sparse approximate inverse preconditioner SPAI [25] computes a matrix M, which minimizes the Frobenius norm of $I - AM$, where I is the identity matrix. M is equal to A^{-1}, when the norm becomes zero. The inverse is usually a full matrix and thus M is only computed approximately. Since the relation

$$\|I - AM\|_F^2 = \sum_{i=1}^{n} \|e_i - Am_i\|_2^2 \tag{16}$$

holds, each column can be determined independent of the others (e_i and m_i are the i-th columns of I and M). A tolerance ε is given to limit the fill-in and to control the accuracy of the approximation of each column and thus the quality of the preconditioner.

The AINV algorithm is based on an incomplete biconjugation process. Two triangular factors Z and W are computed, which approximate the inverse of A:

$$ZW^T \approx A^{-1}. \tag{17}$$

The triangular matrices are sparse approximations of the factors L and U of a full Gaussian elimination. The biconjugation algorithm computes them without the prior computation of L and U. A dropping tolerance is used to preserve the sparsity of the factors. Again, the application of the preconditioner can be performed fully in parallel.

Like in direct methods, the linear systems are ordered before the preconditioner is computed. All mentioned preconditioners except for SPAI are sensitive to orderings. The purpose of the orderings is twofold. On one hand, the quality of the preconditioner depends on the ordering. In ILU(p) as an example, we have seen, that the dropping is based on the positions only. A different ordering leads therefore to a different preconditioner. On the other hand, the ordering also influences the amount of fill-in in the preconditioners (except for ILU(0)). The Reverse Cuthill-McKee (RCM) [26] ordering is usually used for incomplete LU-factorizations. RCM is not suited for both direct methods and the preconditioner AINV, since it would result in high fill-ins in the factors. Multiple minimum degree and nested dissection are normally used for them.

IV. Description of the test problems

TABLE I
GENERAL INFORMATIONS AND STATISTICS OF THE MATRICES USED IN THE NUMERICAL EXPERIMENTS.

name	unknowns	elements	dim	sim
2D_27628_bjtcai	27'628	442'898	2D	f
2D_54019_highK	54'019	996'414	2D	f
3D_28984_Tetra	28'984	599'170	3D	f
3D_51448_3D	51'448	1'056'610	3D	f
barrier2-9	115'625	3'897'557	3D	d
ibm_matrix_2	51'448	1'056'610	3D	f
igbt3	10'938	234'006	2D	d
matrix-new_3	125'329	2'678'750	3D	f
matrix_9	103'430	2'121'550	3D	f
nmos3	18'588	386'594	2D	d
ohne2	181'343	11'063'545	3D	d
para-4	153'226	5'326'228	3D	d
circuit_1	2'624	35'823	2D	m
circuit_2	4'510	21'199	2D	m
ecl32	51'993	380'415	2D	m
pre2	659'033	5'959'282	2D	m
wang3	26'064	177'168	3D	m
wang4	26'068	177'196	3D	m

This section gives an overview of the matrices that are used for the numerical experiments. Some general informations about the matrices are given in Table I. They are extracted from different semiconductor device simulations with different simulators. A part stems from simulations with FIELDAY [3] from the IBM Thomas Watson Research Center. Those are marked with an "f" in the column "sim". Others, labeled with "d", originate from the semiconductor device simulator DESSIS$_{ISE}$ [2], which is a product of ISE Integrated Systems Engineering Inc. The last matrices wang3 and wang4 come from semiconductor device simulation from a public sparse matrix collection [27] and all other matrices are circuit systems from the same public collection. These matrices are marked with an 'm' in the Table I.

In Table II we have also listed the condition numbers and the number of diagonal dominant rows (d.d. rows) and columns for the original matrices and for the matrices permuted and scaled with MPS. We have used the algorithm MC71 from HSL together with the direct solver PARDISO to estimate the condition numbers.

