HIGH PERFORMANCE PRECONDITIONING ON SUPERCOMPUTERS FOR THE 3D DEVICE SIMULATOR MINIMOS

K. P. Traar and W. Mader

SIEMENS AG Österreich, Electronic Developement,
Gudrunstr. 11, A-1101 Vienna, Austria

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

Institut für Mikroelektronik, TU-Vienna,
Gußhausstrasse 27-29, A-1040 Vienna, Austria

Abstract

Discretization and iterative solution of the semiconductor equations in a three-dimensional rectangular region lead to very large sparse linear systems. Nevertheless, design engineers and scientists of device physics need reliable results in short time in order to draw the best advantage out of computer simulation when designing new technologies and advanced devices. This goal can be achieved by use of preconditioned iterative methods for the solution of the linear equations on powerful computers as vector and concurrent supercomputers. To achieve optimum performance of such methods special algorithms and coding techniques have to be used in order to allow vectorization and parallelization of the inherent recurrence relations. We have investigated Jacobi and incomplete LU-factorization methods along with various hand tuning options with special emphasis on application on a SIEMENS/Fujitsu VP200. Results are compared with those obtained on a Cray-2 and an ALLIANT/FX40 minisupercomputer.

Introduction

Rapid development in very large scale integration (VLSI) technology has lead to submicron semiconductor devices. In the eighties two-dimensional simulators were able to predict device performance accurately enough to give valuable aid in successful chip design. Further shrinking in the device dimensions make fully three-dimensional simulation necessary.

However, the threshold to use a 3D-simulator for scientists in device physics as well as for design engineers lies in the performance of the simulation program. Generally speaking, systems of partial differential equations with highly nonlinear coefficients have to be solved within the time of the order of one minute for a given set of boundary values.

In the past decade MINIMOS has proven to be a very powerful tool in 2D-simulation of MOS devices [1]. Recently, a 3D version of the program has been made [2]. However, time and memory consumption were rather intensive for the very first version. Therefore, much effort has been undertaken to design faster solvers for the large sparse linear systems arising from the semiconductor equations. Preconditioned iterative methods turned out to be the best choice to achieve both stability and high performance especially for the solution of the nonlinear systems resulting from the continuity equations [3]. To optimally exploit the resources of supercomputers special algorithms and coding techniques have to be used to allow vectorization and/or parallelization of the recurrence relations inherent in such methods. Our main task was to optimize the performance of 3D-MINIMOS on a SIEMENS/Fujitsu VP200.

The Problem and the Solution Method

The Basic Partial Differential Equations

The classical partial differential equations (PDE) of semiconductor devices in the stationary state [4] are Poisson equation (1) and carrier continuity equations for electrons (2) and holes (3) and are given as follows:

$$\nabla(\epsilon\nabla\psi) = q(C_T + p - n) \tag{1}$$

$$\nabla\left(\mu_n k\nabla(nT_n) - \mu_n n\nabla\psi\right) = R \tag{2}$$

$$\nabla\left(\mu_p k\nabla(pT_p) + \mu_p p\nabla\psi\right) = R \tag{3}$$

ψ is the electrostatic potential, the permitivity is denoted by ϵ, q denotes the elementary charge, C_T is net doping, n and p are electron and hole concentrations, k denotes the

CH2916-5/90/0000/0224/$01.00 © IEEE

Boltzmann constant, and T_n and T_p denote the absolute temperature of electrons and holes, respectively. The mobilities of the carriers are denoted by μ_n and μ_p, R denotes the net recombination/generation rate.

To solve this set of coupled differential equations of the unknowns ψ, n and p with MINIMOS it is discretized in a three-dimensional rectangular domain by finite differences and boundary conditions are specified. At the terminal contacts, which (in general) are assumed to be ideally ohmic, the boundary conditions are of Dirichlet type. Homogenous Neumann boundary conditions are used for the artificial interfaces in the deep bulk and inhomogenous Neumann boundary conditions are applied in case of non-vanishing interface charge and interface recombination velocity.

Due to huge variations of the distributions of the carrier concentrations and the potential in small subareas of the simulation domain, a non-uniform grid is used. The solution of this system of PDEs in MINIMOS is obtained using a block-nonlinear iterative algorithm frequently called Gummel's method [5].

