Performance issues for iterative solvers in device simulation

Qing Fan
P. A. Forsyth [1]
J. R. F. McMacken[2]
and Wei-Pai Tang[3]

The Research Institute of Advanced Computer Science is operated by Universities Space Research Association, The American City Building, Suite 212, Columbia, MD 21044, (410) 730-2656

Work reported herein was sponsored by NASA under contract NAS 2-13721 between NASA and the Universities Space Research Association (USRA) and by the Natural Sciences and Engineering Research Council of Canada, and by the Information Technology Research Center, which is funded by the Province of Ontario. The last author is on sabbatical leave from University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

[1]Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.
[2]Goal Electronics Inc.
[3]On sabbatical leave from University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

Key words. Device simulation, iterative solver, preconditioning.

Abstract. Due to memory limitations, iterative methods have become the method of choice for large scale semiconductor device simulation. However, it is well known that these methods still suffer from reliability problems. The linear systems which appear in numerical simulation of semiconductor devices are notoriously ill-conditioned. In order to produce robust algorithms for practical problems, careful attention must be given to many implementation issues. This paper concentrates on strategies for developing robust preconditioners. In addition, effective data structures and convergence check issues are also discussed. These algorithms are compared with a standard direct sparse matrix solver on a variety of problems.

1. Introduction. The increasing demands for more accurate semiconductor device modeling have pushed the development of numerical methods to a new era. One of these new developments is the study of techniques for the solutions of very large ill-conditioned sparse linear systems. In the past, software developers favored the use of sparse direct methods. These solvers could be treated as black box modules, since a reliable solution could (almost) always be obtained. However, the memory requirements of these direct methods makes them impractical for large scale simulations. Consequently, attention has shifted to the use of iterative methods for device simulation.

During the past decade, there has been considerable interest in Krylov subspace acceleration coupled with various preconditioning techniques. A great deal of effort [2, 19, 24, 25, 26, 13] has been devoted to developing iterative acceleration methods, while *robust* preconditioners have received less attention, mainly because of inherent theoretical difficulties. However, a good preconditioner is necessary for a robust iterative algorithm. Many workers have observed that a superior preconditioner can boost performance by an order of magnitude[27], while a better acceleration technique may only improve the performance by $10-30\%$. In [4, 24], various new developments in iterative methods for device simulation have been summarized.

Although iterative methods seem to be the only practical choice for large scale simulations, direct methods are still used in many situations. This is because many existing iterative methods may fail to converge when the matrix is very ill-conditioned and careful attention is not given to many implementation issues.

The objective of this article is to investigate several issues which are important for the performance of iterative methods in device simulation. In particular, our emphasis will be on construction of an effective preconditioner. Both level based [ ] and drop tolerance [ ] preconditioners will be tested. A new two step preconditioner, which treats the electric potential terms in the Jacobian in a very accurate manner, will also be developed. Simple block scaling [ ], which has been suggested as a preconditioner for device simulation problems, will also be compared to the above methods. Some other issues which will also be addressed include the ordering of the unknowns [ ], choice of acceleration method, and the convergence check criteria. All these methods were tested in a commercially available drift diffusion device simulator [ ] which uses full Newton iteration for solution of the nonlinear algebraic equations.

2. CHORDV and test problems. CHORD System V[ref JRF McMacken, SG Chamberlain, CHORD: A modular semiconductor device simulation development tool incorporating external network models, IEEE Trans. CAD, v8, n8, p826-836, 1989] is a semiconductor device simulator which uses fully coupled Newton iteration to solve a variety of carrier transport models. In this paper, we are concerned with the traditonal two- carrier, drift-diffusion equations: Poisson's equation and the electron and hole current continuity equations.

$$\nabla^2\psi + \frac{q}{\epsilon}(p - n + N_D - N_A) = 0$$

$$\frac{\partial n}{\partial t} = \frac{1}{q}\nabla \cdot J_n - R$$

$$-\frac{\partial p}{\partial t} = \frac{1}{q}\nabla \cdot J_p - R$$

Here, ψ is the electrostatic potential, p, n the hole and electron concentrations, J_n, J_p the corresponding current densities, N_D, N_A the ionized donor and acceptor concentrations and R the net recombination. Using the drift-diffusion approximation, we can write the electron and hole currents as

$$J_n = -q\mu_n n \nabla \psi + qD_n \nabla n$$

$$J_p = -q\mu_p p \nabla \psi - qD_p \nabla p$$

where μ_n, μ_p and D_n, D_p are the carrier mobility and diffusion coefficients. These expressions are combined to yield a system of three equations in three unknowns (ψ,n,p). Our carrier mobility models are taken from Nishida and Sah[ref T. Nishida, C-T. Sah, A physically based mobility model for MOSFET numerical simulation, IEEE Trans ED, vED-34, n2, p310-320, 1987] and includes components due to lattice vibration, ionized impurities, surface scattering and velocity saturation. The recombination term includes Shockley-Read-Hall, Auger and impact ionization. We convert the diffusion coefficients to effective mobilities using the Einstein relation.