As stated earlier and shown in Table II, most of the matrices are very ill-conditioned. The influence of MPS on

TABLE II
CONDITIONING AND DIAGONAL DOMINANCE STATISTICS.

	original			scaled and permuted with MPS		
name	condest	d.d.rows	d.d.cols	condest	d.d.rows	d.d.cols
2D_27628_bjtcai	6.46e+19	11'774	13'571	1.16e+07	11'725	13'372
2D_54019_highK	5.85e+31	16'761	23'041	3.99e+07	16'749	21'835
3D_28984_Tetra	1.36e+41	12'869	13'365	1.97e+08	12'869	18'024
3D_51448_3D	1.29e+24	16'097	22'948	5.03e+07	16'110	22'196
barrier2-9	5.49e+22	29'553	5'739	1.07e+19	26'280	53'226
ibm_matrix_2	1.29e+24	16'097	22'950	4.99e+07	16'740	22'337
igbt3	4.74e+19	2'795	121	1.09e+09	2'770	5'237
matrix-new_3	3.48e+22	68'450	78'339	6.59e+08	68'357	72'185
matrix_9	3.08e+23	36'258	38'926	2.71e+07	37'052	40'808
nmos3	1.09e+21	3'068	36	6.28e+06	3'048	6'671
ohne2	3.30e+21	43'283	3'804	2.03e+20	42'509	90'323
para-4	3.70e+23	41'321	6'271	4.22e+20	39'530	76'797
circuit_1	1.89e+05	1'217	1'144	2.38e+06	1'261	2'504
circuit_2	2.69e+04	3'466	560	2.19e+04	3'283	487
ecl32	9.41e+15	31'446	35'574	1.94e+08	31'837	31'745
pre2	3.11e+23	6'729	33	2.99e+14	178'421	152'798
wang3	1.07e+04	16'097	22'950	4.96e+03	21'926	21'676
wang4	4.91e+04	16'097	22'950	5.48e+03	24'762	21'526

these condition number is significant for most of the matrices. The spectrum benefits from nonsymmetric matrix scalings and permutations and the ill-conditioning is greatly reduced for most of the matrices. The number of diagonal dominant rows and columns is also affected by the permutation and scaling and increases by the preprocessing with MPS.

V. NUMERICAL RESULTS

The numerical experiments were performed on one processor on a Regatta pSeries 690 Model 681 SMP nodes with a Power4 running at 1.3 GHz. The processor contains 32 KB L1 data cache, two fixed-point, two floating point and two load/store execution units, associated with a large L2 cache, 1440 KB, and an additional L3 cache directory and controls. All algorithms were implemented in Fortran 77 and C in 64-bit mode. The codes were compiled by *xlf* and *xlc* with the -O3 optimization option and are linked with the IBM's Engineering and Scientific Subroutine Library (ESSL) for the basic linear algebra subprograms (BLAS) that are optimized for RS6000 processors.

A key design principle for producing high-performance scalable parallel, direct solver software is to perform partial pivoting as little as possible, whereas the key design principle for robustness is to allow as much pivoting as possible. A compromise is complete block supernode diagonal pivoting, where rows and columns of a supernode can be interchanged without affecting the computational task-dependency graph. This allows the a priori computation of the nonzero structure of the factors and allows at the same time a limited complete pivoting with the diagonal supernodes blocks. Placing large entries on the diagonal suggests the stability and accuracy of the direct factorization process with reduced pivoting in PARDISO can be improved. This is especially true for circuit simulations, where it is very common for a lot of zeros to appear on the diagonals.

The numerical behavior of the complete block diagonal supernode pivoting method in PARDISO with the original matrix and the nonsymmetric scaled and permuted matrix is illustrated in Table III. A failure in the computation of the factorization (zero pivot) is marked with "•".

Two comments on the results in Table III are in order. First, we notice that for most of the semiconductor device simulation matrices, the number of nonzeros in the factors and the operation count is not affected by a nonsymmetric permutation and scaling. However, we have observed for these matrices that the numerical accuracy still benefits from the permutation. Secondly, it can be seen that MPS greatly improves the robustness of PARDISO and all four circuit simulation matrices can now be solved with a backward error that is close to machine precision.