A special feature of MINIMOS 5 is the capability of handling non-planar interfaces using the well known box integration method. For more details on the program features see reference [6].

Discretization and Iterative Solution

From the numerical point of view the properties of the linear systems resulting from discretization and linearization in the Gummel cycles are of interest: The nonlinear Gummel modification of the Poisson equation is linearized by a one-step Newton iteration and the resulting matrix is symmetric, positive definite and has property A.

For the nonlinear carrier continuity equations the terms R and μ have to be further investigated. μ depends on the driving forces, but this fact is neglected in MINIMOS during the nonlinear iteration. The net recombination R consists of three parts, namely Shockley-Read-Hall, Auger, and Impact Ionization. Whereas Impact Ionization is only updated in the generation subcycles and therefore is assumed to be constant during the Gummel cycles, the derivatives of R^{SHR} and R^{AU} with respect to n and p are accounted for. Whereas the derivatives of R^{SRH} allways increase diagonal dominance, the contributions of R^{AU} may decrease it and may even destroy definiteness of the system. These negative contributions have to be discarded during iteration.

Using a modified Scharfetter-Gummel interpolation scheme [7] to allow for carrier temperature dependent mobilities [8] the box integration scheme leads to nonsymmetric system matrices with property A.

After discretization the matrices resulting from the Poisson equation and the carrier continuity equations can be viewed as block tridiagonal matrices with the off-diagonal elements being diagonal matrices responsible for the coupling between planes (points with constant k). The diagonal blocks may again be viewed as block tridiagonal systems with the off-diagonal blocks being diagonal matrices, now responsible for the coupling between lines in a plane. The remaining diagonal blocks are tridiagonal systems which define the relations between the unknowns in a line.

In the 2D-simulator the symmetric systems are solved by a cyclic Jacobian conjugate gradient (CJCG) method and the nonsymmetric ones by Gaussian elimination. These solvers have proven to be very effective in the past. However, for 3D problems Gaussian elimination will both need to much memory for most of the operational environments and be rather slow. Well chosen iterative methods can cope with both problems.

Iterative Methods and Preconditioning

Various methods for the solution of a linear system $Ax = b$ with seven star discretization have been reported [9], the most promising of them for our purposes are conjugate gradients (CG) [10], [11] for symmetric positive definite problems and projection-type methods such as ORTHODIR, ORTHOMIN, and ORTHORES [12], GMRES [13], LSQR [14], Bi-CG [15], and CGS [16] for nonsymmetric ones. As the applications for the simulator may have various different levels of complexity it was important to find solvers which are fast and stable for all the different simulation problems. For simplicity of installation and use we decided to chose only two solvers for one MINIMOS-3D installation, one for the Poisson equation and one for both carrier continuity equations.

It turned out that preconditioning of iterative solvers is inevitable to fulfill the stability requirements. Generally, preconditioning can be viewed as finding a system $B\tilde{x} = \tilde{b}$ instead of $Ax = b$, where B has better spectral properties than A with respect to the acceleration method used. This can be achieved by defining $\tilde{x}$ by $P_R\tilde{x} = x$ and $\tilde{b} = P_L b$, where $B = P_L A P_R$ and $P_R P_L$ can be viewed as approximate inverse to A. For the cases we have studied intensively, the product $P_L^{-1}P_R^{-1}$ reads either

$$K = P_L^{-1}P_R^{-1} = tridiag(A) \tag{4}$$

indicating a line Jacobi preconditioner or

$$K = P^{-1}P^{-1} = (S + X)X^{-1}(X + T) \tag{5}$$

indicating incomplete factorization methods.

In this paper we focus on cases where X in (5) is a diagonal matrix corresponding to a point preconditioner. We have also used a plane Jacobi preconditioner as well as an incomplete line LU preconditioner and compared our FORTRAN production code with the NSPCG [17] package. Results obtained are presented in [3].

