THREE-DIMENSIONAL SIMULATION OF SEMICONDUCTOR DEVICES ON SUPERCOMPUTERS

K.P. Traar

SIEMENS AG Austria
Electronic Developement
Gudrunstrasse 11, A–1101 Wien, Austria

M. Stiftinger, O. Heinreichsberger, and S. Selberherr

Technical University Vienna
Institute for Microelectronics
Gusshausstrasse 27–29, A–1040 Wien, Austria

Abstract

For the selfconsistent solution of the three-dimensional semiconductor device equations large and sparse linear systems of equations have to be solved repeatedly. In order to obtain reliable results with modest demands on computer resources preconditioned iterative methods for the solution of the linear systems of equations are applied. Various iterative methods for the solution of the nonsymmetric discretized carrier continuity equations, e.g. conjugate gradients applied to the normal equations, a symmetrized conjugate gradient method, GMRES, and CGS, are compared. As preconditioners Jacobi and incomplete LU factorization methods have been investigated. For an efficient implementation on modern vector and vector-concurrent computers special coding techniques have to be applied in order to allow vectorization and/or parallelization of the recurrence relations in the preconditioners. Results obtained on a SIEMENS/Fujitsu VP200, a Cray-2, and on Minisupercomputers such as ALLIANT/FX40 and VAX 6260 are presented.

1 INTRODUCTION

The numerical analysis of semiconductor devices [24] has become an important tool as well for design engineers to predict the device performance as for device physicists to understand the internal behavior of the device. Whereas in the eighties two-dimensional simulators were able to fulfill these demands, shrinking device dimensions have led to submicron devices, which can only be described satisfactorily by three-dimensional simulation.

In order to be able to make three-dimensional device simulators economically applicable, answers have to be obtained within a time comparable to two-dimensional simulations a few years ago. This can be achieved by running those programs on modern supercomputers. But also a lot of software development in the field of numerical analysis has to be done, which was not so important for two-dimensional device simulators.

Therefore much effort has been undertaken to develop fast solvers for the sparse and large linear systems of equations, when for our MOS-/MESFET device simulator MINIMOS [25] a three-dimensional version was developed. Iterative solvers for the linear systems appear to be the most suitable solution methods in the three-dimensional case. The solution of the symmetric, positive definite, and well conditioned Poisson equation by the preconditioned conjugate gradient method is straightforward [16]. For the nonsymmetric and commonly bad conditioned [2] continuity equations rapidly convergent and numerically stable solvers are sought. Conjugate gradient-like methods turn out to be the most suitable choice [19]. As there are only few theoretical results concerning the convergence behavior of iterative methods for nonsymmetric linear systems of equations available, we have compared some important algorithms.

Convergence (and thus reliability) of the applied iterative method depends quite critically on the preconditioner. The Jacobi preconditioner has recently again become a matter of interest, because it can be vectorized and parallelized rather easily. The high quality of preconditioners based upon incomplete LU factorization (ILU) is firmly established in numerical analysis, thus we also concentrated on adaptive fill-in ILU preconditioning throughout our investigations.

The vectorization of the recurrence relations in the LU factorization preconditioners is done by a hyperplane method [1][27]. An efficient implementation of this method with list vectors and zero padding is discussed.

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 shows the theoretical and Section 7 the implementational aspects of the used preconditioners. Section 8 concludes with numerical results.

2 THE BASIC PARTIAL DIFFERENTIAL EQUATIONS

The semiconductor equations [24] 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

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

with the space charge $\rho = q \cdot (p - n + C)$, 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 $\vec{J}_{n,p}$ read

$$\mathrm{div}\vec{J}_n - q \cdot \frac{\partial n}{\partial t} = q \cdot R \tag{2}$$

$$\mathrm{div}\vec{J}_p + q \cdot \frac{\partial p}{\partial t} = -q \cdot R \tag{3}$$

where R denotes the carrier generation and recombination rate and the current densities $\vec{J}_{n,p}$ are given by

$$\vec{J}_n = -q \cdot \mu_n \cdot n \left(\mathrm{grad}\psi - \frac{1}{n} \cdot U_t \cdot \mathrm{grad}\, n \right) \tag{4}$$

$$\vec{J}_p = -q \cdot \mu_p \cdot p \left(\mathrm{grad}\psi + \frac{1}{p} \cdot U_t \cdot \mathrm{grad}\, p \right). \tag{5}$$

Calculating the carrier mobilities we take into account lattice, impurity, and surface scattering and velocity saturation for high electric fields. $U_t = \frac{k \cdot T}{q}$ denotes the thermal voltage.

