Computer Physics Communications 67 (1991) 145–156
North-Holland

On the numerical solution of the three-dimensional semiconductor device equations on vector-concurrent computers

S. Selberherr, M. Stiftinger, O. Heinreichsberger and K.P. Traar

Institute for Microelectronics, Technical University Vienna, Vienna, Austria
and
SIEMENS AG Austria, Vienna, Austria

The discretization of the semiconductor equations in three-dimensional device simulators leads to very large sparse linear systems of equations. While the solution of the Poisson equation – e.g. by the conjugate-gradient method – is straightforward, the iterative solution of the carrier continuity equations is nontrivial due to both the nonsymmetry and the poor conditioning of the coefficient matrices. As a consequence we have investigated conjugate-gradient-like iterative methods such as conjugate gradients applied to the normal equations, a symmetrized conjugate-gradient method, ORTHOMIN, GMRES and three squared biconjugate-gradient algorithms. All these methods were implemented in conjunction with incomplete factorization preconditioners, since the large condition number of the coefficient matrices makes preconditioning indispensable. We demonstrate the effectiveness of our implementation on vector and vector-concurrent supercomputers, such as the Fujitsu VP200, Cray-2, and on minisupercomputers, such as the ALLIANT/FX40 and VAX 6260.

1. Introduction

In this paper the computational "number-crunching" aspect of the numerical solution of the three-dimensional stationary semiconductor equations on a rectangular domain is considered. This nonlinear boundary value problem (BVP) is usually tackled by either a damped (and possibly inexact) Newton technique [3,6], or by the well-known Gummel algorithm [8], a nonlinear block Gauss–Seidel scheme. While the first method exhibits superlinear local convergence, the latter proves superior stability independent from a possibly poor initial guess. No Jacobi matrix is required thus keeping memory requirements low contrary to the Newton method. The locally linear convergence of Gummel's algorithm, often considered as its major drawback, can be improved by nonlinear convergence acceleration [15]. In the following we shall deal exclusively with the latter method.

The decoupled and appropriately discretized equations produce very large sparse systems of equations the solution time of which dominates program execution. In the case of Gummel's algorithm the discretization of the linearized Poisson equation for the electrostatic potential ψ leads to a symmetric positive-definite coefficient matrix, which, under favourable assumptions on the grid, is well conditioned.

For the carrier continuity equations, however, the situation is different (see e.g. ref. [2]). The coefficient matrices of the discretized carrier continuity equations are nonsymmetric due to the usual exponential (Scharfetter–Gummel) interpolation scheme. Unlike the Poisson equation the continuity equation can be conditioned poorly, especially in the case of high terminal bias including substantial impact ionization.

Iterative solvers for the linear systems appear to be the most suitable solution methods for linear systems in the three-dimensional case. The solution of the Poisson equation by the preconditioned conjugate-gradient method is straightforward [16]. For the nonsymmetric continuity equations rapidly convergent and numerically stable solvers are sought. Conjugate-gradient-like methods turn out to be the most suitable choice [14].

Convergence (and thus reliability) of the applied iterative method depends quite critically on the preconditioner. The high quality of preconditioners based upon incomplete LU factorization (ILU) is firmly established in numerical analysis, thus we used adaptive fill-in ILU preconditioning throughout our investigations.

The recursion operations in the backsubstitution process of the triangular L, U factors constitute the main computational bottleneck on a vector computer. We have implemented the triangular solvers using a list-vector technique resulting in a 130 megaflop throughput for the triangular solves on the VP200 supercomputer. The main result of this paper is that the execution time of our three-dimensional device simulator on a vector-supercomputer is comparable to the execution time of the two-dimensional program version on a common minicomputer or workstation.

The paper is organized as follows: section 2 gives a brief overview of the basic partial-differential equations and section 3 treats the discretization and the iterative solution of the nonlinear system of equations. Section 4 deals with some important algebraic properties of the nonsymmetric coefficient matrices. In section 5 a number of methods for the iterative solution of nonsymmetric linear systems are reviewed and compared against some new iterative methods. Section 6 gives theoretical and section 7 implementational notes on the used preconditioners. Section 8 concludes with numerical results.

2. The basic partial differential equations

We consider the time-invariant case of the semiconductor equations only. In the transient case additional terms increase the diagonal dominance of the discretized equations and therefore improve their condition number, if an implicit backward time-difference scheme is used. The semiconductor equations [23] in the variables (ψ, n, p) consist of the Poisson equation and the carrier continuity equations. Poisson's equation for the electrostatic potential ψ reads

$$\operatorname{div}(\epsilon \operatorname{grad} \psi) = -\rho, \tag{1}$$

with the space charge $\rho = q(p - n + C)$, where C denotes the net doping concentration, n the hole, p the electron concentrations and q the elementary charge. The carrier continuity equations for the electron and hole current densities $\boldsymbol{J}_{n,p}$ read

$$\operatorname{div} \boldsymbol{J}_n = qR, \tag{2}$$

$$\operatorname{div} \boldsymbol{J}_p = -qR, \tag{3}$$

where R denotes the carrier generation and recombination rate.