The three equations are discretized across a two-dimensional domain using box integration[ref CS Rafferty, MR Pinto, RW Dutton, Iterative methods in device simulation, IEEE Trans. CAD, vCAD-4, n4, p462-471, 1985] applied to a non-orthogonal grid. Consider the grid node i and its neighbours j, j+1, j+2, etc. shown in Figure[Figure C.1 from my thesis]. In the box integration method, we begin by constructing perpendicular bisectors of the grid to develop a control volume. Note that this approach restricts us to using grid meshes which do not contain obtuse angles.

Next we assume that the electric field and current are constant across each face of the polygon as well as all physical properties. Using a simple difference form for the field, Poisson's equation becomes

$$\sum_{j=1}^{n} \left[\frac{h_j^+ + h_j^-}{l_j}\right] [\psi_j - \psi_i] + \frac{q}{\epsilon}[p_i - n_i + N_{Di} - N_{Ai}] \sum_{j=1}^{n} \left[a_j^+ + a_j^-\right] = 0$$

To discretize the continuity equations we need a difference form for the partial derivatives over time. We use the simple backward Euler expression

$$\frac{\partial f}{\partial t} = \frac{f_{k+1} - f_k}{\delta t}$$

Thus, the electron continuity equation becomes

$$\frac{1}{q}\sum_{j=1}^{n} \left[h_j^+ + h_j^-\right] J_{njk+1} - \left[R_{ik+1} + \frac{n_{ik+1} - n_{ik}}{\delta t}\right] \sum_{j=1}^{n} \left[a_j^+ + a_j^-\right] = 0$$

The Scharfetter-Gummel approximation for current density[ref DL Scharfetter, HK Gummel, Large signal analysis of a silicon Read diode oscillator, IEEE Trans. ED, vED-16, n1, p64-77, 1969] is given by

$$J_{njk+1} = -q\frac{\phi_T \mu_n}{l_j}\left[n_{ik+1}B(-\delta\psi_{jk+1}/\phi_T) - n_{jk+1}B(\delta\psi_{jk+1}/\phi_T)\right]$$

where $B(x)$ is the Bernoulli function

$$B(x) = \frac{x}{\exp x - 1}$$

and ϕ_T is the thermal voltage. Thus, the resulting discrete form of the electron continuity equation is

$$\frac{1}{q}\sum_{j=1}^{n}\left[h_j^+ + h_j^-\right] - q\frac{\phi_T \mu_n}{l_j}\left[n_{ik+1}B(-\delta\psi_{jk+1}/\phi_T) - n_{jk+1}B(\delta\psi_{jk+1}/\phi_T)\right] -$$

$$\left[R_{ik+1} + \frac{n_{ik+1} - n_{ik}}{\delta t}\right]\sum_{j=1}^{n}\left[a_j^+ + a_j^-\right] = 0$$

A similar expression may be developed for the hole continuity equation.

In CHORD, the transport model is solved using Newton iteration. Given a set of equations with residuals $r_k, k = 1..n$, we linearize the system about a point x_k and solve the linear system $J_k \Delta x_k = -r(x_k)$ where J_k is the Jacobian matrix formed by computing partial derivatives of r_k. Our solution estimate is then updated by the Newton step Δx and the process repeated until the system is converged. In this paper, we will focus on iterative methods of solving the linear system.

Two typical semiconductor models: Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) and bipolar junction transistor (BJT) are used for testing purposes. The n-channel MOSFET device is a simplified self-aligned n-channel MOSFET with a 2um drawn channel length and a 25nm thick gate insulator (see Figure 2.1). The source and drain are 0.25um abrupt junctions in a lightly doped p-type substrate (5.0e+15 cm-3). There is no channel implant. We assume an ideal structure with no oxide charge or interface charge. The MOSFET problem has 15587 unknowns.

FIG. 2.1. *An n-channel MOSFET*

The second device is a bipolar junction transistor (see Figure 3.1) that is an active three-terminal device which can be used as an amplifier or switch. There are areas of applications in which the bipolar transistor is superior to MOSFET, such that, in high-power devices and in high-speed logics for high-performance computers. This device is a simple vertical *npn* transistor formed by two ion-implant steps and a thermal anneal, an n+ buried layer is used to reduce the parasitic collector resistance. The BJT problem has 13758 unknowns.

TEST PROBLEM 3???????

Each test problem for a given device model consists of a sequence of subrproblems for a gradually increasing value of source-drain voltage. Each of these subproblems requires many Newton steps, which in turn requires many Newton solves. Consequently, each test problem exercises the matrix solution algorithm over many different Newton iterations and values if the source-drain voltage. Consequently, the total time for the entire test problem is a good indication of the robustness and reliability of the matrix solver over a wide range of situations.

The Newton iteration convergence criteria was ?????????.

3. Inner Iteration Convergence Criteria. The complete solution process in a device simulation consists of an inner and outer iteration. The outer iteration is the nonlinear Newton method. At each Newton step, an ill-conditioned linear system is solved. If the Jacobian is solved using an iterative method, then the convergence tolerance must be specified. For the first few Newton steps, the nonlinear residual in (4.2) is quite large. Therefore, a large residual reduction in the linear iteration is needed for an accurate Δx. After the Newton iterations reaches a certain accuracy, the nonlinear residual becomes smaller. The update Δx only affects the last few digits of the final solution. As a result, the relative error requirement for the solution of the Jacobian can be lowered.