In this section, we also present the numerical results to see how nonsymmetric permutations influence the iterative solution of the linear systems in semiconductor device and circuit simulation. Usually, incomplete LU-factorizations are used in these fields and therefore we have tested different versions of them. Results with the approximate inverse preconditioner SPAI were also conducted.

For the preconditioning, there are always several choices: either left or right preconditioning can be used. For incomplete factorizations a combination of both is also possible. We applied left preconditioning in all cases. This implies, that the preconditioned residuals do not correspond to the unpreconditioned ones. However, the error of the preconditioned and the original system remains the same. In a majority of the tested matrices, the preconditioned residual and the residuals of the original systems are reduced by the same order of magnitude.

A real right hand side b was used in the numerical experiments for the FIELDAY and DESSIS$_{ISE}$ matrices. For all other matrices an artificial solution of $x_i = 1, 1 \leq i \leq n$ was used. The initial guess x_0 was always zero for the

TABLE III
NUMBER OF FACTOR ENTRIES ($\times 10^6$) AND OPERATION COUNT ($\times 10^9$) IN THE FACTORS COMPUTED BY PARDISO.

Matrix	None		MPS	
	entries	ops	entries	ops
2D_27628_bjtcai	3.12	0.45	3.12	0.45
2D_54019_highK	7.99	1.44	7.99	1.44
3D_28984_Tetra	13.2	8.42	13.2	8.42
3D_51448_3D	28.3	25.5	28.3	25.5
barrier2-9	127.	217.	125.	212.
ibm_matrix_2	28.3	25.5	28.3	25.5
igbt3	1.15	.100	1.15	.100
matrix-new_3	93.4	177.	93.4	177.
matrix_9	93.4	177.	93.4	177.
nmos3	2.47	0.30	2.47	0.30
ohne2	233.	334.	233.	334.
para-4	178.	341.	178.	341.
circuit1	•	•	0.51	0.01
circuit2	•	•	0.47	0.01
ecl32	25.1	22.4	25.1	22.4
pre2	•	•	97.9	166.
wang3	16.1	12.3	7.63	4.58
wang4	6.26	3.02	6.26	3.02

preconditioned iterative methods.

Two of the most famous iterative methods have been tested: BICGSTAB [15] and GMRES(m) [14]. To avoid too high memory consumption, we restarted GMRES after 20 iterations. In both methods, we have implemented two different stopping criteria: the iteration is stopped, if either the preconditioned residual is reduced by a factor of 10^{-8} or 200 iterations are reached. The performance of both iterative methods is comparable. Our numerical results show, that GMRES is faster, if only a few iterations are enough for the convergence. This is especially the case, if we change the tolerance from 10^{-8} to 10^{-4}. However, for the tolerance we used in the numerical experiments, more iteration steps are required and BICGSTAB becomes slightly faster in a majority of the cases. Our experience from complete semiconductor device simulations also indicate to prefer BICGSTAB over GMRES. For these reasons, we present the results for the former only.

As mentioned in Section III-B, a symmetric permutation is usually applied to the linear system before the preconditioner is computed. Thus the iterative solution consists of four steps:

1) Determination of a nonsymmetric matrix permutation and scaling
2) A symmetric permutation with RCM [26] is computed
3) Creation of a preconditioner.
4) Call of an iterative method (BICGSTAB)

The first two steps are optional. For the second step, we use RCM unless otherwise mentioned, since this is often the best choice for incomplete factorizations [28].