The current densities $\boldsymbol{J}_{n,p}$,

$$\boldsymbol{J}_n = \mu_n n \boldsymbol{F}_n, \tag{4}$$

$$\boldsymbol{J}_p = \mu_p p \boldsymbol{F}_p, \tag{5}$$

are assumed to be proportional to the driving forces $\boldsymbol{F}_{n,p}$, proportionality being determined by the carrier mobilities $\mu_{n,p}$. Solid-state thermodynamical statistics confirm the applicability of an extended drift-diffusion approach for the driving forces [11]

$$\boldsymbol{F}_n = -q\left(\operatorname{grad} \psi - \frac{1}{n}\operatorname{grad}\left(\frac{kT_n}{q} n\right)\right), \tag{6}$$

$$\boldsymbol{F}_p = -q\left(\operatorname{grad} \psi + \frac{1}{p}\operatorname{grad}\left(\frac{kT_p}{q} p\right)\right), \tag{7}$$

where carrier heating is modeled by carrier temperature $T_{n,p}$. Approximations for the carrier temperatures $T_{n,p}$ can be derived by a series expansion of the energy conservation equation

$$T_{n,p} = T_0 + \frac{2}{3}\frac{q}{k}\tau^{\epsilon}_{n,p}\left(v^{\text{sat}}_{n,p}\right)^2\left(\frac{1}{\mu^{\text{LISF}}_{n,p}} - \frac{1}{\mu^{\text{LIS}}_{n,p}}\right), \tag{8}$$

with the Boltzmann constant k, the ambient temperature T_0 and the energy relaxation times $\tau^{\epsilon}_{n,p}$. The superscripts of $\mu_{n,p}$ denote the mobilities due to lattice (L), impurity (I), and surface (S) scattering and their decrease due to carrier heating (F).

The carrier generation and recombination rate R on the right-hand side of the carrier continuity equations represents the sum of the impact ionization rate R^{II}, the Shockley–Read–Hall recombi-

nation rate R^{SRH} and the Auger recombination rate R^{AU},

$$R = R^{II} + R^{SRH} + R^{AU}. \tag{9}$$

3. Iterative solution of the nonlinear system of equations

We use finite-difference discretization in a rectangular spatial domain to treat the semiconductor equations numerically. At the idealized Ohmic terminal contacts Dirichlet boundary conditions hold, at artificial interfaces in the deep bulk homogenous Neumann boundary conditions have to be applied. Inhomogenous Neumann boundary conditions are valid in case of interface charges for the electrostatic potential and in case of nonvanishing interface recombination velocity for the carrier concentrations.

The nonlinearity of the discretized coupled system of equations can be treated in different ways: classical Newton or quasi Newton schemes make a simultaneous solution of the three semiconductor equations necessary and have a locally quadratic convergence behavior. We restrict ourselves to the block iterative Gummel algorithm [8], which allows a sequential solution of the three equations within one outer iteration. It is less sensitive to the initial guess than Newton methods but has the disadvantage of even sublinear convergence behavior in the case of high-current simulations. The nonlinear Gummel modification of Poisson's equation is linearized by a first-order series expansion. The nonlinearities in the carrier mobilities and carrier temperatures are neglected, the derivatives of R^{II} with respect to n and p are updated in a superimposed impact ionization *subcycle* and can therefore be neglected, too. Both $(\partial R^{SRH}/\partial n)$ and $(\partial R^{SRH}/\partial p)$ increase the diagonal dominance in the resulting coefficient matrices and are therefore taken into account. $(\partial R^{AU}/\partial n)$ and $(\partial R^{AU}/\partial p)$ do not necessarily have such a stabilizing effect. Negative contributions to the main diagonal of the coefficient matrices are therefore discarded. The resulting scheme reads

$$\text{div grad } \psi^{k+1} = -\frac{q}{\epsilon}\left(\frac{\partial (p-n+C)^k}{\partial \psi} \times (\psi^{k+1}-\psi^k) + p^k - n^k + C\right), \tag{10}$$

$$\text{div } J_p(\psi^{k+1}, p^{k+1}) = -q\left(R(\psi^{k+1}, n^k, p^{k+1}) + \frac{\partial R}{\partial p}(p^{k+1}-p^k)\right), \tag{11}$$

$$\text{div } J_n(\psi^{k+1}, n^{k+1}) = q\left(R(\psi^{k+1}, n^{k+1}, p^{k+1}) + \frac{\partial R}{\partial n}(n^{k+1}-n^k)\right). \tag{12}$$

In order to cope with the exponential dependence of the carrier densities on the electrostatic potential and in order to allow carrier temperature dependent mobilities, a modified version of the Scharfetter–Gummel interpolation scheme [22] for the carrier concentrations is used. Nonplanar interfaces are discretized by the well-known box integration method.

4. Algebraic properties of the coefficient matrices

The linear interpolation of the electrostatic potential between adjacent gridlines and the modification by Gummel's algorithm (using natural ordering) leads to a symmetric, positive definite, 2-cyclic coefficient matrix of the discretized Poisson equation. The solution can easily be achieved by the standard preconditioned conjugate-gradient algorithm. The exponential Scharfetter–Gummel interpolation scheme in the discretization of the carrier continuity equations produces nonsymmetric, 2-cyclic coefficient matrices A. The diagonal dominance is guaranteed by the derivatives of Shockley–Reed–Hall and Auger recombination with respect to the carrier concentrations. In the absence of recombination terms the main diagonal elements equal the negative column sum of the offdiagonal elements. This implies at least semidefiniteness of A.

From the exponential interpolation scheme it

can easily be seen, that A can be transformed to a symmetric, positive definite matrix $\bar{A}$ [4],

$$\bar{A} = W_{n,p} A W_{n,p}^{-1}, \tag{13}$$

by a diagonal matrix W with positive elements w_i. The w_i are given by

$$w_{i,n} = \exp\left(-\frac{\psi_i}{2U_t}\right), \qquad w_{i,p} = \exp\left(\frac{\psi_i}{2U_t}\right), \tag{14}$$

for electrons and holes. $U_t = kT/q$ denotes the thermal voltage. The enormous number range of the $w_{i,n,p}$ inhibits an explicit symmetrization and a solution of the resulting symmetric linear system by conjugate gradients. For a maximum electrostatic potential of 5 volts and a minimum temperature of 77 K (liquid nitrogen temperature) exponents of the order $\log_{10}(w_{i,\max}) = 164$ arise.