In the symmetric case (for $S = T^T$) when a conjugate gradient (CG) accelerator is used and when $S = L$ (L is the strictly lower triangular part of A) we obtain the well known ICCG[0]-algorithm for $\text{diag}(K) = \text{diag}(A)$ and it's modified variant, the MICCG[0]-algorithm for $\text{rowsum}(K) = \text{rowsum}(A)$ [19]. In both cases K has the same sparsity pattern as A (no additional fill-in occurs, which is expressed by the suffix [0]).

More general methods allow for fill-in resulting in the [M]ICCG[q] algorithms with $q = 1, 2, \ldots$, indicating that fill-in for q non-zero patterns is included in matrix K and S and T are no longer the lower and upper triangular parts of A. The number of iterations is reduced for these methods and the stability is improved, however, memory requirements and time per iteration are enlarged. Due to the good performance of the [M]ICCG[0] method we saw no needs to investigate these methods in more detail.

Similar methods are applied to the nonsymmetric case resulting in the ILU[q] algorithms. We have introduced a modified ILU algorithm (MILU) using the following analogy: In the symmetric case (Poisson equation) $\sum^j A_{ij} \leq A_{ii}$ holds for the row sum. As in the nonsymmetric case (carrier continuity equations) the similar relation $\sum^i A_{ij} \leq A_{jj}$ holds for the column sum we use a preconditioner with $\text{columnsum}(K) = \text{columnsum}(A)$ resulting in the MILU[q] algorithm.

We have selected the CGS method which has proven to be the best choice for nonsymmetric systems [3] for application with [M]ILU preconditioning. As in the symmetric case a modification parameter ω in the range $[0, 1]$ is introduced to tune the modification. Unlike in the symmetric case, where the number of iterations can be reduced by a factor of about two for a well chosen ω of about 0.95, we observed that in the nonsymmetric case the reduction of iterations seems to depend strongly on the given problem. For P-channel MOS transistors the overall performance of the simulator can be enhanced by about 20 % for $\omega = 0.8$ (see Table 1). Symmetric (two-sided) and nonsymmetric preconditioning were used together with the Eisenstat tricks reviewed in the next chapter.

However, the dependence of the overall performance on the modification parameter ω has to be further investigated.

Table 1: Comparison of ILU and MILU preconditioned CGS solvers.
CPU-time/s of MINIMOS run for NMOSFET example on VAX 8800 (one processor).

preconditioning	**ILU** $\omega = 0$	**MILU** $\omega = 0.8$
two-sided:	2273.3	2083.4
left-sided:	3372.6	2274.3

Implementation and Vectorization

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 vector computers. However, those components containing recurrence relations do not vectorize or parallelize when coded in a straightforward manner. For the Jacobi methods we have to deal with a first order linear recursion. For the preconditioning with incomplete factorization linear recursions of several orders appear.

During the last few years many authors have suggested vectorizable alternatives to the above mentioned problem [18]–[25]. They showed how the problem can be solved in principle. The sucess of these methods highly depends on the type of supercomputer used. As the SIEMENS/Fujitsu VP200 has a rather slow scalar unit and it's vector unit operates on very long vectors as compared to other supercomputers ($n_{1/2}$ is rather large [21], [26] it is especially important to optimize both the vectorization rate to avoid scalar operations and the vector speed by using vectors as long as possible.

All efforts to improve performance of the dotproduct and the SAXPYs on VP200 (compiler FORT77/VP V10L03.031) and ALLIANT FX40 (fortran/fx 4.2.40) by improving the coding were not very successful gaining at maximum a few percent, thus showing that present day compilers let a user focus on the algorithmic aspects of vectorization.

Efficient Implementation of $P_L A P_R v$

For the following we use the identity $A = L + D + U$, with L and U being the strictly lower and upper triangular parts of A, respectively, and D being the main diagonal of A. Scaled

matrices are indicated by a tilde, a bar denotes matrices of the preconditioner.

The above mentioned [M]ILU[0] preconditioner allows for right ($P_L = I$), left ($P_R = I$) and two-sided preconditioning ($P_L = (X^{1/2} + X^{-1/2} L)^{-1}$, $P_R = (X^{1/2} + X^{-1/2} U)^{-1}$). For the last case the scaling $\tilde{A} = X^{-1/2} A X^{-1/2}$ is recommended ($X^{-1/2}$ exists for cases where A is positive definite) leading to a preconditioner $(\tilde{L} + I)(I + \tilde{U}) = \bar{L}\bar{U}$.