In Table IV we have listed the numerical results with and without a nonsymmetric permutation for ILU(0). A failure in the computation of the factorization (zero pivot) is marked with "•". The situations in which the iterative method did not converge are labeled with "‡". The influence of MPS on the number of iterations is not as high as one would expect. For some matrices the number of iterations is even worse with MPS than without. However, in our experience ILU(0) with the nonsymmetric permutation and scalings is slightly more stable. If the iterative method failed with MPS, then without the method did not succeed either. But in some situations, the linear systems could only be solved with MPS. Our results coincide with observations from others [20], that is, if a system can be solved by ILU(0) without MPS, then its influence is not significant or even disadvantageous. We have also tested other nonsymmetric permutations. With BT the incomplete factorization ILU(0) is often similar to the results without a nonsymmetric permutation. However, we have seen in other results not listed here, that ILU(0) with BT is often unstable and the iterative method does not reduce the unpreconditioned residuals at all. The MSD ordering is the worst one: the creation of the preconditioner often fails due to zero pivot. The results from MPD are comparable to those of MPS.

TABLE IV
NUMBER OF ITERATIONS OF BICGSTAB PRECONDITIONED WITH ILU(0).

Matrix	None	BT	MSD	MPD	MPS
2D_27628_bjtcai	‡	‡	‡	160	177
2D_54019_highK	156	156	•	155	169
3D_28984_Tetra	‡	‡	•	‡	‡
3D_51448_3D	66	66	•	81	64
barrier2-9	‡	‡	•	104	128
ibm_matrix_2	88	88	‡	104	93
igbt3	106	106	128	95	107
matrix-new_3	125	125	‡	166	143
matrix_9	68	68	•	71	66
nmos3	‡	‡	‡	‡	‡
ohne2	128	‡	165	‡	167
para-4	‡	‡	‡	‡	168
circuit_1	‡	2	2	2	2
circuit_2	89	17	16	16	14
ecl32	139	139	‡	139	131
pre2	•	•	•	•	•
wang3	•	•	41	41	41
wang4	57	54	57	57	43

In our ILUT(ε, q) implementation we do not limit the number of entries in each row, i.e. we set $q = \infty$. The impact of nonsymmetric permutations on ILUT(0.01,∞) is quite high. The number of iterations for the convergence can be found in Table V. A lot of systems can only be solved with MPS. However, not every nonsymmetric permutation gives good results. The quality of BT and MSD with ILUT is comparable to the findings with ILU(0). A better choice is MPD, which is better than omitting a nonsymmetric permutation. The MPS ordering requires the fewest iterations for convergence in almost all cases. In addition, there are often fewer nonzeros in the incomplete factors for this ordering than for the others. As a result, the computation of the factorization is faster and one iteration step is cheaper with MPS.

The results of the ILUPT(5,0.01) factorization are given in Table VI. Here, the behavior is again different than for the previous two factorizations. A lot of systems can be solved in combination with or without nonsymmetric permutation. What remains the same is the quality of the MSD ordering, where the computation of the preconditioner sometimes fails. The

TABLE V
NUMBER OF ITERATIONS OF BICGSTAB PRECONDITIONED WITH ILUT(0.01,∞)

Matrix	None	BT	MSD	MPD	MPS
2D_27628_bjtcai	‡	‡	‡	‡	125
2D_54019_highK	‡	‡	•	‡	89
3D_28984_Tetra	‡	‡	•	‡	79
3D_51448_3D	135	135	•	66	32
barrier2-9	‡	‡	•	‡	171
ibm_matrix_2	136	136	•	68	34
igbt3	‡	‡	‡	‡	54
matrix-new_3	88	88	‡	74	47
matrix_9	77	77	•	57	30
nmos3	‡	‡	‡	‡	22
ohne2	‡	‡	‡	‡	85
para-4	77	‡	‡	‡	53
circuit_1	•	3	3	3	4
circuit_2	24	33	15	15	5
ecl32	44	44	‡	44	55
pre2	•	•	•	•	•
wang3	•	•	24	24	29
wang4	31	27	31	31	23

results of BT are better than before, but still not optimal in the sense, that the unpreconditioned residuals are not reduced. Again, MPD is a better alternative. On the average, the best choice is MPS, but the difference with MPD and without a nonsymmetric permutation is not as high as for ILUT.