The symmetrizability guarantees a positive real spectrum of A. Therefore iterative methods are safely applicable.

5. Selected iterative methods for the linear systems

We have selected some basic projection-type iterative methods [20] for the nonsymmetric linear systems. In the following section some theoretical aspects and the practical applicability will be discussed. All of these methods are preconditioned by incomplete LU factorization.

5.1. CGNR

This algorithm applies conjugate gradients to the normal equations. It solves the symmetric, positive definite problem

$$A^{\mathrm{T}}Ax = A^{\mathrm{T}}b \tag{15}$$

by the classical conjugate-gradient algorithm. It is clear, that the matrix-product $A^{\mathrm{T}}A$ is never built explicitly. It constructs a unique sequence of vectors

$$x_k \in x_0 + \langle A^{\mathrm{T}}r_0, (A^{\mathrm{T}}A)A^{\mathrm{T}}r_0, \ldots, (A^{\mathrm{T}}A)^{k-1}A^{\mathrm{T}}r_0\rangle, \tag{16}$$

where $\| r_k \|_2 = \min$, which is equal to the orthogonality condition

$$r_k \perp \langle AA^{\mathrm{T}}r_0, (AA^{\mathrm{T}})^2 r_0, \ldots, (AA^{\mathrm{T}})^k r_0\rangle. \tag{17}$$

Because of this minimization property, the convergence is strictly monotonic but it is determined by the *squares* of the singular values of A [17]. Therefore it can be expected that the convergence behavior is rather poor. Numerical experiments confirm, that this algorithm cannot satisfy our requirements.

5.2. Symmetrized CG

The similarity of the coefficient matrices to symmetric, positive-definite (SPD) matrices can be exploited by a symmetrized conjugate-gradient method, which avoids the explicit symmetrization of A [12]. The cumbersome symmetrization matrix W is only required for the computation of the iteration parameters, where it appears in the numerator and denominator. This allows scaling in order to avoid floating-point under- or overflow. This algorithm constructs a unique sequence of vectors

$$x_k \in x_0 + \langle r_0, Ar_0, \ldots, A^{k-1}r_0\rangle. \tag{18}$$

The following orthogonality condition for the residuals holds:

$$r_k \perp \langle W^2 r_0, W^2 A r_0, \ldots, W^2 A^{k-1} r_0\rangle. \tag{19}$$

This algorithm minimizes the $[W^2A]^{1/2}$-norm of the error vector. In our applications this algorithm can only be applied in low-voltage simulations, for which the number range of the symmetrization matrix W fits into the number range of the computer used.

5.3. ORTHOMIN

This algorithm [20] constructs a unique sequence of vectors

$$x_k \in x_0 + \langle r_0, Ar_0, \ldots, A^{k-1}r_0\rangle \tag{20}$$

so that $\| r_k \|_2 = \min$. Therefore the following orthogonality condition for the residuals holds:

$$r_k \perp \langle Ar_0, A^2 r_0, \ldots, A^k r_0\rangle. \tag{21}$$

ORTHOMIN converges monotonically. The minimization property of this algorithm is guaranteed by explicit A^2-orthogonalization of the kth search direction p_k to all previous search directions p_0, $p_1, \ldots, p_{k-1}$. This requires the storage of all previous search directions and the computation of k inner products per iteration. This is not possible in practice, so we use a *truncated* version of the ORTHOMIN algorithm. Only the last m (where m is a constant) search-direction vectors are kept. This restricts the storage requirements and the arithmetic work per iteration, but the ORTHOMIN algorithm loses its optimality properties. About the truncated version few theoretical results are available.

5.4. GMRES

Similar to ORTHOMIN this algorithm [21] minimizes $\| r_k \|_2$. x_k belongs to the same *Krylov* subspace as with ORTHOMIN, but a unique sequence of vectors

$$x_k \in x_0 + \langle r_0, Ar_0, \ldots, A^{k-1} r_0 \rangle \tag{22}$$

is built my means of an Arnoldi construction of an orthogonal basis. Assuming that the procedure has converged in k steps ($\| r_k \|_2 < \epsilon$), a $(k+1) \times k$ upper Hessenberg least squares problem has to be solved, so that x_k is the best approximation to the true solution.

The minimization property of GMRES is equal to the orthogonality condition

$$r_k \perp \langle Ar_0, A^2 r_0, \ldots, A^k r_0 \rangle \tag{23}$$

and guarantees monotonic convergence.

Full orthogonalization requires the storage of k "back"-vectors and the calculation of $k+1$ inner products at the kth iteration. To limit storage requirements we use a *restarted* version of GMRES. The algorithm is restarted after m (where m is a constant) iterations, if it has not yet converged. The approximation to the solution at the end of every m iterations is used as initial guess for the next m iterations.

The least-squares problem is solved by QR factorization using Householder transformation. The QR factorization of the upper Hessenberg matrix in addition provides the residual for no extra cost. As the QR factorization is updated for every iteration, the residual is always known and the algorithm can immediately be stopped when the required accuracy is reached.

The truncated version of course loses optimality, but restarting preserves monotonicity in the residual-norm.

5.5 BIOMIN^2 (CGS), BIORES^2, BIODIR^2

They are built by squaring the Lanczos biorthogonalization algorithms BIOMIN, BIORES and BIODIR [10,24]. From the computational point of view they are more efficient than the original procedures. BIOMIN^2 (CGS) and BIORES^2 do not involve the matrix-vector multiplication $A^T v$. This can be important, if the matrix-elements are stored in a general data structure. All three squared algorithms produce the same iterates for the same initial guess (as do the nonsquared ones).