Here a dynamic switch of the linear convergence criteria is implemented in the linear solver. Initially, the convergence requirement is that the intial l_2 residual be reduced by 10^{-6}. After the *nonlinear* residual is reduced below 10^{-3} (compared to the initial *nonlinear* residual, the linear residual reduction switchs to 10^{-3}. This dynamic switch can generally save 10-15% of the CPU time.

FIG. 3.1. *A bipolar junction transistor*

4. Block data structure and scaling. In Newton's method, a system of linear equations with the Jacobian is solved every iteration. Since device simulation involves a system of coupled partial differential equations, it is natural to exploit the block structure of the Jacobian in order to obtain a good preconditioner. Two methods which use this concept are the Alternate Block Factorization (ABF) [3] and the Modified Singular Perturbation (MSP) [28].

For the purposes of illustration, assume that the device model in question is a drift diffusion model having three coupled partial differential equations. If the Jacobian equations are ordered so that all the electric potential equations are grouped first, followed by the electron conservation equations, and then the hole conservation equations, and the unknowns are ordered so that all the electric potentials are first, then the electron concentrations, and finally the hole concentrations, then the Jacobian matrix can be partitioned as

$$J = \begin{bmatrix} J_{\phi\phi} & J_{\phi n} & J_{\phi p} \\ J_{n\phi} & J_{nn} & J_{np} \\ J_{p\phi} & J_{pn} & J_{pp} \end{bmatrix} \tag{4.1}$$

where J_{ij} , $i,j = \phi, n, p$ denotes the block of derivatives of the equation for electric potential, electron conservation, or hole conservation, with respect to the electric potential, electron concentration or hole concentration. Then the ABF preconditioner is

$$D^{-1} = \begin{bmatrix} D_{\phi\phi} & D_{\phi n} & D_{\phi p} \\ D_{n\phi} & D_{nn} & D_{np} \\ D_{p\phi} & D_{pn} & D_{pp} \end{bmatrix}^{-1}$$

where D_{ij} is the diagonal matrix of J_{ij}. More recently, the Approximate Block Elimination (ABE) has been proposed which uses an incomplete 3×3 block factorization of the original block structured Jacobian.

In this paper, another kind of block structure of the Jacobian matrix is used. The unknown variables at each physical grid node are grouped together to form an $n \times n$ block matrix, where n is the number of grid nodes. In general, the diagonal block elements of this block Jacobian may have different sizes since the number of unknowns for each node varies. This is due to the fact that different models are used in different device materials and at the device contacts. The motivation for explicity considering the block structure are the following:

All the nonzero blocks are regarded as dense. Consequently, we need only to specify the sparsity pattern for the nonzero blocks. As a result, the integer space needed for specifying the structure of the block sparse matrix is an order of magnitude less compared to the original scalar sparse matrix (assuming that the average number of unknowns per node is at least three). This block structure will be more attractive when the semiconductor model becomes more sophisticated, since typically, more detailed physics requires more equations per node. Our study [7] indicates that one of the basic operations in iterative methods, namely matrix vector multiplication, takes less time if a block data structure is used compared to the scalar case. It is clear that the cache hit ratio will be higher in the block case.

More importantly, the new block structure can allow us to reduce the strong coupling between unknowns associated with a single grid node before the iterative process and other preconditioning steps begin. If we order the unknowns and equations so that all equations and unknowns associateds with a node are ordered consecutively, then

$$J = \begin{bmatrix} J_{11} & J_{12} & \cdots & J_{1n} \\ J_{21} & J_{22} & \cdots & J_{2n} \\ & & \ddots & \\ J_{n1} & J_{n2} & \cdots & J_{nn} \end{bmatrix}$$

TABLE 4.1
Block scaling

Test (N)	Methods applied	Total linear iterations	Total linear CPU time
MOSFET (15583)	ABF	1723	1772.08
	Two step	73	465.04
BJT (13758)	ABF	Not converge	
	Two step	117	523.54

where J_{ii} are typically 1×1, $2 \times 2, \cdots, 7 \times 7$ block matrices. For drift-diffusion models, the maximum size of J_{ii} is usually four (some of the nodes may have the electric current as an additional unknown). At each Newton step we need to solve the equation

$$J\Delta x = -r \tag{4.2}$$

where J is the Jacobian, Δx is the vector of updates, and r is the residual vector.

First, a block scaling is applied to equation(4.2). The scaling matrix

$$S = \begin{bmatrix} J_{11}^{-1} & 0 & \cdots & 0 \\ 0 & J_{22}^{-1} & \cdots & 0 \\ & & \ddots & \\ 0 & 0 & \cdots & J_{nn}^{-1} \end{bmatrix}$$

is multiplied on both sides of equation (4.2). The preconditioned Krylov space method is then applied to the scaled matrix equation

$$SJ\Delta x = Sr.$$

Note that this scaling step is equivalent to applying the ABF preconditioner if a permutation is applied. The difference here is that a further preconditioning process will be applied to the scaled system. The improvement in using a block scaling followed by further preconditioning is significant compared to using block scaling alone.

Table 4.1 shows the difference in performance if a further preconditioning step is taken. Two Jacobian matrices were generated at intermediate Newton iterations from a typical simulation. The two step preconditioner uses a block scaling followed by the best preconditioned BI-CGSTAB method which will be described in more detail in following sections. Table 4.1 shows the total number of solver iterations for all Newton iterations, as well as the total number of unknowns N. Clearly, block scaling (ABF) alone is not sufficient for a robust technique.