TABLE VI
NUMBER OF ITERATIONS OF BICGSTAB PRECONDITIONED WITH ILUPT(5,0.01)

Matrix	None	BT	MSD	MPD	MPS
2D_27628_bjtcai	155	155	70	89	76
2D_54019_highK	96	96	•	100	83
3D_28984_Tetra	194	194	•	100	86
3D_51448_3D	59	59	•	59	37
barrier2-9	72	81	•	47	‡
ibm_matrix_2	64	64	‡	60	39
igbt3	36	36	57	37	44
matrix-new_3	89	89	‡	87	48
matrix_9	50	50	•	52	31
nmos3	51	51	112	48	30
ohne2	59	‡	59	90	72
para-4	51	‡	45	51	53
circuit_1	•	2	2	2	2
circuit_2	•	11	11	11	5
ecl32	89	89	‡	89	54
pre2	•	200	•	•	•
wang3	•	•	46	46	27
wang4	43	46	43	43	21

The number of iterations give a good indication of the reliability of preconditioner. However, the total time to perform the four steps for the iterative solution is more important because it directly influences the time for a semiconductor device simulation. In Table VII, the total time for different nonsymmetric orderings and preconditioners are given. ILU(0) without a nonsymmetric permutation is often faster than with MPS. However, the latter succeeds for more systems. ILUT(0.01,∞) together with MPS is for all but one examples the fastest combination. It is between two and three times faster and significantly more stable than ILU(0). The performance of ILUPT(5,0.01) is slightly better than ILU(0) but does not reach the one for ILUT. A comparison of the times to perform the factorization shows, that ILUT is much cheaper to compute than ILUPT. For the largest matrices, the former is about ten times faster. The large difference comes from the fact, that ILUPT contains at least the nonzero structure of the original matrix. This has a significant influence on the number of elements in the factors. As an example, for matrix “ohne2” about six times more elements appear in the factors of ILUPT. As a result, this makes both the factorization and one iteration step more expensive. Interestingly to note is the observation, that MPS reduces the factorization time of ILUPT in about half of the systems, but the number of nonzeros remains the same.

TABLE VIII
NUMBER OF ITERATIONS OF BICGSTAB PRECONDITIONED WITH SPAI

	SPAI(0.1)		SPAI(0.4)	
Matrix	None	MPS	None	MPS
2D_27628_bjtcai	‡	‡	‡	‡
2D_54019_highK	‡	‡	‡	‡
3D_28984_Tetra	‡	103	‡	178
3D_51448_3D	‡	84	‡	143
barrier2-9	‡	131	‡	181
ibm_matrix_2	‡	77	‡	151
igbt3	‡	‡	‡	‡
matrix-new_3	‡	145	‡	‡
matrix_9	‡	102	‡	154
nmos3	‡	54	‡	97
ohne2	•	•	•	•
para-4	‡	101	‡	‡
circuit_1	4	4	6	6
circuit_2	‡	60	‡	69
ecl32	‡	‡	‡	‡
pre2	‡	‡	‡	‡
wang3	63	63	128	123
wang4	‡	79	‡	92

The preconditioner SPAI can be a good choice in parallel environments, since the computation as well as the application of it is entirely parallel. The goal of our numerical experiments with this preconditioner was to answer the questions, whether nonsymmetric permutations and scalings are beneficial for SPAI or not and if it is a viable alternative to incomplete LU-factorizations. The computation of SPAI does not depend on symmetric orderings and therefore we omitted them for the numerical experiments.

Our numerical results with SPAI are summarized in Table VIII. The quality of the preconditioner SPAI is influenced by a tolerance. Two different tolerances have been used in the experiments. For the original matrices, BICGSTAB preconditioned with SPAI failed to converge for nearly all matrices, i.e. the maximum number of iterations were reached. For those matrices, where 200 iterations are reached, the preconditioned residuals were nevertheless reduced by some amount. However, the corresponding unpreconditioned residuals were not reduced at all, but increased by some orders. The preprocessing with MPS helped in both regards: the iterative method succeeded for much more matrices than before. In addition, the gap between the preconditioned and

TABLE VII
TOTAL TIME IN SECONDS FOR COMPLETE ITERATIVE SOLUTION.