In contrast to ORTHOMIN and GMRES the storage requirements of BIOMIN^2 (CGS), BIORES^2 and BIODIR^2 (also of CGNR and of the symmetrized CG) do not increase during the iteration process.

The x_k belong to the Krylov subspace

$$x_k \in x_0 + \langle r_0, Ar_0, \ldots, A^{k-1} r_0 \rangle. \tag{24}$$

For the residuals a *biorthogonality* condition holds ($\hat{r}_0$ denotes the initial residual for the "transposed" system):

$$r_k \perp \langle \tilde{r}_0, A^T \tilde{r}_0, \ldots, (A^T)^{k-1} \tilde{r}_0 \rangle. \tag{25}$$

The biorthogonalization algorithms have *no* minimization properties. Therefore the residual does not decrease monotonically. Very often an erratic convergence behavior can be observed. This may increase the influence of roundoff errors, an effect which has always to be kept in mind because of the enormous number range of the coefficient matrices of the linear systems we have to deal with.

The biorthogonalization algorithms may break down by division by zero, if certain inner products vanish. A breakdown is likely to occur, if r_0 and/or

$\tilde{r}_0$ are chosen inappropriately, but was never observed in our device simulations.

A comparison of the squared Lanczos algorithms shows, that BIOMIN^2 (CGS) needs less computational work per iteration than BIORES^2 and BIODIR^2. Roundoff errors cause the iterates of the three biorthogonalization algorithms to differ from each other during the iteration process. BIOMIN^2 (CGS) proves to be numerically most stable as illustrated by an example in section 8.

A comparison of all tested algorithms has shown that BIOMIN^2 (CGS) performs best for our applications in the sense of minimizing the overall computational work and storage requirements. This is remarkable, because it is out of the class of biorthogonalization algorithms which, in contrary to the other tested methods, have no minimization property.

6. Preconditioning

Due to the large condition number of the coefficient matrices of the discretized continuity equations efficient preconditioning is necessary in order to guarantee fast and reliable convergence of the chosen iterative methods for *all* of our different simulation problems. We are searching for easily invertible approximations to the matrix A which allow a transformation of the linear system $Ax = b$ to the system $B\tilde{x} = \tilde{b}$ with superior spectral properties. The matrix B is of course never formed explicitly. In addition to every matrix vector multiplication Av, the linear system $P^{-1}w$ has to be solved where $w = Av$ and P is the preconditioning matrix. This system must be solvable much easier than $A^{-1}b$.

We have implemented a block Jacobi preconditioner

$$P_J = D, \tag{26}$$

where D is the tridiagonal part of A, and an incomplete LU (ILU) factorization preconditioner [16]

$$P_{ILU} = P_L P_R = (\tilde{L} + \tilde{D})\tilde{D}^{-1/2}\tilde{D}^{-1/2}(\tilde{U} + \tilde{D}), \tag{27}$$

where $\tilde{L}$ and $\tilde{U}$ are strictly lower and upper triangular matrices and $\tilde{D}$ is a diagonal matrix. In the symmetric case $\tilde{L}^T = \tilde{U}$ holds and the preconditioner is the usual incomplete Cholesky (IC) decomposition preconditioner.

For the block Jacobi preconditioner left-preconditioning has been chosen,

$$P_J^{-1}Ax \equiv B_J x = P_J^{-1}b \equiv \tilde{b}_J; \tag{28}$$

for the ILU preconditioners left- and split-preconditioning has been implemented,

$$P_R^{-1}P_L^{-1}Ax \equiv B_{ILU_{left}}x = P_R^{-1}P_L^{-1}b \equiv \tilde{b}_{ILU_{left}}, \tag{29}$$

$$P_L^{-1}AP_R^{-1}P_R x \equiv B_{ILU_{split}}\tilde{x} = P_L^{-1}b \equiv \tilde{b}_{ILU_{split}}. \tag{30}$$

For the split ILU preconditioner x has to be unscaled at the end of the iterative process: $x = P_R^{-1}\tilde{x}$.

It can be shown that stationary iterative methods based on the ILU splitting converge at least as fast as methods based on a block Jacobi splitting. It can be expected that the accelerators we have discussed in section 5 behave in a similar way. On the other hand the computational work for the block Jacobi preconditioner is smaller and as there are only first-order recurrences, vectorization and/or parallelization is more straightforward than for the ILU preconditioners. But there are a few disadvantages: as can be expected from theory the block Jacobi preconditioner causes worse convergence behavior for all tested accelerators than ILU preconditioners. For high bias simulations where the drift term dominates in the current relations (6) and (7), unpleasant numerical effects can be observed, such as convergence stagnation or near-breakdown in the Lanczos process at the beginning of the iteration. Another point of distress is the sensitivity of the block Jacobi preconditioner to the orientation of x, y, and z in the three-dimensional simulation domain. "Swapping" the coefficient matrix in the most favourable direction causes computational overhead. The main disadvantage is that this direction is not the same for all our examples and sometimes changes during the outer iteration process of one example. As an efficient algorithm to detect the best preconditioning direction could not be established, we do

not recommend this type of preconditioner for general use.

The second class of preconditioners we tested extensively are the incomplete LU preconditioners with allowable fill-in denoted by ILU(k) [1,5,13]. For $k = 0$ the matrices $\tilde{L}$ and $\tilde{U}$ are equal to the strictly lower and upper parts L and U of A. The sparsity pattern of the triangular factors $(\tilde{L} + \tilde{D}) + \tilde{D}^{-1} + (\tilde{U} + \tilde{D})$, therefore, is the same as for A. For an exact LU factorization of A the sparsity pattern of the triangular factors would result in banded matrices where the bandwidth is given by the distance NX × NY of the outermost diagonals (denoting the coupling between the planes) from the main diagonal in the coefficient matrix A (NX, NY and NZ denote the number of gridlines in x-, y-, and z-direction). For $k = 1$ the fill-in caused by the ILU(0) nonzero pattern is taken into account, for $k = 2$ the fill-in caused by the ILU(1) sparsity pattern and so on (ILU(NX × NY) would be the exact LU factorization of A). It is easy to see that a higher degree of fill-in reduces the number of iterations which are necessary to solve the linear system, but increases the work per iteration and the storage requirements.