Eisenstat [27] has pointed out that the computation of the vector $\bar{U}^{-1} A \bar{L}^{-1} v_i$ can be implemented efficiently using the identity

$$\bar{U}^{-1} A \bar{L}^{-1} v_i = t + \bar{L}^{-1}(v_i - W t) \qquad (6)$$

where $W = (\tilde{D} - 2I)$ and $t = \bar{U}^{-1} v_i$.

In a similar manner one can recast the expression $(U + X)^{-1} X (X + L)^{-1} A v_i$ to

$$\begin{aligned} (U + X)^{-1} X (X + L)^{-1} A v_i = \\ = \bar{U}^{-1} \bar{L}^{-1} (\bar{L} + \bar{U} + X^{-1} D - 2I) v_i = \\ = \bar{U}^{-1} (v_i + \bar{L}^{-1} M v_i) \end{aligned} \qquad (7)$$

where now $M = (\bar{U} + X^{-1} D - 2I)$, $\bar{U} = (X^{-1} U + I)$, and $\bar{L} = (X^{-1} L + I)$.

In the 3D case the matrix vector product for a heptadiagonal system needs about $13 \times N$ flops. For two-sided preconditioning this amount is replaced by only about $3 \times N$ flops using the Eisenstat trick thus saving $10 \times N$ flops. The modified trick for the one-sided preconditioning explained above saves about $3 \times N$ flops.

The one-sided preconditioning (we have used the left-sided one) converges faster but requires approximately $7 \times N$ flops more per iteration than the two-sided one. However, we have observed that two-sided preconditioning is favourable for our applications (see Table 1).

Vectorized Solution of Tridiagonal Systems

Vectorizable solution for tridiagonal systems may be obtained by different methods like recursive doubling, cyclic reduction [21], and the partition method [22], [25].

The most efficient solver for the VP200 as well as for the ALLIANT FX40 (with two processors) was that 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:

```
      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)
```

In Table 2 we show cpu-time results on VP200 for test matrices obtained during MINIMOS runs. Unless stated otherwise all the tests used matrices obtained from the NMOSFET example described in Section 4. Speed-up gives the relation between the straightforwardly coded, autovectorized version and the vectorized partition method. As Poisson solver we used a cyclic Jacobi CG accelerated method, the continuity equations were solved using a line Jacobi BiCG.

Table 2: Comparison of Jacobi Solvers on VP200 (values given are CPU-time/s, It. is the iteration count).

Equation	It.	autovector	partition	speed-up
Poisson:	10	0.054	0.014	3.87
Minorities:	64	0.773	0.177	4.36

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:

$$x_{i,j,k} = y_{i,j,k} - a_{i,j,k}\, x_{i-1,j,k} - b_{i,j,k}\, x_{i,j-1,k} - c_{i,j,k}\, x_{i,j,k-1} \qquad (8)$$

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:

$$\tilde{y}_{i,j,k} = y_{i,j,k} - c_{i,j,k}\, x_{i,j,k-1} \qquad (9)$$

This is a vector operation of length $O(NX * NY)$, NX and NY being the number of grid points in x- and y-direction, respectively. 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 to bad for a CRAY or for our supermini ALLIANT FX40, but the VP200 due to the large $n_{1/2}$ 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 vectorizable coding for 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.

A vectorizable alternative to equation (8) has been suggested by van der Vorst and is called the **truncated Neumann series** approach [28]. Typically one has to compute:

$$x_{i,j,k} = y_{i,j,k} - a_{i,j,k}\left(y_{i-1,j,k} - a_{i-1,j,k}\, y_{i-2,j,k}\right) \qquad (10)$$

which can be combined with the accumulation of lines mentioned above. However, for our applications the results for this method were disappointing.