5. Preconditioning Issues . Preconditioned Krylov subspace methods are the standard iterative methods for device simulation. Much of the past research has concentrated on the behavior of different acceleration methods [1, 4, 5, 15, 18, 21, 22, 24].

Many authors have observed that GMRES, CGS and Bi-CGSTAB are the relatively robust choices among the many. Our experience agrees with previous work. In particular, we found that Bi-CGSTAB is generally the most robust method. For example, Figure 5.1 shows the total CPU times for problem the MOSFET problem for various values of the Drain-Source voltage. Clearly, Bi-CGSTAB is the most efficient acceleration method. Tests for other problems showed a similar trend. Consequently, all computations will be carried out using Bi-CGSTAB for the remainder of this paper.

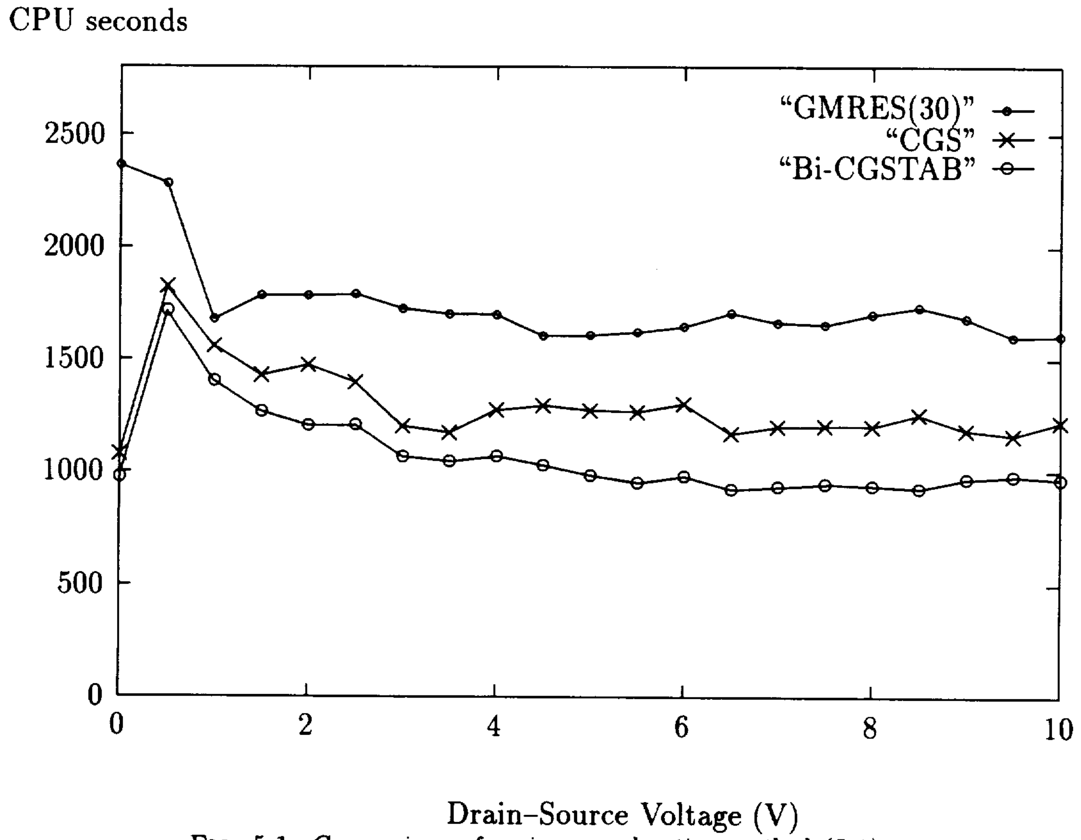

FIG. 5.1. *Comparison of various acceleration methods(5.1)*

For completeness, we give the preconditioned Bi-CGSTAB algorithm below. (LU represents the incomplete factorization of matrix A and $\overline{Aq}$ is a vector.)

```
$r_0 = b - Ax_0$
$\hat{\omega}_0 = \beta = \alpha_0 = 1$
$\overline{Aq}_0 = q_0 = 0$
For $i = 1, 2, \ldots$
    $\hat{\beta} = (r_0, r_{i-1});\quad \omega_i = (\hat{\beta}/\beta)(\bar{\omega}_{i-1}/\alpha_{i-1})$
    $\beta = \hat{\beta}$
    $q_i = r_{i-1} + \omega_i(q_{i-1} - \alpha_{i-1}\overline{Aq}_{i-1})$
    $\bar{q}_i = (LU)^{-1} q_i$
```

```
        $\overline{Aq_i} = A\bar{q}_i$
        $\hat{\omega}_i = \hat{\beta}/(r_0, \overline{Aq_i})$
        $s = r_{i-1} - \hat{\omega}_i \overline{Aq_i}$
        $\bar{s} = (LU)^{-1} s$
        $t = A\bar{s}$
        $\alpha_i = (L^{-1}t, L^{-1}s)/(L^{-1}t, L^{-1}t)$
        $x_i = x_{i-1} + \hat{\omega}\bar{q}_i + \alpha_i \bar{s}$
        if $x_i$ accurate enough, then quit
        $r_i = s - \alpha_i t$
    End
```

Although a suitable acceleration scheme may improve the performance by 10-30% on average, a good preconditioner may speed up convergence by an order of magnitude [4, 27]. In this paper, we will concentrate on the issues related to incomplete LU preconditioners, since these are regarded as among the most robust for semiconductor simulations [4, 22].