For the incomplete LU preconditioner $\tilde{D}$ can be computed such that $\mathrm{diag}(P_{\mathrm{ILU}}) = \mathrm{diag}\ (A)$ or alternatively such that $P_{\mathrm{ILU}} - A$ has zero *column* sums, which leads to modified incomplete factorization-type preconditioners (MILU) originated by Gustafsson [9] for Poisson type equations (in the symmetric case this is equal to rowsum $(P_{\mathrm{IC}}) =$ rowsum (A)). For the symmetric and positive definite Poisson equation, the magnitude of the main diagonal element is greater or equal than the sum of the offdiagonal elements of the same row (or column). For the coefficient matrices of the discretized carrier continuity equations, an analogous relationship for the columns holds. A modification factor α in the interval [0,1] is usually introduced to smoothly sweep between ILU and MILU factorization. Our results concerning the choice of such a modification factor do not admit a clear statement. It seems that $\alpha = 1$ always decreases the performance with respect to $\alpha = 0$ slightly. We found a number of device examples where a choice of $\alpha = 0.5$ yields a performance enhancement of about 10–30% concerning the iteration count. However, this gain is partly compensated by the higher arithmetic work for the factorization. This is rather disappointing, as for the symmetric Poisson equation a modification factor of $\alpha = 0.95$ reduces the iteration count up to 50%.

The (M)ILU(0) preconditioners can be implemented very efficiently. For the left (M)ILU(0) preconditioner we scale the coefficient matrix from the left side by $\tilde{D}$. The scaled matrix $A_{s_{\mathrm{left}}} = \tilde{D}A$ can be written as a sum of a strictly lower, a diagonal, and a strictly upper matrix: $A_{s_{\mathrm{left}}} = L_{s_{\mathrm{left}}} + D_{s_{\mathrm{left}}} + U_{s_{\mathrm{left}}}$. Using this scaling the matrix vector product $B_{\mathrm{ILU}_{\mathrm{left}}} v$ can be simplified to

$$B_{\mathrm{ILU}_{\mathrm{left}}} v = \left(I + U_{s_{\mathrm{left}}}\right)^{-1} \Big[v + \left(I + L_{s_{\mathrm{left}}}\right)^{-1} \times \left(D_{s_{\mathrm{left}}} - I + U_{s_{\mathrm{left}}}\right) v \Big]. \tag{31}$$

An analogous simplification for the split (M)ILU(0) preconditioner is well known as Eisenstat's trick [7]. The coefficient matrix A is scaled symmetrically by $\tilde{D}^{1/2}$: $A_{s_{\mathrm{split}}} = \tilde{D}^{1/2} A \tilde{D}^{1/2}$. Then $B_{\mathrm{ILU}_{\mathrm{split}}} v$ can also be written as

$$B_{\mathrm{ILU}_{\mathrm{split}}} v = \Big[t + \left(I + L_{s_{\mathrm{split}}}\right)^{-1} \left(v - \left(2I - D_{s_{\mathrm{split}}}\right) t \right) \Big], \tag{32}$$

with

$$t = \left(I + U_{s_{\mathrm{split}}}\right)^{-1} v. \tag{33}$$

The split (M)ILU(0) preconditioner requires two additional scratch vectors, but only scalar-vector multiplications and vector updates have to be performed, whereas the left (M)ILU(0) needs no extra storage but a triangular matrix vector multiply. We are not aware of analogous tricks for higher fill-in preconditioners.

There are a number of other preconditioners such as the SSOR [1], least-squares polynomial, Neumann polynomial and their line and/or block variants. Numerical experiments carried out with the NSPCG software package [18] identified none of them competitive with ILU.

7. Implementation of ILU and IC preconditioners on vector and vector-concurrent computers

Since the solution of the linear systems consumes most of the overall CPU-time in a device

simulator, it is important to implement the linear solvers as efficiently as possible. Using the iterative methods we have discussed in section 5, all vector operations except for the backsolves of triangular systems that are introduced by the incomplete LU (or Cholesky) factorization preconditioners can be vectorized and/or parallelized straightforwardly. Our goal was to find an implementation of those backsubstitutions for vector computers which is as generally applicable as possible and efficient on most of the modern vector and vector-concurrent computers. On the other hand we concentrated on vectorization techniques that are unlikely to degrade the performance of the incomplete factorization preconditioners, hence we do not consider multicolor orderings [19]. We further excluded computers that permit only unity-stride vector operations such as the CYBER 205 and did not care of decrease of the performance due to possible memory bank conflicts on some machines. As for some vector computers good performance can only be achieved for long vector length this was also an aim developing our vectorizable linear solver code.