We consider the time-invariant case of the semiconductor equations only. In the transient case additional terms increase the diagonal dominance of the discretized carrier continuity equations and therefore improve their condition number, if an implicit backward time difference scheme is used.

3 DISCRETIZATION AND ITERATIVE SOLUTION OF THE NONLINEAR SYSTEM OF EQUATIONS

We use finite difference discretization in a rectangular spatial domain to treat the semiconductor equations numerically. The coefficient matrices of the discretized equations have seven nonzero diagonals in natural ordering. 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 charge and 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 [3][6] make a simultaneous solution of the three semiconductor equations necessary and have a locally quadratic convergence behavior. We restrict ourselves to Gummel's algorithm [9], a block iterative Gauss–Seidel method, which allows a sequential solution of the three equations within each outer iteration. It is less sensitive to the initial guess than Newton's method and no Jacobian has to be calculated, thus keeping storage requirements low contrary to the Newton schemes. The locally linear convergence of Gummel's algorithm, often considered as its major drawback, can be improved by nonlinear convergence acceleration [15]. The nonlinear Gummel modification of Poisson's equation and the nonlinearity in the generation and recombination terms are linearized by a first order series expansion, the nonlinearities in the carrier mobilities and carrier temperatures are neglected. The resulting scheme reads

$$\mathrm{div}\,\mathrm{grad}\psi^{k+1} = -\frac{q}{\epsilon}\left(p^k - n^k + C + \frac{\partial (p - n + C)^k}{\partial \psi} \left(\psi^{k+1} - \psi^k \right) \right) \tag{6}$$

$$\mathrm{div}\vec{J}_p\left(\psi^{k+1}, p^{k+1}\right) = -q\left(R\left(\psi^{k+1}, n^k, p^k\right) + \frac{\partial R}{\partial p}\left(p^{k+1} - p^k\right) \right) \tag{7}$$

$$\mathrm{div}\vec{J}_n\left(\psi^{k+1}, n^{k+1}\right) = q\left(R\left(\psi^{k+1}, n^k, p^{k+1}\right) + \frac{\partial R}{\partial n}\left(n^{k+1} - n^k\right) \right). \tag{8}$$

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 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 grid lines and the modification by Gummel's algorithm 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, which can be transformed to a symmetric, positive definite matrix $\bar{A}$ [4][14]

$$\bar{A} = W_{n,p} \cdot A \cdot W_{n,p}^{-1} \tag{9}$$

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

$$w_{i,n} = \exp\left(-\frac{\psi_i}{2 \cdot U_t}\right), \quad w_{i,p} = \exp\left(\frac{\psi_i}{2 \cdot U_t}\right) \tag{10}$$

for electrons and holes. $U_t = \frac{k \cdot T}{q}$ denotes the thermal voltage. The enormous number range of the $w_{i,n,p}$ inhibits an explicit symmetrization.The symmetrizability guarantees a positive real spectrum of A.

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.

5.1 CGNR

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

$$A^T A x = A^T b \tag{11}$$

by the classical conjugate gradient algorithm. It is clear, that the matrix-product $A^T A$ is never built explicitly. Due to the minimization property of the conjugate gradient method the convergence behavior 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 have confirmed, 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 the linear system [12]. The cumbersome symmetrization matrix W is only required for the computation of the iteration parameters, where it appears both in the nominator and denominator. This allows scaling in order to avoid floating point under– or overflow. This algorithm minimizes the $\left[W^2 A\right]^{\frac{1}{2}}$-norm of the error vector. It has proved, that 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 GMRES

This algorithm [21] minimizes $\|r_k\|_2$. An orthogonal basis of x_k is built by means of an Arnoldi construction. Assuming that the procedure has converged in k steps ($\|r_k\| < \epsilon$), 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 within this orthogonal basis.
Full orthogonalization requires the storage of k "back"–vectors and the calculation of $k+1$ inner products at the k^{th} iteration. To limit storage requirements we *restart* GMRES after m (where m is a constant) iterations, if the method has not yet converged. The approximation to the solution at the end of every m iterations is used is initial value for the next m–step Arnoldi process. The truncated version GMRES(m) of course loses optimality but restarting preserves monotonicity in the residual–norm.