Before we pursue the different issues in constructing a good preconditioner, a block graph representation of the Jacobian is useful. If we view the grid node v_i as the node of a graph representing the block sparse matrix and map the nonzero block element J_{ij} to an edge connecting the two nodes v_i, v_j, the block sparse Jacobian

$$J = \begin{bmatrix} J_{11} & J_{12} & \cdots & J_{1n} \\ J_{21} & J_{22} & \cdots & J_{2n} \\ & & \ddots & \\ J_{n1} & J_{n2} & \cdots & J_{nn} \end{bmatrix} \tag{5.1}$$

can be represented by a graph form. During the factorization process, new nonzero block elements will be introduced. A notion of "fill level" can be introduced to each position of the block matrix. We define initially

$$\text{level}_{ij}^{(0)} = \begin{cases} 0, & \text{if } J_{ij} \neq 0, \\ \infty, & \text{otherwise.} \end{cases} \tag{5.2}$$

The nonzero block elements then have a fill level 0. As the elimination proceeds, fill will be introduced. At the k^{th} step of the elimination, the fill levels are given by

$$\text{level}_{ij}^{(k)} := \min\left(\text{level}_{ik}^{(k-1)} + \text{level}_{kj}^{(k-1)} + 1, \text{level}_{ij}^{(k-1)}\right).$$

In other words, the fill level of a fill is the length of the shortest path between node v_i and v_j through the nodes eliminated before v_i and v_j [20]. The fill level is commonly used to decide the sparsity pattern of an ILU factorization.

5.1. Ordering. The matrix ordering affects the computational efficiency of a matrix solver in many ways. For direct methods, a good ordering technique is essential in order to minimize the amount of fill. In parallel (or vector) processing, ordering again plays a crucial rule. A number of studies have examined the effect of matrix ordering on the quality of preconditioners for iterative methods based on an incomplete factorization [6, 10, 11, 16, 17, 23, 14]. In [10, 11, 12] evidence was presented to demonstrate that matrix ordering can have a profound effect on the quality of preconditioners. A heuristic method was developed that was shown to produce good matrix ordering. Unfortunately, some of the techniques for an effective ordering developed in [10, 11, 12] can only be applied to scalar sparse matrices. The ordering generated by these new algorithms may destroy the block pattern we employed here. However, the graph based orderings (where we view each block of the Jacobian as a node in the graph) can handle the block matrix (5.1) naturally.

The algorithm for generating a scalar ILU preconditioner can be directly extended to the block case. The block incomplete LU factorization can then be interpreted as a graph elimination process[20]. CHORDV uses a box integration method for the discretization of the partial differential equations. The graph representation of (5.1) is in fact the graph of the grid used to discretize the device. Note that most of the grid generation algorithms in semiconductor simulation involve some kind of refinement process. Consequently, the ordering of the grid nodes can be very scattered. It is well known that a scattered or random ordering is very effective for ILU methods [ ]. In addition, after the refinement the resulting grid is usually unstructured. Consequently, there is no obvious natural way to order the grid nodes. The Reverse Cuthill-McKee (RCM) ordering [20] originally was proposed as a good technique for reducing the profile of a sparse matrix. The basic idea of the ordering algorithm is to construct the level set from a starting node of the graph representing the sparse matrix. The reverse order of the level set will produce a small profile. We can also view the RCM ordering as a generalized natural ordering of an unstructured grid. For a rectangular grid, RCM ordering is the diagonal ordering of the mesh , if a corner node is chosen as the starting node. This ordering algorithm can be naturally generalized to the block Jacobian case.

RCM ordering is known as an effective ordering [16, 17] for ILU preconditioning in many applications. The standard RCM ordering can produce a poor ordering (for an iterative solver) if the problem is anisotropic. A revised version for a weighted graph can alleviate this problem [8]. A comparison of an ILU method which uses the original ordering and the RCM ordering was carried out for Test Problem 3. WHERE IS THIS PROBLEM DESCRIBED???? Figure 5.2 give the total CPU time required to obtain the steady state solution for various values of the Drain-Source Voltage for Test Problem 3, using the original ordering (ORG), RCM ordering (RCM), and, for comparison, the time for a direct solver (SPARSPAK) [ ] is also shown. Her,e an ILU(0) preconditioner is used with Bi-CGSTAB acceleration. The three curves On average, RCM ordering reduces the total CPU time by about 30-40% compared to the original ordering. Consequently, in the following, RCM ordering will be used unless otherwise noted.