A data dependence analysis shows that there exist diagonal planes, so-called *hyperplanes* or *computational wavefronts* H_m in which the unknowns are only dependent on the unknowns in the hyperplane H_{m-1} for the lower triangular matrix or on those in the hyperplane H_{m+1} for the upper triangular system [1,25]. If the matrix elements are denoted by the indices (i1, i2, i3) in the three spatial directions, the equation

$$\mathrm{i1} + (k+1)(\mathrm{i2} + \mathrm{i3}) = m \tag{34}$$

is valid for the indices of all mesh points in the hyperplane H_m. k denotes the degree of fill-in, m is a constant. This allows an independent calculation of all unknowns in one hyperplane. The hyperplanes of course have to be treated sequentially. It is obvious that the number of points in one hyperplane first increases from one up to $\mathcal{O}$ (NX × NZ)/2 and then decreases to one provided that NY > MAX(NX, NZ) (otherwise permute the unknowns appropriately). The main problem lies in the fact that the unknowns and the matrix entries have either to be reordered or addressed indirectly. As explicitly reordering the matrix entries requires a great amount of additional storage for higher fill-in preconditioners (the addressing of the offdiagonal elements is also rather complicated or has to be done indirectly), we chose the second approach and compute a list vector **MASK** which the addresses of the unknowns in hyperplane-ordering are written in at the begin of the iteration process. A second list vector **LIST** contains the addresses of the elements of the first list vector at which new hyperplanes begin. This leads to the following source code for the backward substitution of the lower triangular preconditioning matrix of the ILU(0) preconditioner (the main diagonal is unity):

```
      X(1)=R(1)
      DO 1 L=2,NX+NY+NZ-2
      DO 1 M=LIST(L-1)+1,LIST(L)
      I=MASK(M)
1     X(I)=R(I)-B(I)*X(I-1)-D(I)
           *X(I-NX)-F(I)*X(I-NX*NY)
```

R denotes the right-hand side and **B**, **D**, **F** the strictly lower-triangular part of A_s. The inner loop is vectorizable. For the (M)ILU(1) preconditioner the upper bound of the outer loop equals **NX+2*(NY+NZ-2)** and the calculation of **X(I)** requires taking into account the fill-in terms **-D1(L)*X(I-NX+1)** and **-F1(L)*X(L-NX*NY+1)**. Analogously the upper bound for the (M)ILU(2) preconditioner is **NX+3*(NY+NZ-2)** and the additional terms **-D2(L)*X(L-NX+2)** and **-F2(L)*X(L-NX*NY+2)** must be taken into account. The above described approach is used in a similar manner to vectorize the ILU factorization at the beginning of the iteration. A vectorization of the MILU preconditioner by this approach is only possible for no fill-in ($k = 0$) preconditioners.

8. Numerical results and conclusions

At first the convergence of several iterative methods which have been treated in section 5 is examined. As test matrix serves the coefficient matrix of the electron continuity equation of the first Gummel-iteration of an n-channel silicon MOSFET (1.5 micron channel length) from the

device simulator MINIMOS 5. Bias conditions are: $U_S = U_B = 0$ V (source, bulk), $U_G = 1$ V (gate) and $U_D = 3$ V (drain). The mesh size is $29 \times 31 \times 35$ in x, y and z direction. As error measure the maximum norm of the error vector $e_n = \| \bar{x} - x_n \|_\infty$ is used. $\bar{x}$ denotes the solution vector obtained by Gaussian elimination (this explaines the small grid size of the chosen example). The tests were carried out on a Digital VAX 8800 in double precision arithmetic with precision of 1.0 of 1.387×10^{-17}.

Figure 1 shows the convergence curves. CGNR and GMRES(2) are certainly not competitive. The convergence behavior of the symmetrized CG, SYMCG, is not satisfactory, too. ORTHOMIN(5) and especially GMRES(5) are save, but slow alternatives to BIOMIN^2 (CGS). The advantage of GMRES(5) is the monotonic convergence behavior. But in all our examples BIOMIN^2 (CGS) has proved to be the best choice. This is clear if the arithmetic work per iteration for the tested accelerators, which is shown in table 1, is taken into account. A matrix-vector multiplication is the most time-consuming, a scalar-vector multiplication is the least time-consuming operation.

In the next figure we want to compare the robustness of the three squared biorthogonalization algorithms BIOMIN^2 (CGS), BIORES^2 and BIODIR^2 against roundoff errors. The same test problem as before is used and the convergence curves starting from IT = 100 are displayed in fig. 2. Both BIOMIN^2 (CGS) and BIORES^2 stagnate at a minimum error, however BIODIR^2 diverges after having run through an error minimum. Obviously BIOMIN^2 (CGS) is the most stable algorithm.

Table 1
Comparison of arithmetic work

Solver	Ax	(x, y)	$x+y$	αx
CGNR	2	2	4	10
Symm. CG	1	3	4	10
ORTHOMIN(m)	3	$m+2$	$m+2$	$m+2$
GMRES(m)	1	$\frac{1}{2}m+1$	$\frac{1}{2}m$	$\frac{1}{2}m+1$
BIOMIN^2 (CGS)	2	2	7	6
BIORES^2	2	2	10	16
BIODIR^2	2	2	9	13

The ILU (k) preconditioners with levels $k = 0$, 1, 2 are compared in fig. 3. As can be seen from the nonmonotonic convergence behavior BIOMIN^2 (CGS) has been chosen as accelerator. This figure shows that a higher degree of fill-in really improves the convergence behavior.