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

They are built by squaring the Lanczos biorthogonalization algorithms BIOMIN, BIORES, and BIODIR [11][26]. From the computational point of view they are more efficient than the original procedures. All three squared algorithms produce the same iterates for the same initial guess, but both from the aspect of the work per iteration and of numerical stability BIOMIN^2 (CGS) outperforms the other two procedures [14].
The biorthogonalization algorithms have *no* minimization properties. Therefore the residual usually does not decrease monotonically. Very often an erratic convergence behavior can be observed. This may increase the influence of roundoff errors.
The biorthogonalization algorithms may break down by division by zero also in exact arithmetic, if certain inner products vanish. A breakdown is likely to occur, if the initial guesses are chosen inappropriately, but was never observed in our examples.

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 the only algorithm out of those we have tested, which has no minimization property.

6 PRECONDITIONING

It turned out that preconditioning of iterative solvers is inevitable to fulfill the stability requirements. 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 a linear system with the preconditioning matrix as coefficient matrix has to be solved. This system must be solvable much easier than $Ax = b$.
We have investigated a block Jacobi preconditioner

$$P_J = D \tag{12}$$

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

$$P_{ILU} = P_L P_R = \left(\tilde{L} + \tilde{D}\right) \tilde{D}^{-\frac{1}{2}} \tilde{D}^{-\frac{1}{2}} \left(\tilde{U} + \tilde{D}\right) \tag{13}$$

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 well known incomplete Cholesky decomposition (IC) preconditioner.
For the Jacobi preconditioner left preconditioning has been chosen

$$P_J^{-1} A x \equiv B_J x = P_J^{-1} b \equiv \tilde{b}_J, \tag{14}$$

for the ILU preconditioners left and split preconditioning has been implemented

$$P_R^{-1} P_L^{-1} A x \equiv B_{ILU_{left}} x = P_R^{-1} P_L^{-1} b \equiv \tilde{b}_{ILU_{left}} \tag{15}$$

$$P_L^{-1} A P_R^{-1} P_R x \equiv B_{ILU_{split}} \tilde{x} = P_L^{-1} b \equiv \tilde{b}_{ILU_{split}}. \tag{16}$$

For the split ILU preconditioner x has to be unscaled at the end of the iterative process: $x = P_R^{-1}\tilde{x}$.
The computational work for the 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 as can be expected from theory [8] the Jacobi preconditioner causes worse convergence behavior for all tested accelerators than ILU preconditioners. For high bias simulations, where the drift term dominates in the the current relations (4) and (5) the Jacobi preconditioner does not guarantee a fast and reliable convergence of the iterative methods. Therefore 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(q) [1][5][13]. For $q = 0$ the matrices $\tilde{L}$ and $\tilde{U}$ are equal to the strictly lower and upper matrices L and U of A. The sparsity pattern of the triangular factors $\tilde{L}$ and $\tilde{U}$ therefore is the same as for the triangular parts of A. For $q = 1$ the fill-in caused by the ILU(0) nonzero pattern is taken into account, for $q = 2$ the fill-in caused by the ILU(1) sparsity pattern and so on (ILU($NX \cdot NY$) would be the exact LU factorization of A). A higher degree of fill-in of course 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 diag (P_{ILU}) = diag (A) or alternatively such that $(P_{ILU} - A)$ has zero *column* sums, which leads to modified incomplete factorization type preconditioners (MILU) originated by Gustafsson [10] for Poisson type equations (in the symmetric case this is equal to $rowsum(P_{IC}) = rowsum(A)$). For the symmetric and positive definite Poisson equation the modulus of the main diagonal is greater or equal than the offdiagonal elements of the very 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. We found a number of device examples where a choice of $\alpha = 0.5$ yields a performance enhancement of about 10% to 30% concerning the iteration count. This is rather disappointing, as for the symmetric Poisson equation a modification factor of $\alpha = 0.95$ reduces the iteration count up to 50%.
For $q = 0$ efficient implementations of the multiplications of the preconditioned coefficient matrix B and an vector v are possible. For the left (M)ILU(0) preconditioner we scale the coefficient matrix from the left side by $\tilde{D}$. The scaled matrix $A_{s_{left}} = \tilde{D}A$ can be written as a sum of a strictly lower, a diagonal, and a strictly upper matrix: $A_{s_{left}} = L_{s_{left}} + D_{s_{left}} + U_{s_{left}}$. The number of *flops* can be reduced from $13N$ (N is the number of grid points) to $10N$ by

$$\begin{aligned} B_{ILU_{left}} v &= \left(I + U_{s_{left}}\right)^{-1} \Big[v + \left(I + L_{s_{left}}\right)^{-1} \cdot \\ &\quad \cdot \left(D_{s_{left}} - I + U_{s_{left}}\right) v \Big]. \end{aligned} \tag{17}$$