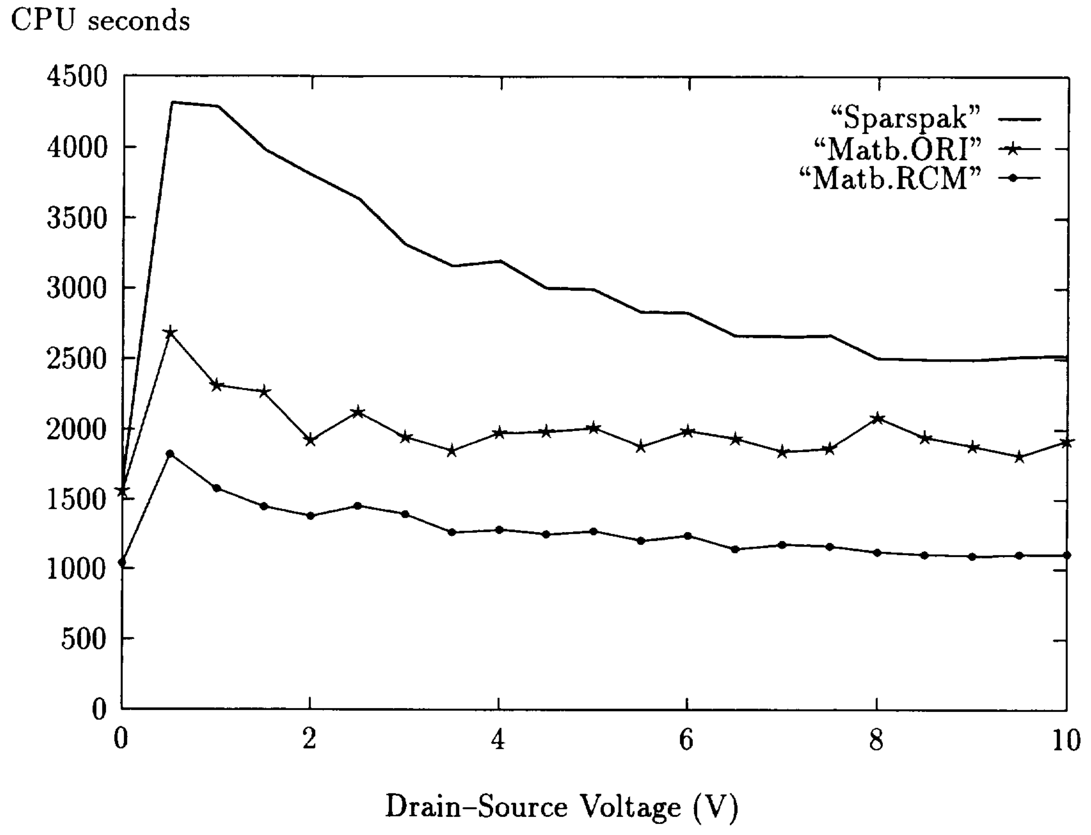

FIG. 5.2. *Comparison of direct solver (Sparspak) and iterative solver (Matb), RCM ordering (Matb.RCM) and ORI ordering (Matb.ORI). ILU(0) preconditioner and Bi-CGSTAB acceleration used in the iterative solver.*

5.2. Sparsity pattern. The key step during the incomplete LUfactorization process is to determine the sparsity pattern of the L and U. To find the optimal sparsity pattern for the ILU factorization is a much more difficult task than the solution of the Jacobian itself. Typically, some simple heuristics are used for determine if a fill element will be discarded. The common strategies are:

1. **By a drop tolerance,** $ILU(\varepsilon)$.

   Let $J_{ij}^{(k)}$ be the submatrix which remains after $(k-1)$ steps of (incomplete) elimination [????]. The drop tolerance method discards fill if

$$|J_{ij}^{(k)}| < \varepsilon\sqrt{|J_{ii}^{(k)}J_{jj}^{(k)}|}. \tag{5.3}$$

   There are many possible drop criteria which can be used [ ]. We will use the criteria (5.3) since it is similar to that used in [ ]. Note that since the Jacobian matrix is not symmetric positive definite, we do not use the diagonal modification suggested in [ ]. Even for symmetric positive definite problems, the diagonal modification usually results in a slow method [ ].

   It is probably not a good idea to extend the drop tolerance approach to the block case. First, the computation of the norm of the block element J_{ij} is not an inexpensive operation. As well, the coefficients in each block of the Jacobian

TABLE 5.1
Drop Tolerance Compared to Two Step

Test (N)	Methods applied	Total linear iterations	Total linear CPU time	Double workspace
MOSFET (15583)	Two step	73	465.04	1,087,806
	$\varepsilon = 0.5$	741	2426.10	660,080
	$\varepsilon = 0.01$	660	2370.84	729,729
	$\varepsilon = 0.001$	416	2508.34	1,141,275
	$\varepsilon = 0.0001$	363	4071.69	1,784,119
BJT (13758)	Two step	117	523.54	987,151
	$\varepsilon = 0.5$	1439	3700.06	534,994
	$\varepsilon = 0.01$	1226	3320.64	572,442
	$\varepsilon = 0.001$	1029	3505.31	772,080
	$\varepsilon = 0.0001$	579	2980.67	1,108,358

in device simulation often vary by many orders of magnitude ($10^{10} - 10^{16}$). One large entry in a block element may result in keeping the entire fill block, which may not be desirable. Consequently, the drop tolerance method will be applied to each individual element in the Jacobian matrix in the following. A similar approach was used in [ ].