The next variant to be discussed is one which uses the (effective) accumulation for the planes and solves 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 solution of $\bar{L}x = y$ is reported for the so-called **hyperplane method** [18], [23], [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 it's 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 $O(NX * NZ/2)$. This is done by forming a vector out of all the unknowns of each hyperplane by putting 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. There is still another problem to be solved: What about the unknowns on the boundary of the simulation domain? Van der Vorst [23] 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:

```
      DO 1 L=1,NX+NY+NZ-2
      DO 1 M=MPT(L-1)+1,MPT(L)
      I=LIST(M)
1     X(I) = Y(I) - A(I)*X(I-1) - B(I)*X(I-NX)
     &                          - C(I)*X(I-NX*NY)
```

Note that almost all vectorizable algorithms discussed can be used to vectorize the factorization in a similiar manner.

In Table 3 we show the cpu-time requirements for the different variants to solve triangular systems using ILU-preconditioners. The methods explained above are compared to the straightforward (autovectorized) implementation. MICCG[0] was used for the Poisson equation test matrix (NMOSFET example) and ILU[0]-CGS for the continuity equation (minorities).

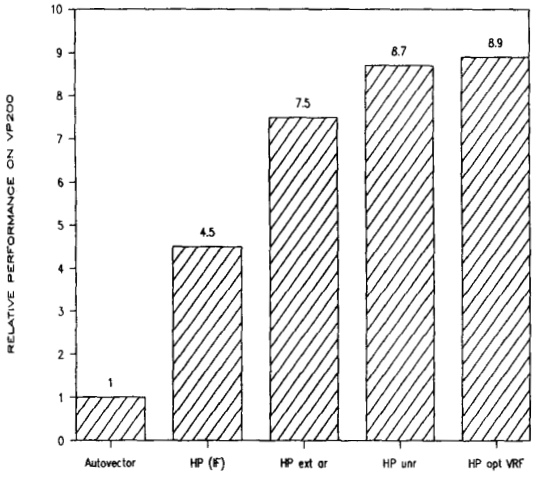
Table 3: Comparison of Triangular System Solvers (cpu-time/s).

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

In Figure 1 we show the gain of the different vectorization measures as compared to unvectorized straightforward elimination coding. Also shown are the effects of further hand tuning measures, like unrolling the outer loop or optimizing the vector register file for the SIEMENS version. We have

also tried reordering to avoid indirect addressing of the coefficients. This resulted in a 10 % gain for the ALLIANT only.

Figure 1: Effects of tuning methods to enhance vectorization on VP200: hyperplane with IF-statements, with extended arrays, additional loop unrolling, and additionally optimized vector register file, compared to autovectorized straightforward implementation.



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

$$i + (q+1)(j+k) = m$$

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.

In Table 4 we compare 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 CGS-accelerator 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.

The results of Table 5 were computed using MINIMOS test matrices (minority continuity equation) of the NMOSFET example.

Table 4: Comparison of ILU(q)-CGS Solvers with $q = 0, 1, 2$ with Standard and Hyperplane Method (values given are CPU-time/s, It. is the iteration count).

[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]	63	35.1	2.78	48.2	10.5	259	157.0

Table 5: Comparison of Hyperplane ILU[q]-CGS Solvers.

Method	VP200		ALLIANT	
	time/s	Iterations	time/s	Iterations
ILU[0]-CGS	0.107	33	4.74	33
ILU[1]-CGS	0.119	17	3.95	17
ILU[2]-CGS	0.227	21	5.58	21

Note, that the main advantage of fill-in preconditioners is not speed but stability.

The ILU[0]-CGS needs two inner products (in the performance limit of 350 Mflops on the VP200 [19], [26], seven vector updates (250 Mflops), and six matrix vector products. Using the Eisenstat trick this results in 48 N flops per iteration [3]. Taking into account the flops needed by the factorization and for computing the norms we measured about 150 Mflops on the VP200 and about 40 Mflops on the CRAY-2. For ILU[1] and ILU[2] slightly lower performance rates are obtained.

For the autovectorized version we have 24 N flops for the non-vectorizable 3-term recursion (6) performing at about 6 Mflops, thus we can estimate the performance to be about $48/24 \times 6 = 12$ Mflops on VP200 (assuming very high performance for the vectorized components), which agrees reasonably with the measured 12.6 Mflops.