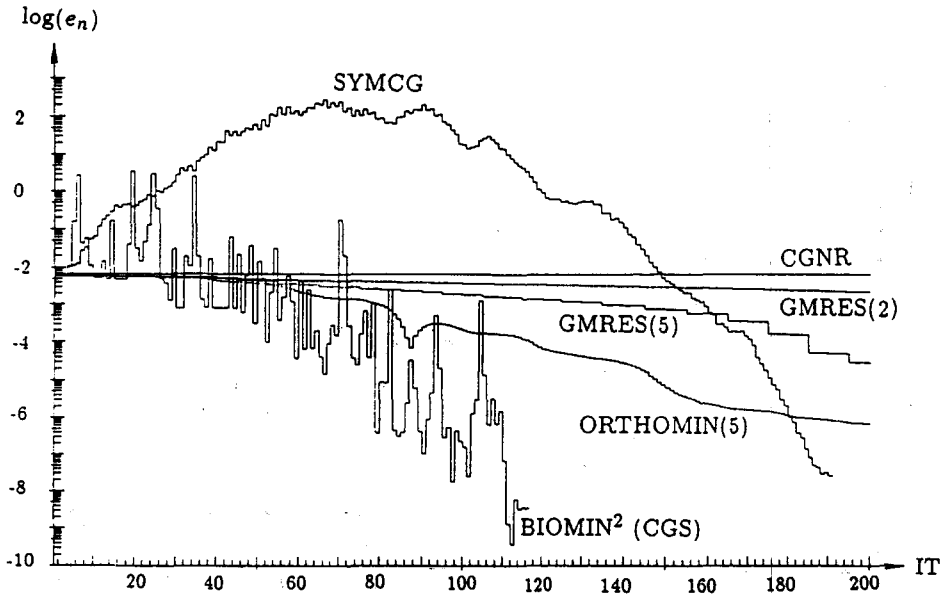

Fig. 1. Convergence of accelators.

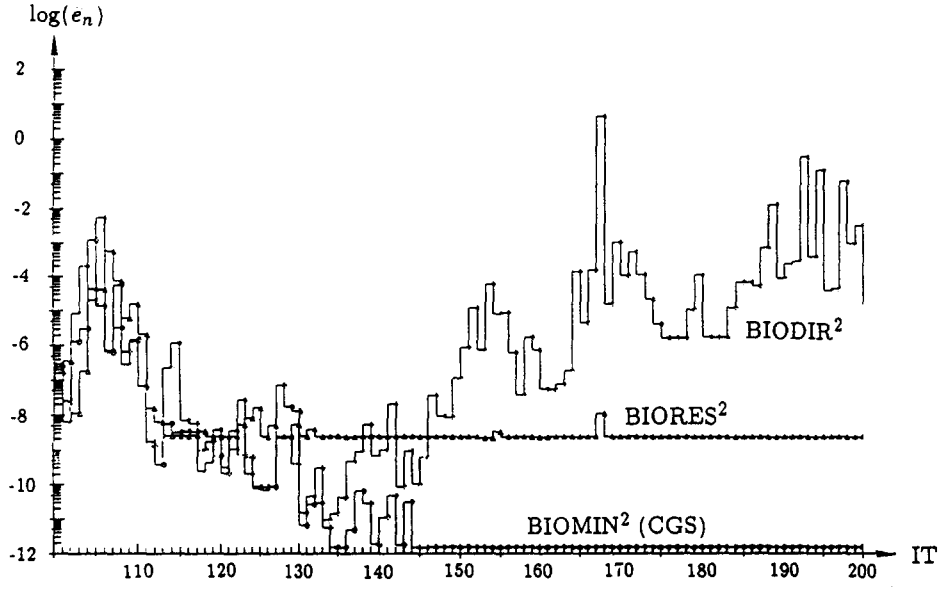

Fig. 2. Comparison of squared Lanczos methods.

Table 2 compares the performance of the hyperplane ILU(k) implementation on a Fujitsu VP200 (VP), a Cray-2 (C2) and an ALLIANT/FX40 (AL) computer. The tests have been carried out on a $40 \times 40 \times 40$ grid and the results are mean values of 100 backsubstitutions. Besides the

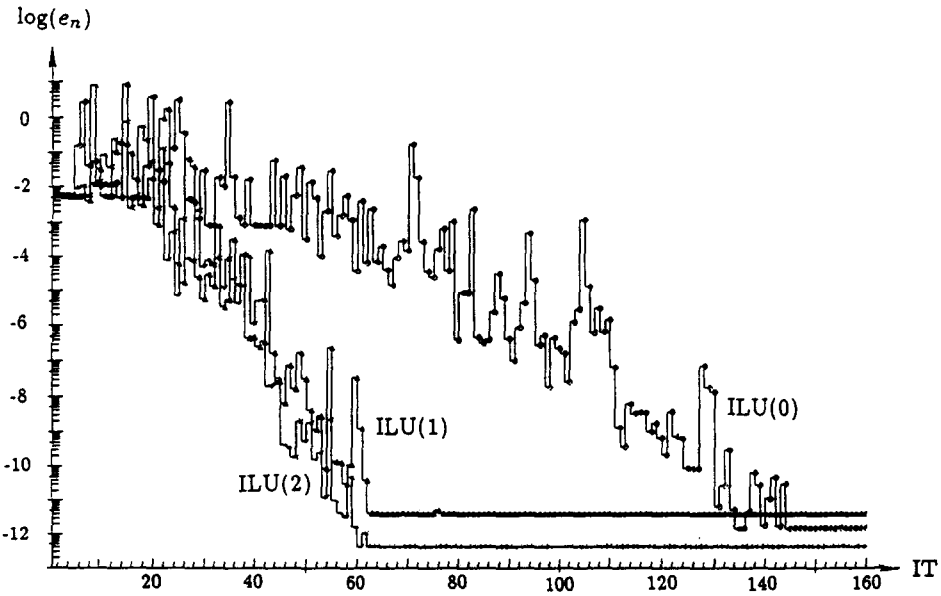

Fig. 3. Comparison of preconditioners.