In general, we have found that a drop tolerance incomplete factorization is not an effective technique for semiconductor device simulations. The characteristics of the Jacobian in this particular application cause problems for a drop tolerance approach. It is well known that the Jacobian in device simulation is extremely ill conditioned. In other words, a small perturbation in the factorization may cause a large change in preconditioner. The basic idea of the drop tolerance heuristic becomes questionable in this case. Many experiments have supported this observation [9]. It often happens that a smaller tolerance may result a "worse" preconditioner.

The drop tolerance method (applied with criteria (5.3) to each element of the Jacobian) was compared with use of a two stage method (to be described in the following Section). Table 5.1 shows the total CPU time required for the matrix solve, the number or iterations, and the storage required (for a single value of the source-drain voltage), for various values of the drop tolerance ε. For the drop tolerance method, the total CPU times do not decrease monotonically as the drop tolerance is decreased, which is somehwat disturbing. The drop tolerance preconditioner is between six and ten times slower than the Two step preconditioner. It is clear that, for the same amount of storage, the Two step method is far superior to the drop tolerance approach.

2. **By fill level,** $ILU(\ell)$.

The fill block $J_{ij}^{(k)}$ at k^{th} step of the factorization will be discarded, if

$$\text{level}_{ij}^{(k)} > \ell.$$

It is clear that the larger the ℓ the better the preconditioner. When $\ell = n$, a complete factorization is obtained. However, large values of ℓ will be too expensive in terms of storage for 3-D problems. Our experiments indicate that the performance (in terms of total CPU time) stops improving after $\ell = 2$ even for 2-D problems.

For the level approach, only one symbolic factorization is needed for the entire simulation as long as the grid does not change. The sparsity patterns of the incomplete factorization are the same for different Newton steps or time steps. As a result, the numerical and symbolic factorization can be separated to make the factorization process more efficient. This contrasts with the drop tolerance incomplete factorization, since a different sparsity pattern will result when the Jacobian changes. Therefore, the incomplete factorization process is more expensive for a drop tolerance preconditioner. The level approach is not only easy to implement, it is even more efficient for the block Jacobian case. Indeed, the symbolic factorization phase in this case costs only a very small fraction of symbolic factorization cost if the Jacobian was not considered as a block matrix.

Table 5.2 shows the total CPU times for Test Problem ???? for various levels of fill of the ILU. The improvement from level zero to level one is significant. However, the improvement in going to levels higher than one is marginal. We also list a detailed record of a typical simulation in Table 5.2 (for a single value of the source-drain voltage). We can see that the number of iterations is monotone decreasing as the level increase. However, the amount of fill for the preconditioner becomes larger as the level of fill increases. Therefore, the higher cost for each iteration will eventually outweigh the reduction in number of iterations.

TABLE 5.2
Comparison of Levels 0, 1, 2, 3 and 4.

Method	Newton iter #	Linear iter #	Total CPU linear time	Total CPU time	Double workspace
Level(0)	13	178	701.33	898.18	830998
Level(1)	12	111	510.02	681.79	923823
Level(2)	12	85	485.42	670.90	1087806
Level(3)	12	83	558.10	729.52	1264461
Level(4)	12	70	580.68	768.87	1437259
Direct Method	12			1870.62	2529297