	ILU(0)		ILUPT(5,0.01)		ILUT(0.01,∞)		SPAI(0.1)		SPAI(0.4)	
Matrix	None	MPS	None	MPS	None	MPS	None	MPS	None	MPS
2D_27628_bjtcai	‡	2.53	2.13	1.19	‡	1.26	‡	‡	‡	‡
2D_54019_highK	4.45	5.11	3.15	2.95	‡	2.08	‡	‡	‡	‡
3D_28984_Tetra	‡	‡	3.26	1.81	‡	1.02	‡	19.9	‡	5.20
3D_51448_3D	2.56	2.53	2.51	2.29	8.54	1.18	‡	38.5	‡	9.08
barrier2-9	‡	22.2	15.0	‡	‡	10.9	‡	401	‡	89.9
ibm_matrix_2	2.96	3.34	2.70	2.36	8.54	1.25	‡	38.3	‡	9.33
igbt3	0.68	0.75	0.36	0.40	‡	0.31	‡	‡	‡	‡
matrix-new_3	10.4	11.2	7.93	6.39	23.3	3.35	‡	145	‡	‡
matrix_9	4.86	4.80	4.73	5.05	18.4	3.08	‡	82.8	‡	14.7
nmos3	‡	‡	0.77	0.54	‡	0.33	‡	9.11	‡	2.55
ohne2	66.0	86.6	50.8	46.9	‡	18.1	•	•	•	•
para-4	‡	35.8	16.9	13.2	6.63	6.31	‡	382	‡	‡
circuit_1	0.02	0.02	•	1.17	•	0.01	3.01	3.10	1.45	1.45
circuit_2	0.08	0.02	•	0.03	0.02	0.01	‡	1.23	‡	0.943
ecl32	2.81	2.76	2.08	1.90	2.72	1.70	‡	‡	‡	‡
pre2	•	•	•	•	•	•	‡	‡	‡	‡
wang3	•	0.46	•	0.51	•	0.54	6.16	8.24	1.50	1.10
wang4	0.53	0.46	0.46	0.38	0.62	0.40	‡	7.61	‡	1.48

unpreconditioned residuals was smaller than before.

The reduction of the tolerance from 0.4 to 0.1 in SPAI lead to a significant improvement in the number of iterations and more systems were successfully solved. On the other hand, the time to compute the approximate inverse preconditioner increases about the same factor, in which the tolerance is reduced. Since for SPAI the overall solution time is dominated by the computation of the preconditioner, the larger tolerance usually results in a smaller solution time, despite the number of iterations is larger.

The numerical experiments revealed some disadvantages of the sparse approximate inverse. First of all, SPAI could solve fewer systems than the incomplete factorizations. Comparing the number of iterations to reduce the preconditioned residuals by 10^{-8}, the incomplete LU-factorizations require fewer iterations. The major drawback of SPAI is the expensive computation. The creation of SPAI(0.4), which is faster than SPAI(0.1), takes between 10 up to 500 times longer than an ILUT(0.01,∞) factorization, despite the use of a highly optimized implementation. The total times are also given in Table VII. On the average, SPAI(0.4) is between 2 and 8 times slower compared with the best incomplete factorization. In addition, complete semiconductor device simulations revealed, that SPAI is not robust enough and does not give satisfactory results.

VI. CONCLUSION

We have presented numerical experiments with linear systems originating from semiconductor device simulation and from circuit simulations to study the influence of nonsymmetric permutations on direct and iterative solvers. The results show, that the use of nonsymmetric permutations can improve the performance for both classes of linear solvers.

The numerical experiments for the direct method for unsymmetric general matrices indicate that the use of row permutations with complete block diagonal supernode pivoting enables the static computation of the task-dependency graph, resulting in an overall factorization strategy that is both reliable and cost-effective on shared memory multiprocessing architectures (SMPs). Further evidence of the usefulness of nonsymmetric matrix scalings on SMPs has been provided by Schenk and Gärtner in [12].