An analogous simplification for the split (M)ILU(0) preconditioner, which saves even $6N$ flops, is well known as Eisenstat's trick [7]. The coefficient matrix A is scaled symmetrically by $\tilde{D}^{\frac{1}{2}}$: $A_{s_{split}} = \tilde{D}^{\frac{1}{2}} A \tilde{D}^{\frac{1}{2}}$. Then $B_{ILU_{split}} v$ can also be written as

$$B_{ILU_{split}} v = \left[t + \left(I + L_{s_{split}}\right)^{-1} \left(v - \left(2I - D_{s_{split}}\right) t\right) \right] \tag{18}$$

with

$$t = \left(I + U_{split}\right)^{-1} v. \tag{19}$$

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 THE PRECONDITIONERS ON VECTORCOMPUTERS

The basic components of all methods described above are operations like vector updates, inner products, and sparse matrix vector products, which are quite easily implementable and lead to high performance codes on vectorcomputers. However, those components containing recurrence relations do not vectorize or parallelize when coded in a straightforward manner. For the Jacobi preconditioner we have to deal with a first order linear recursion. For the preconditioning by incomplete factorization linear recursions of several orders appear.

7.1 VECTORIZED SOLUTION OF TRIDIAGONAL SYSTEMS

Vectorizable solution for tridiagonal systems may be obtained by different methods like recursive doubling, cyclic reduction [23], and the partition method [28][30].

The most efficient solver for the VP200 as well as for the ALLIANT/FX40 (with two processors) was the one factorizing the tridiagonal system into two bidiagonal ones and solving them by the partition method. Using the fact that the tridiagonal matrix is blockdiagonal due to the finite difference discretization no fill-in occurs and the solution of the bidiagonal system further simplifies to (NX, NY, and NZ denote the number of grid points in x–, y–, and z–direction, respectively.)

```
      DO 1 I=2,NX
      DO 1 J=1,NY*NZ
1     X(I,J) = X(I,J) - A(I,J)*X(I-1,J)
```

7.2 VECTORIZED SOLUTION OF TRIANGULAR SYSTEMS

In the following we will label the unknowns of the linear system $Ax = b$ with a triple of indices (i, j, k) corresponding to the grid coordinates of the three spatial dimensions.

The recurrence relation to be solved for the 3D problem with finite differences discretization reads for $q = 0$

$$x_{i,j,k} = r_{i,j,k} - b_{i,j,k}\, x_{i-1,j,k} - d_{i,j,k}\, x_{i,j-1,k} - f_{i,j,k}\, x_{i,j,k-1}. \tag{20}$$

The simplest way to achieve vectorizable codes in the recurrence relation is to *accumulate* all contributions to the vector x from an already computed plane $k-1$ to the right hand side of the equation:

$$\tilde{r}_{i,j,k} = r_{i,j,k} - f_{i,j,k}\, x_{i,j,k-1} \tag{21}$$

This is a vector operation of length $O(NX \cdot NY)$. In MINIMOS this yields vector lengths of about 400 up to 3600 and good performance is achievable.

One can proceed in a similar manner now to accumulate the contributions to x in a fixed plane from an already computed line $j-1$. This yields vector operations of length $O(NX)$, typically between 20 and 60 for MINIMOS. This is not too bad for a Cray or for our superminicomputer ALLIANT/FX40, but the VP200 due to the large $n_{1/2}$ [23] is far off the performance limits.

To end up one has to solve the remaining first order recurrence. Due to the short vector length it is no use to try to vectorize this recursion, so it is executed at a poor speed on the VP200 and the overall gain due to vectorization is low. Note however, that by unrolling the loop for the bidiagonal system on ALLIANT/FX40 the performance of this operation gains by about a factor of 2.

The next variant to be discussed is one which uses the (effective) accumulation for the planes and solves them by a *grid-diagonal* approach. A grid-diagonal is defined by the set of all grid points (i, j, k), for which $i + j = const$, for a fixed k. Unknowns in a diagonal can be computed in vector mode now from already computed quantities of the previous grid-diagonal.