Conclusions

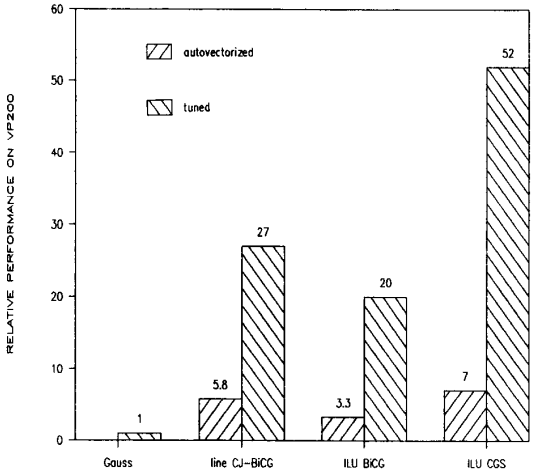
For the discretized Poisson equation a line Cyclic Jacobi-CG (CJ-CG) solver and [M]ILU preconditioned CG (called

[M]ICCG[0]) solvers were used. The CJ-CG solver is very fast and stable and for low size problems even beats the MICCG[0] method. However, for medium and large size examples MICCG[0] is the best choice. Therefore, for the 2D case we still use the CJ-CG solver.

For the carrier continuity equations the best solvers of each class were investigated with respect to their vectorization potential: line/plane CJ-BiCG, ILU-BiCG and [M]ILU-CGS. For small size problems (e.g. the 2D case), the line CJ-BiCG ist the fastest and for this type of problems stable enough as well. However, for larger size examples it is not stable, therefore it cannot be used for 3D MINIMOS. The plane CJ-BiCG is stable but slow. For 3D problems the fastest solvers turned out to be of CGS type [3]. We have investigated and compared different preconditioning methods for CGS, including ILU[0] to ILU[2] and MILU[0].

The performance of various solvers on the VP200 as compared to Gaussian elimination is shown in Figure 2.

Figure 2: Performance on VP200 of several solvers for nonsymmetric matrices relative to Gaussian elimination.



Tuning programs is an infinite sink, and though progress is still possible the gain saturates. Moreover, the MINIMOS program is tailored to solve specific problems of device simulation for design engineers and we have to look how the overall performance has improved. For demonstration we have selected two examples of rather low complexity (see Table 6). One N-channel MOSFET with channel length and width of about one micron and a gate oxide thickness of 0.0150 microns and one P-channel MOSFET with similar dimensions, but a gate oxide thickness of 0.08 microns. Bias conditions were U $= U$ $=$ 3V for the NMOSFET example and $U_{DS} = -1$V, $U_{GS} = -4$V, and $U_{BS} = 2$V for the PMOSFET example. Test matrices obtained by the NMOSFET example had a total of $22 \times 36 \times 19$ grid points for the Poisson equation and $22 \times 21 \times 19$ for the continuity equations.

Table 6: CPU-Time/s for one Bias Condition on Different Computers.

Device	VP200	CRAY-2	ALLIANT
PMOSFET	28.87	72.92	718.2
NMOSFET	30.32	55.77	629.5

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.

Usually the gain achieved by vectorization is expressed in terms of the speed-up. For the VP200 the speed-up for the NMOSFET-Example is 13, the overall vectorization rate is 96 %.

A straightforward analysis of the CPU time consumption shows that tuning the solvers further only has 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.

Acknowledgement

We are indebted to 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 and to M. Schubert and H.-P. Falkenburger for their aid on performing tests on the CRAY-2 at the RUS Stuttgart.

References

[1] S. Selberherr, "The Status of MINIMOS," in: *Simulation of Semiconductor Devices and Processes*, edited by: K. Board, D.R.J. Owen, ISBN 0-906674-59-X, pp. 2–15, 1986.

[2] M. Thurner, P. Lindorfer, S. Selberherr, "Numerical Treatment of Nonrectangular Field-Oxide for 3D MOSFET Simulation," *SISDEP Proc.*, pp. 375–381, Montpellier, 1988.

[3] O. Heinreichsberger, S. Selberherr, M. Stiftinger, and K. Traar, "Fast Iterative Solution of Carrier Continuity Equations in 3D MOS/MESFET Simulations," *SIAM J. Sci. Stat. Comput.*, to be published.