From our experiments, we see that, for the preconditioned iterative Krylov subspace solvers, nonsymmetric permutations combined with scalings give the best results in terms of the number of required iterations and the time to compute the solution. Especially for the preconditioner ILUT, where the dropping is based on the numerical values, the nonsymmetric permutation has a significant impact. The influence is smaller for the incomplete factorizations ILU(0) and ILUPT, where the positions of the values determine the dropping. The robustness of those preconditioners is nevertheless improved with the use of the nonsymmetric permutations. On an average, the most efficient preconditioner for our matrices was the ILUT factorization with nonsymmetric scalings and permutation (ILUT+MPS), which outperformed the others in a number of cases. The method ILUT+MPS is both robust and cost effective and it is the only algorithm that could solve nearly all our test matrices from semiconductor device and circuit simulations.

The results conducted with SPAI showed, that this preconditioner is not competitive with the incomplete LU-factorizations for the matrices occurring in device and circuit simulation. The time to compute the preconditioner dominates the solution process and is several times larger than for the incomplete LU-factorizations. In addition, the robustness of SPAI is worse for matrices from semiconductor device and circuit simulations.

ACKNOWLEDGMENT

The authors wish to thank Iain Duff and Jacko Koster for the opportunity to use the MC64 graph matching subroutines. Parts of this work were performed while the first author was an academic visiting scientist at the IBM T.J. Watson Research Center. The hospitality and support of IBM are greatly appreciated.

REFERENCES

[1] W. V. Roosbroeck, "Theory of flow of electrons and holes in germanium and other semiconductors," *Bell Syst. Tech. J.*, vol. 29, pp. 560–607, 1950.

[2] Integrated Systems Engineering AG, *DESSIS_ISE Reference Manual*, ISE Integrated Systems Engineering AG (http://www.ise.com), 2003.

[3] IBM Thomas Watson Research Center, *Fielday Reference Manual*, IBM Thomas Watson Research Center (http://www.research.ibm.com), 2003.

[4] R. Bank, D. Rose, and W. Fichtner, "Numerical methods for semiconductor device simulation," *SIAM Journal on Scientific and Statistical Computing*, vol. 4, no. 3, pp. 416–435, 1983.

[5] A. Gupta, "Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices," *SIAM Journal on Matrix Analysis and Applications*, vol. 24, no. 2, pp. 529–552, 2002.

[6] ——, "Recent advances in direct methods for solving unsymmetric sparse systems of linear equations," *ACM Transactions on Mathematical Software*, vol. 28, no. 3, pp. 301–324, September 2002.

[7] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, "A fully asynchronous multifrontal solver using distributed dynamic scheduling," *SIAM J. Matrix Analysis and Applications*, vol. 23, no. 1, pp. 15–41, 2001.

[8] O. Schenk, K. Gärtner, and W. Fichtner, "Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors," *BIT*, vol. 40, no. 1, pp. 158–176, 2000.

[9] I. S. Duff and J. Koster, "The design and use of algorithms for permuting large entries to the diagonal of sparse matrices," *SIAM J. Matrix Analysis and Applications*, vol. 20, no. 4, pp. 889–901, 1999.

[10] X. Li and J. Demmel, "A scalable sparse direct solver using static pivoting," in *Proceeding of the 9th SIAM conference on Parallel Processing for Scientic Computing*, San Antonio, Texas, March 22-34,1999.

[11] A. Gupta, "WSMP: Watson sparse matrix package (Part-II: direct solution of general sparse systems," IBM T. J. Watson Research Center, Yorktown Heights, NY, Tech. Rep. RC 21888 (98472), November 20, 2000.

[12] O. Schenk and K. Gärtner, "Solving unsymmetric sparse systems of linear equations with PARDISO," *to appear in Future Generation Computer Systems*, 2003.

[13] P. Sonneveld, "CGS, a fast Lanczos-type solver for nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 10, pp. 36–52, 1989.

[14] Y. Saad and M. H. Schultz, "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Scientific and Statistical Computing*, vol. 7, pp. 856–869, 1986.