High computational speed for the triangular backward substitutions is reported for the so-called *hyperplane method* [1][29]. A hyperplane H_m is now defined by all triples (i, j, k) for which $i + j + k = m$. All unknowns belonging to H_m can be computed independently from those belonging to the previous plane H_{m-1}.

A straightforward implementation of this algorithm would consist of three nested loops, the outermost one for all hyperplanes, one for all planes, and the innermost one for all diagonals of this plane. The two inner loops can be executed in vector mode. However, the FORT77 compiler of the VP200 denies to vectorize multiple loops with variable loop lengths. Because of its large $n_{1/2}$ it is very important for the VP200 to gain advantage from the fact that the number of unknowns in the hyperplanes is rather large — up to $O(N^{\frac{2}{3}})$. This is done by forming a vector out of all the unknowns of each hyperplane by storing the addresses of the unknowns to be processed in a list vector and marking the beginnings of the individual hyperplanes before starting the iterations. By this way the two inner loops are combined to one, which now can be totally vectorized.

Inherent to this method is the need for indirect addressing. And there is still the problem of how to avoid

unallowed addressing on the boundary of the simulation domain. Van der Vorst [29] has suggested to accumulate the contributions of the different sub-diagonals in one hyperplane by individual loops. This however introduces considerable loop overhead. Other possibilities are to calculate those unknowns outside the loop or to avoid unallowed addressing by use of IF-statements. We have obtained best results by extending the arrays of the unknowns at the lower and upper ends by an amount of the number of elements in one plane and fill them with zeros. The algorithm then reads:

```
      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)
     1          -F(I)*X(I-NX*NY)
```

Note that almost all vectorizable algorithms discussed can be used to vectorize the factorization (except for the the modified ILU preconditioners for $q > 0$) in a similar manner.

For preconditioning allowing fill-in (i.e. (M)ILU(q), $q = 1, 2, \ldots$) the definition of the hyperplane H_m must be extended to the set of mesh points fulfilling the relation

$$i + (q+1)(j+k) = m \qquad (22)$$

where q denotes the degree of fill-in. For the lower triangular system the unknowns in H_m can be calculated independently from those of H_{m-1}, for the upper triangular system from those of H_{m+1}. Implementation by the above mentioned list vector method is straightforward.

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 a nonplanar 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), $U_G = 0.5$V (gate) and $U_D = 1$V (drain). 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, on the horizontal axis the number of flops divided by N, the number of grid points, is plotted.

Figure 1 shows the convergence curves. CGNR is certainly not competitive. In this easy example the symmetrized CG, SYMCG is almost as fast as BIOMIN^2 (CGS), but unfortunately this solver is not applicable for higher bias voltages. GMRES is a reliable, but slow alternative to BIOMIN^2 (CGS).

In Table 1 we show the CPU time requirements (in seconds) for one solution of the Poisson and carrier continuity equations using different variants to solve the triangular systems introduced by the modified IC(0) ($\alpha = 0.95$) and ILU(0) preconditioners. The methods described in Section 7 are compared to the straightforward (autovectorized) implementation. The test example was a n–channel MOSFET with a channel length of 1.5 micron and the bias conditions $U_D = U_G = 3V$ and $U_S = U_B = 0V$.

Table 2 shows a comparison (CPU time in seconds) of the vector performance of hyperplane preconditioners with different degrees of fill-in and the standard non-vectorizable recursions (autovector). We have used ILU(0), ILU(1), and ILU(2) together with the conjugate gradient method on different machines. The matrices for the linear system were obtained by discretizing the Laplace equation on a rectangular grid with 40^3 grid points. Although the number of iterations (IT) is obviously reduced by a higher degree of fill-in, this is compensated by the higher work per iteration. Due to Eisenstat's trick ILU(0) needs less CPU time than the higher degree fill-in preconditioners in this example.

Table 3 compares the performance of the hyperplane triangular backsubstitutions for the ILU(q) preconditioners on a SIEMENS/Fujitsu VP200 (VP), a Cray-2 (C2) and an ALLIANT/FX40 (AL) computer. The test example is as for Table 2. Besides the CPU time for one backsubstitution in milliseconds (ms), the overall achieved speed-up over the trivial code in the solution of the triangular systems and the megaflop (Mflops) rate are presented. The hyperplane code is obviously better suitable for the VP200 than for the Cray-2. One reasons is the good performance of the Fujitsu vectorcomputer for long vector length. Possible memory bank conflicts on the Cray due to the indirect addressing according to the hyperplane-ordering decreases the performance of the Cray supercomputer substantially.