Table 2
Hyperplane-ILU(k) on vectorcomputers

FACT	VP			C2			AL		
	B	S	M	B	S	M	B	S	M
ILU(0)	4	13.75	96	14	4.30	27	212	1.34	1.8
ILU(1)	8	12.12	96	26	4.76	30	327	1.51	2.3
ILU(2)	8	12.62	128	30	5.93	34	410	1.64	2.5

CPU time for one backsubstitution (B) in milliseconds (ms), the overall achieved speed-up over the trivial code in the solution (S) of the triangular systems and the megaflop (MFlop) rate (M) are presented. It seems that the code is better suitable for the VP200 than for the Cray-2. One reason is the good performance of the Fujitsu vector computer for long vector length. Possible memory-bank conflicts on the Cray due to the indirect addressing according to the hyperplane-ordering decrease the performance of the Cray supercomputer substantially.

In order to show that our code can also be parallelized to a high extent on slightly coupled multiprocessor computers, we carried out tests for the hyperplane backsubstitutions on a 6-scalar-processor Digital VAX 6260. Table 3 shows speed-ups against one processor.

Acknowledgements

This work has been supported by SIEMENS AG, Munich, and Digital Equipment Corporation, Hudson. The authors are indebted to Martin Schubert and Hans-Peter Falkenburger from the Institute of Microelectronics Stuttgart for their help on performing tests on the CRAY-2, and to Dr. Martin Thurner from the Campusbased Engineering Center Vienna (Digital Equipment Corporation) for performing measurements on the VAX 6260. We wish to thank H. Dietrich, G. Koessl and H. Wiktorin of the Computer Services of Cooperate Research and Development, SIEMENS AG, Munich, for valuable help in gaining access to the VP200.

Table 3
Parallel hyperplane ILU(0) on VAX 6260

	Processors					
	1	2	3	4	5	6
MFlops	0.58	1.15	1.64	2.06	2.41	2.65
Speedup	1.00	1.98	2.82	3.54	4.14	4.56

References

[1] C.C. Ashcraft and R.G. Grimes, On vectorizing incomplete factorization and SSOR preconditioners, SIAM J. Sci. Stat. Comput. 9 (1988) 122.
[2] U. Ascher, P.A. Markovich, C. Schmeisser, H. Steinrueck and R. Weiss, Conditioning of the steady state semiconductor problem, Tech. Rep. 86-18, Dept. of CS, Univ. of British Columbia, Vancouver (1986).
[3] R.E. Bank and D.J. Rose, Global approximate Newton methods, Numer. Math. 37 (1981) 279.
[4] R.E. Bank, D.J. Rose and W. Fichtner, Numerical methods for semiconductor device simulation, IEEE Trans. Electron Devices 30 (1983) 1031.
[5] P. Concus, G.H. Golub and G. Meurant, Block preconditioning for the conjugate gradient method, SIAM J. Sci. Stat. Comput. 6 (1985) 220.
[6] R.S. Dembo, S. Eisenstat and T. Steinhaug, Inexact Newton methods, SIAM J. Sci. Stat. Comput. 18, (1981) 400.
[7] S.C. Eisenstat, Efficient implementation of a class of preconditioned conjugate gradient methods, SIAM J. Sci. Stat. Comput. 2 (1981) 1.
[8] H.K. Gummel, A selfconsistent iterative scheme for one-dimensional steady state transistor calculations, IEEE Trans. Electron Devices 11 (1964) 455.
[9] I. Gustafsson, A class of first-order factorization methods, BIT 18 (1978) 142.
[10] M.H. Gutknecht, The unsymmetric Lanczos algorithms and their relations to Padé approximations, continued fractions and the QD algorithm, in: Preliminary Proc. Copper Mountain Conf. Iterative Methods, April 1990, Vol. 2.
[11] W. Hänsch and S. Selberherr, MINIMOS 3: a MOSFET simulator that includes energy balance, IEEE Trans. Electron Devices 34 (1987) 1074.
[12] L.A. Hageman, F.T. Luk and D.M. Young, On the equiv-

alence of certain iterative acceleration methods, SIAM J. Numer. Anal. 17 (1980) 852.
[13] K. Hane, Supercomputing for process/device simulations, in: Proc. Sixth Int. NASECODE Conf., Trinity College, Dublin, July 1989, p 11.
[14] S.J. Polak, C. Den Heijer and W.H.A. Schilders, Semiconductor device modelling from the numerical point of view, Int. J. Numer. Methods Eng. 24 (1987) 763.
[15] T. Kerkhoven and Y. Saad, Acceleration techniques for decoupling algorithms in semiconductor simulation, Tech. Rep., Dept. of Computer Science, Univ. of Illinois at Urbana–Champaign (1987).
[16] H. Meijerink and H. Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, Math. Comput. 31 (1977) 148.
[17] N.M. Nachtigal, S.C. Reddy and L.N. Trefethen, How fast are nonsymmetric iterations?, in: Preliminary Proc. Copper Mountain Conf. Iterative Methods, Vol. 4, April 1990.
[18] T.C. Oppe, W.D. Joubert and D.R. Kincaid, NSPCG User's Guide., Center of Numerical Analysis, University of Texas at Austin (1984).
[19] Y. Oyanagi, Hyperplane vs. multicolor vectorization of incomplete LU preconditioning for the Wilson fermion on the lattice, J. Inf. Process. 11 (1) (1987) 32.
[20] Y. Saad, The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems, SIAM J. Sci. Stat. Comput. 19 (1982) 485.
[21] Y. Saad and M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems., SIAM J. Sci. Stat. Comput. 7 (1986) 856.
[22] D.L. Scharfetter and H.K. Gummel, Large-signal analysis of a silicon read diode oscillator, IEEE Trans. Electron Devices 16 (1969) 64.
[23] S. Selberherr, Analysis and Simulation of Semiconductor Devices (Springer, Wien, 1984).
[24] P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric systems, SIAM J. Sci. Stat. Comput. 10 (1989) 36.
[25] H. Vorst, High performance preconditioning, SIAM J. Sci. Stat. Comput. 10 (1989) 1174.