[15] H. van der Vorst, "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 13, no. 2, pp. 631–644, 1992.

[16] M. Olschowka and A. Neumaier, "A new pivoting strategy for gaussian elimination," *Linear Algebra and its Applications*, vol. 240, pp. 131–151, 1996.

[17] I. S. Duff and J. Koster, "On algorithms for permuting large entries to the diagonal of a sparse matrix," *SIAM J. Matrix Analysis and Applications*, vol. 22, no. 4, pp. 973–996, 2001.

[18] A. Gupta and L. Ying, "On algorithms for finding maximum matchings in bipartite graphs," IBM T. J. Watson Research Center, Yorktown Heights, NY, Tech. Rep. RC 21576 (97320), October 25, 1999.

[19] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li, "Analysis and comparison of two general sparse solvers for distributed memory computers," *ACM Trans. Math. Softw.*, vol. 27, no. 4, pp. 338–421, 2002.

[20] M. Benzi, J. C. Haws, and M. Tuma, "Preconditioning highly indefinite and nonsymmetric matrices," *SIAM J. Scientific Computing*, vol. 22, no. 4, pp. 1333–1353, 2000.

[21] A. Gupta and L. Ying, "A fast maximum-weight-bipartite-matching algorithm for reducing pivoting in sparse gaussian elimination," IBM T. J. Watson Research Center, Yorktown Heights, NY, Tech. Rep. RC 21576 (97320), October 1999.

[22] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[23] C. Ashcraft and R. Grimes, "The influence of relaxed supernode partitions on the multifrontal method," *ACM Transactions on Mathematical Software*, vol. 15, no. 4, pp. 291–309, 1989.

[24] Y. Saad, *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, 1996.

[25] M. Grote and T. Huckle, "Parallel Preconditioning with Sparse Approximate Inverses," *SIAM Journal on Scientific Computing*, vol. 18, pp. 838–853, 1997.

[26] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 24th national conference of the ACM*. ACM, 1969.

[27] T. Davis, *University of Florida Sparse Matrix Collection*, University of Florida, Gainesville, http://www.cise.ufl.edu/davis/sparse/.

[28] M. Benzi, W. Joubert, and G. Mateescu, "Numerical experiments with parallel orderings for ILU preconditioners," *Elect. Trans. Numer. Anal.*, vol. 8, pp. 88–114, 1998. [Online]. Available: citeseer.nj.nec.com/benzi99numerical.html

Olaf Schenk was born in Wilhelmshaven, Germany, in 1967. He received the M.S. degree in applied mathematics and computer science from the University of Karlsruhe, Germany in 1996 and the Ph.D. degree in technical sciences from the Swiss Federal Institute of Technology (ETH), Zürich, Switzerland in 2000. His thesis research concerned the scalable parallel direct solution of large sparse unsymmetric systems of linear equations.

Since 2001 he is a Research Associate at the Computer Science Department of the University in Basel, Switzerland. In the second half of 2002 he has been an academic visiting scientist at the Mathematical Science Department, IBM T.J. Watson Research Center, Yorktown Heights, NY. His current areas of research interests include scientific computing, numerical linear algebra, multilevel methods, optimization, computational grids and its applications to engineering problems.

Stefan Röllin was born in 1974 in Baar, Switzerland. He studied mathematics at the Swiss Federal Institute of Technology (ETH), Zürich, Switzerland. After his studies, in April 2000, he joined the Integrated Systems Laboratory at ETH, where he is working for a doctoral degree. His main research interests are iterative solvers for linear systems from semiconductor device simulation and their parallelization on shared memory multiprocessor machines.

Anshul Gupta was born in 1966 in New Delhi, India. He received a B.Tech. degree from the Indian Institute of Technology, New Delhi, in 1988 and a Ph.D. from the University of Minnesota in 1995, both in Computer Science. He is currently a research staff member at IBM T.J. Watson Research Center, Yorktown Heights, NY. His research interests include parallel algorithms, sparse matrix computations, and applications of parallel processing in scientific computing and optimization.