In order to show that our code can also be parallelized on tightly coupled multiprocessor computers to a high extent we carried out tests for the hyperplane backsubstitutions on a 6-scalar-processor Digital VAX 6260. Table 4 shows speedups against one processor.

To demonstrate that results for three-dimensional device simulations are obtained in relatively short times on supercomputers we have selected two examples of rather low complexity, namely a n–channel MOSFET with channel length and width of about one micron and a p–channel MOSFET with similar dimensions. Bias conditions were $U_{DS} = U_{GS} = 3V$ for the n–MOSFET example and $U_{DS} = -1V$, $U_{GS} = -4V$, and $U_{BS} = 2V$ for the p–MOSFET example. The CPU times (in seconds) for a fully three-dimensional simulation using MICCG(0) for the Poisson equation

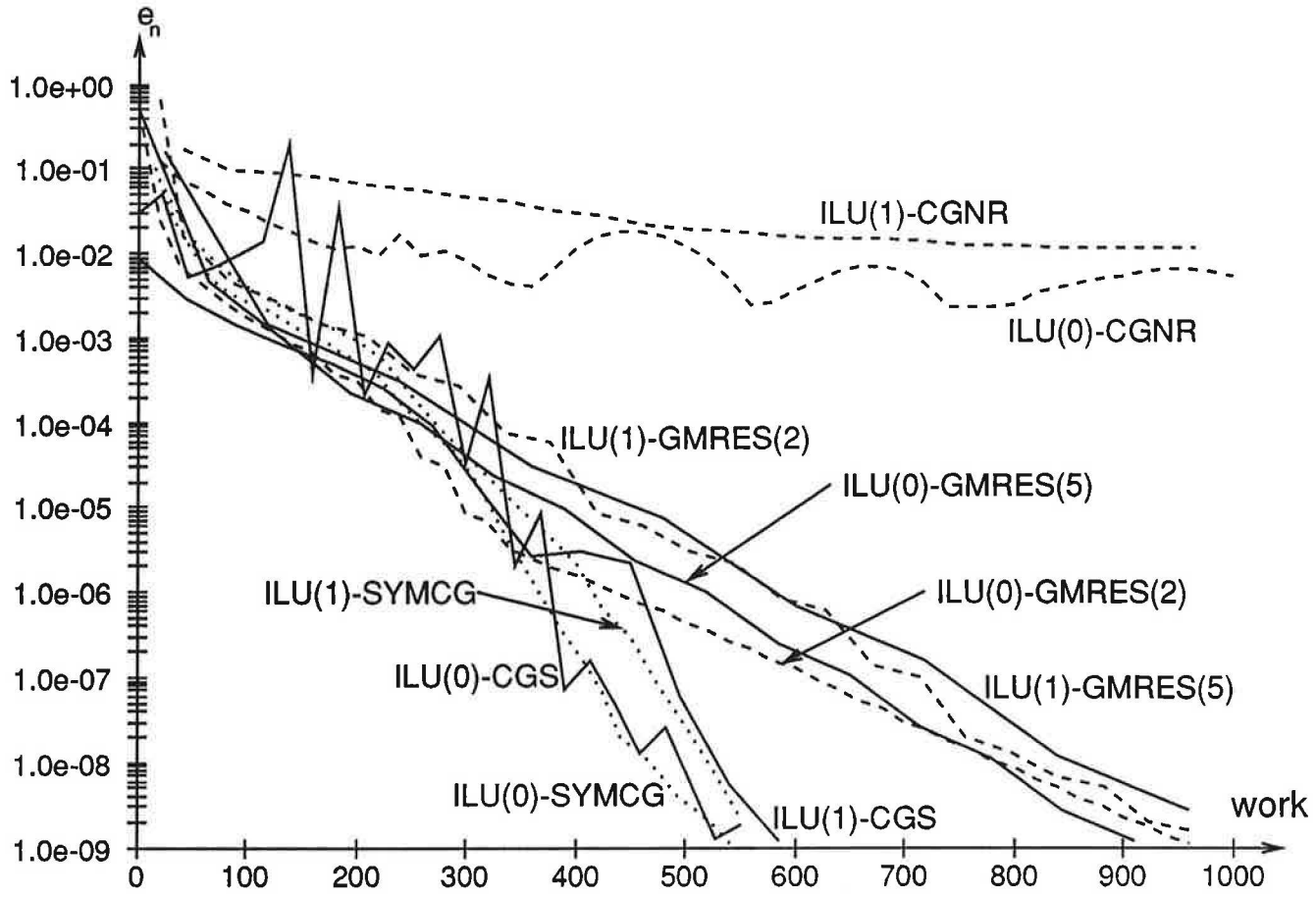

Figure 1: Convergence of Accelerators

Type	VP200		ALLIANT	
	MICCG	ILUCGS	MICCG	ILUCGS
autovector	0.100	0.826	3.19	7.47
accumulation	0.090	0.635	2.60	5.26
grid-diagonal	0.064	0.337	2.84	5.52
hyperplane	0.020	0.093	2.07	3.97