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

[5] H. K. Gummel, "A Self-Consistent Iterative Scheme for One-Dimensional Steady State Transistor Calculations," *IEEE Trans. Electron. Devices*, ED-11, pp. 455–465, 1964.

[6] S. Selberherr et al., "MINIMOS 5.1 Users Guide," Institut für Mikroelektronik der Technischen Universität Wien, 1989.

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

[8] W. Hänsch and S. Selberherr, "MINIMOS 3: A MOSFET Simulator that Includes Energy Balance," *IEEE Trans. Electron. Devices*, ED-34, pp. 1074-1078, 1987.

[9] L. Hageman, and D. M. Young, "Applied Iterative Methods," New York: Academic Press Inc., 1981.

[10] J. A. Meijerink and H. A. van der Vorst, "An Iterative Solution Method for Linear System of which the Coefficient Matrix is a Symmetric M-Matrix," *Math. Comp.*, 31, pp. 148–162, 1977.

[11] D. S. Kershaw, "The Incomplete Cholesky-Conjugate Gradient Method for the Iterative Solution of Systems of Linear Equations," in *Journal of Computational Physics*, Vol. 26, pp. 43-65.

[12] D. M. Young, and K. C. Jea, "Generalized Conjugate Gradient Acceleration of Nonsymmetrizable Iterative Methods," *Linear Algebra and its Applications* 34 159–194 (1980).

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

[14] C. C. Paige, and M. A. Saunders, "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares," *ACM Transactions on Mathematical Software*, Vol. 8, No. 1, pp. 43-71, March 1982.

[15] R. Fletcher, "Conjugate Gradient Methods for Indefinite Systems," *Lecture Notes in Mathematics*, 506, Springer, Berlin, Heidelberg, New York, pp. 73–89, 1976.

[16] P. Sonneveld, "CGS, A Fast Lanczos-Type Solver for Nonsymmetric Systems," *SIAM Journal of Sci. Stat. Computing*, Vol. 10, No. 1, pp. 36–52, Jan. 1989.

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

[18] C. C. Ashcraft, R. G. Grimes, "On Vectorizing Incomplete Factorization and SSOR Preconditioners," *SIAM J. Sci. Stat. Comput.*, Vol. 9, No. 1, pp. 122–151, 1988.

[19] H. A. van der Vorst, "(M)ICCG for 2D Problems on Vectorcomputers," in *Supercomputing*, ed. A. Lichnewsky, C. Saguez, North Holland, pp. 321–333, 1987.

[20] O. Axelson and V. Eijkhout, "Robust Vectorizable Preconditioners for Three-dimensional Elliptic Difference Equations with Anisotropy," in *Special Topics in Supercomputing*, Vol. 3, North Holland, 1987.

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

[22] H. A. van der Vorst, "Vectorization of Linear Recurrence Relations", *SIAM J. Sci. Stat. Comput.*, Vol. 10, No. 1, pp. 27–35, 1989.

[23] H. A. van der Vorst, "High Performance Preconditioning," *SIAM J. Sci. Stat. Comput.*, Vol. 10, No. 6, pp. 1174–1185, Nov. 1989.

[24] H. A. van der Vorst, "Large Tridiagonal and Block Tridiagonal Linear Systems on Vector and Parallel Computers," *Parallel Computing* 5, pp. 45-54, North Holland, 1987.

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

[26] J. van Kats, Aad van der Steen, R. Llurbal, "Result of a Benchmark Test on a Siemens VP-200 Vectorprocessor with Comparisons to Other Supercomputers," in *Special Topics in Supercomputing*, Vol. 3, North Holland, 1987.

[27] S. C. Eisenstat, "Efficient Implementation of a Class of Preconditioned Conjugate Gradiend Methods," *SIAM J. Sci. Stat. Comput.*, Vol. 2, No. 1 , pp. 1–4 , 1981.

[28] H. A. van der Vorst, "The Performance of FORTRAN Implementations for Preconditioned Conjugate Gradients on Vector Computers," *Parallel Computing* 3, pp. 49-58, North Holland, 1986.

[29] SIEMENS Fortran77/VP Programming handbook.