Table 1: Vectorization Methods for Triangular Backsubstitutions

q	IT	VP200		Cray-2		ALLIANT	
		auto	hyper	auto	hyper	auto	hyper
0	61	15.5	1.29	15.1	5.19	128	85.8
1	53	22.4	1.84	30.8	8.23	177	117.0
2	50	27.9	2.21	38.3	8.33	206	124.6

Table 2: Comparison of ILU(q) Preconditioners

q	VP200			Cray-2			ALLIANT		
	time	speed-up	Mflops	time	speed-up	Mflops	time	speed-up	Mflops
0	4	13.75	96	14	4.30	27	212	1.34	1.8
1	8	12.12	96	26	4.76	30	327	1.51	2.3
2	8	12.62	128	30	5.93	34	410	1.64	2.5

Table 3: Hyperplane-ILU(q) Backsubstitutions on Vectorcomputers

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

Table 4: Parallel Hyperplane ILU(0) on VAX 6260

Example	VP200	Cray-2	ALLIANT
n-MOSFET	28.87	72.92	718.2
p-MOSFET	30.32	55.77	629.5

Table 5: CPU Times for Fully Three-Dimensional Device Simulation

and ILUCGS(0) for the carrier continuity equations are shown in Table 5.

The highly satisfactory result is that one bias condition can be simulated in half a minute on the VP200. Thus it is possible to turn ones focus to really complex problems in the near future. For the VP200 the overall vectorization rate is 96%.

A straightforward analysis of the CPU time consumption shows that tuning the solvers further has only little influence on the total performance. For our low complexity example a gain of 2 for the nonsymmetric solver would lead to a total performance gain of only about 5%. For highly complex examples the performance gain will be better as long as swapping memory out of core (either done by a virtual operation system or the virtual memory package distributed with `MINIMOS`) will not become too time consuming, when memory requirements due to large grids increase.

9 ACKNOWLEDGEMENTS

This work has been initiated and sponsored by SIEMENS AG, Munich. It has also been supported by 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.

References

[1] Ashcraft, C.C., Grimes, R.G., "On Vectorizing Incomplete Factorization and SSOR Preconditioners", *SIAM Journal of Scientific and Statistical Computing*, Vol. 9, No. 1, January 1988, pp. 122-151.

[2] Ascher, U., Markovich, P.A., Schmeisser, C., Steinrueck, H., Weiss, R., "Conditioning of the Steady State Semiconductor Problem", Tech. Rep. 86-18, Dept. of CS, Univ. of British Columbia, Vancouver.

[3] Bank, R.E., Rose, D.J., "Global Approximate Newton Methods", *Numerische Mathematik* 37 (1981), pp. 279-295.

[4] Bank, R.E., Rose, D.J., Fichtner, W., "Numerical Methods for Semiconductor Device Simulation", *IEEE ED-30*, pp. 1031-1041, 1983.

[5] Concus, P., Golub, G.H., Meurant, G., "Block Preconditioning for the Conjugate Gradient Method", *SIAM Journal of Scientific and Statistical Computing*, Vol. 6, No. 1, January 1985, pp. 220-252.

[6] Dembo, R.S., Eisenstat, S., Steinhaug, T., "Inexact Newton Methods", *SIAM Journal of Scientific and Statistical Computing*, Vol. 18, (1981), pp. 400-408.

[7] Eisenstat, S.C., "Efficient Implementation of a Class of Preconditioned Conjugate Gradient Methods", *SIAM Journal of Scientific and Statistical Computing*, Vol. 2, No. 1, March 1981, pp. 1-4.

[8] Elman, H.C., Golub, G.H., "Line Iterative Methods for Cyclically Reduced Discrete Convection Diffusion Problems", Proceedings of the Copper Mountain Conference on Iterative Methods, Vol. 1, April 1990.

[9] Gummel, H.K., "A Selfconsistent Iterative Scheme for Onedimensional Steady State Transistor Calculations", *IEEE ED-11*, pp. 455-465, 1964.

[10] Gustafsson, I., "A Class of First Order Factorization Methods", BIT, 18 (1978), pp. 142-156.

[11] Gutknecht, M.H., "The Unsymmetric Lanczos Algorithms and their Relations to Padé Approximations, Continued Fractions and the QD Algorithm", Proceedings of the Copper Mountain Conference on Iterative Methods, Vol. 2, April 1990.

[12] Hageman, L.A., Luk, F.T. and Young, D.M., "On the Equivalence of certain Iterative Acceleration Methods", *SIAM Journal of Numerical Analysis*, Vol. 17, No. 6, December 1980, pp. 852-873.

[13] Hane, K., "Supercomputing for Process/Device Simulations", Proceedings of the Sixth International NASECODE Conference, July 1989, pp. 11-21.

[14] Heinreichsberger, O., Selberherr, S., Stiftinger, M., Traar, K.P., "Fast Iterative Solution of Carrier Continuity Equations for 3D Device Simulation", *SIAM Journal of Scientific and Statistical Computing*, to be published.

[15] Kerkhoven, T., Saad, Y., "Acceleration Techniques for Decoupling Algorithms in Semiconductor Simulation." Tech. Rep., Dept. of Computer Science, Univ. of Illinois at Urbana Champaign, 1987.

[16] Meijerink, H., van der Vorst, H., "An Iterative Solution Method for Linear Systems of which the Coefficient Matrix is a Symmetric M-Matrix", Math. Comp., 31 (1977), pp. 148-162.

[17] Nachtigal, N.M., Reddy, S.C., Trefethen, L.N., "How fast are nonsymmetric iterations ?", Proceedings of the Copper Mountain Conference on Iterative Methods, Vol. 4, April 1990.

[18] Oppe, T.C., Joubert, W.D., and Kincaid, D.R., "NSPCG User's Guide.", Center of Numerical Analysis, University of Texas at Austin, 1984.

[19] Polak, S.J., Den Heijer, C., Schilders, W.H.A., "Semiconductor Device Modelling from the Numerical Point of View", Int. J. Num. Methods in Eng., Vol. 24, pp. 763-838, 1987.

[20] Saad, Y., "The Lanczos Biorthogonalization Algorithm and other Oblique Projection Methods for Solving Large Unsymmetric Systems", *SIAM Journal of Scientific and Statistical Computing*, Vol. 19, No. 3, June 1982, pp. 485-506.

[21] Saad, Y. and Schultz, M.H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems.", *SIAM Journal of Scientific and Statistical Computing*, Vol. 7, No. 3, July 1986, pp. 856-869.

[22] Scharfetter, D.L., Gummel, H.K., "Large–Signal Analysis of a Silicon Read Diode Oscillator.", *IEEE ED-16*, pp. 64-77, 1969.

[23] Schönauer, W., "Scientific Computing on Vector Computers", *Special Topics in Supercomputing*, Vol. 2, North Holland, 1987.

[24] Selberherr, S., "Analysis and Simulation of Semiconductor Devices", *Springer-Verlag Wien New York*, ISBN 3-211-81800-6, 1984.

[25] Selberherr, S., et al., "MINIMOS 5 User's Guide", Technical University Vienna, 1990.

[26] Sonneveld, P., "CGS, A fast Lanczos–Type Solver for Nonsymmetric Systems", *SIAM Journal of Scientific and Statistical Computing*, Vol. 10, No. 1, Jan. 1989, pp. 36-52.

[27] Traar, K.P., Mader, W., Heinreichsberger, O., Selberherr, S., Stiftinger, M., "High Performance Preconditioning on Supercomputers for the 3D Device Simulator Minimos.", Proceedings of the Supercomputing 90, Nov. 1990, pp. 224-231.

[28] van der Vorst, H., Dekker, K., "Vectorization of Linear Recurrence Relations", *SIAM Journal of Scientific and Statistical Computing*, Vol. 10, No. 1, Jan. 1989, pp. 27-35.

[29] van der Vorst, H., "High Performance Preconditioning", *SIAM Journal of Scientific and Statistical Computing*, Vol. 10, No. 6, Nov. 1989, pp. 1174-1185.

[30] Wang, H.H., "A Parallel Method for Tridiagonal Equations", *ACM Trans. Math. Software*, 7, pp. 170-183, 1981.