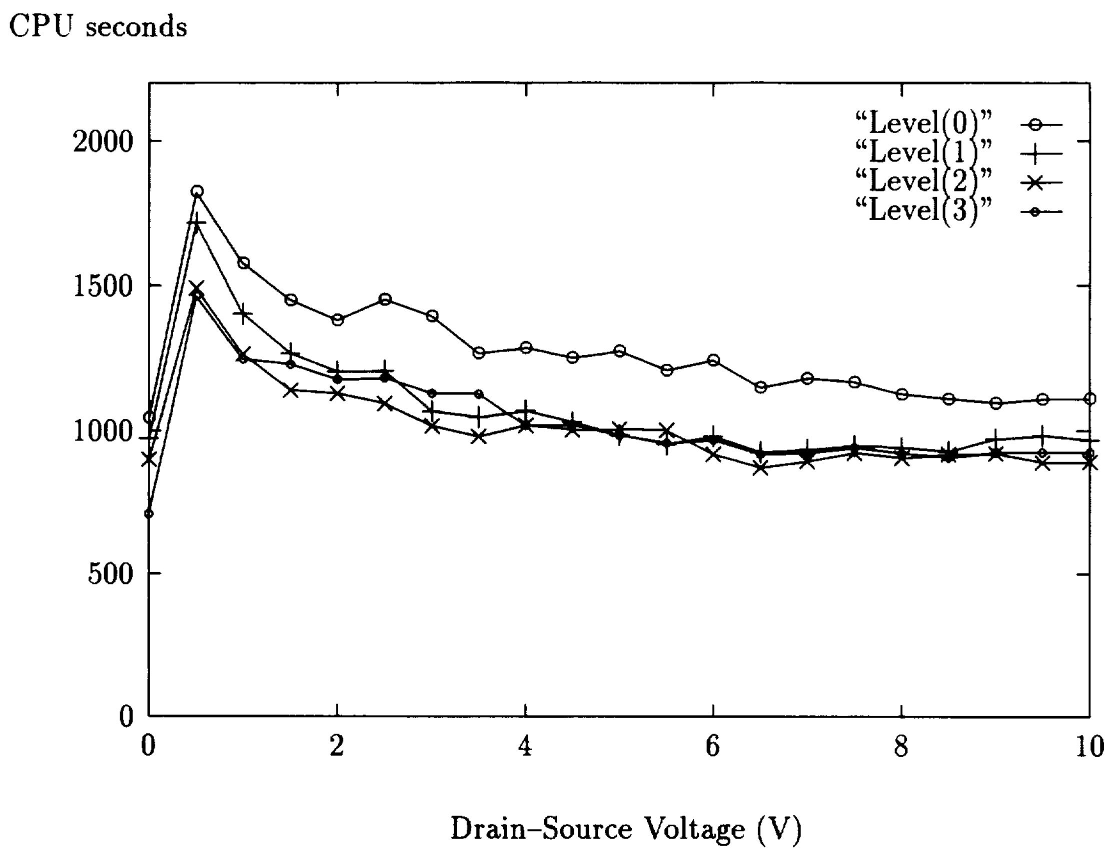

FIG. 5.3. *Comparison of Level 0, 1, 2 and 3 using one-step preconditioner in Bi-CGSTAB acceleration and RCM ordering.*

3. **By the combination of both,** $ILU(\ell,\varepsilon)$.
   A fill entry is dropped if

$$\text{level}_{ij}^{(k)} > \ell \quad \text{or} \quad |J_{ij}| < \varepsilon..$$

   This is an useful heuristic for many applications [11]. However, since the drop tolerance approach by itself does not appear to be very useful in this application, we will not pursue this method further.

5.3. Two step preconditioner. As we can see from the experiments of the previous sections, when the accuracy of the factorization (by using a higher level fill or a smaller drop tolerance) improves, the number of iterations required for convergence decreases. However, the improvement in the number of iterations will not compensate for the higher cost of each iteration after a certain point. The best combination of the strategies thus far is to use RCM ordering plus ILU(1) or ILU(2). For 3-D problem, the ILU(2) approach may require too much storage. In the following, we will use an ILU(1) factorization unless otherwise noted.

Let L_J and U_J be the incomplete factorization of the Jacobian matrix J. The techniques used in the previous sections are attempt to reduce the difference

$$\|J - L_J U_J\|.$$

The basic assumption of this method is that if this difference is small, then the solution y of the preconditioning step

$$L_J U_J y = p \tag{5.4}$$

will be close to the solution of

$$Jx = p.$$

Alternatively, we can use the following criteria for producing an effective preconditioner. Denote M_j a preconditioner of J and y the solution of

$$M_j y = p.$$

A better preconditioner M_j means a smaller residual of

$$r = Jy - p. \tag{5.5}$$

When $r = 0$, a perfect preconditioner is obtained. Therefore a refined preconditioner or a two step preconditioner can be developed as follows.

For the purposes of illustration, assume that the variables are ordered as in (4.1). After we solve the equation (5.4), the residual r in (5.5) is calculated. Let $J_{\phi\phi}$ be the diagonal block element for the potential variable in (4.1) and r_ϕ be the part of residual in (5.5) corresponding to the electric potential equations. The equation

$$J_{\phi\phi} \Delta y_\phi = r_\phi. \tag{5.6}$$

is solved. This problem is much smaller and better conditioned. Let $\Delta y = [\Delta y_\phi, 0, 0]^T$, i.e. only the potential variables become updated. Then, the refined solution $\tilde{y} = y - \Delta y$ is used as the solution for the new two step preconditioner M_t such that

$$M_t \tilde{y} = p.$$

It is easy to see that the new residual

$$\tilde{r} = J\tilde{y} - p$$

will have

$$\tilde{r}_\phi = 0.$$

In another words, the two step preconditioner has more accurate electric potential solution. Consider a case where the drift flow of holes and electrons dominates the diffusion flux. In this case, as the mesh size is reduced, the electric potential derivatives in the Jacobian will dominate the hole and electron concentration derivatives. Essentially, this is because in this situation, the electric potential is a elliptic type variable, while the hole and electron concentrations are hyperbolic type variables.

For the range of two dimensional problems we have tested so far, we have found that us of a direct solve of equation (5.6) is quite efficient. In other words, $J_{\phi\phi}$ is factored once, and equation (5.6) is solved by a forward and back solve each iteration. We use minimum degree ordering [ ] for the initial complete factorization of $J_{\phi\phi}$ (this system is much smaller than the original Jacobian). Of course, for larger 3-D problems, it may be more efficient to use an iterative method to solve equation 5.6.

Note that the two step method is similar to the Combinative technique used in [ ]. Tables 5.3, and 5.4 present some detailed comparisons between the one-step and two step preconditioners for Test Problem ?????. We can see that the total number of linear iterations is reduced significantly with the two step preconditioner. Fig. 5.4 lists the CPU time comparison for a complete test run. Of course, the new preconditioner is more expensive than the single step preconditioner. However, the larger reduction in the number of iterations compensates the extra cost in each iteration.

In Table 5.4, we present a detailed performance comparison between many different techniques (Note that the CPU clock is only accurate to within 5%). It is clear that a careful implementation of the preconditioned Krylov space method is very important for performance. Although some extra memory is needed for the small system (5.6) (see the comparison in Table 5.3), the total memory requirement for the two step preconditioner is still competitive with a direct method.

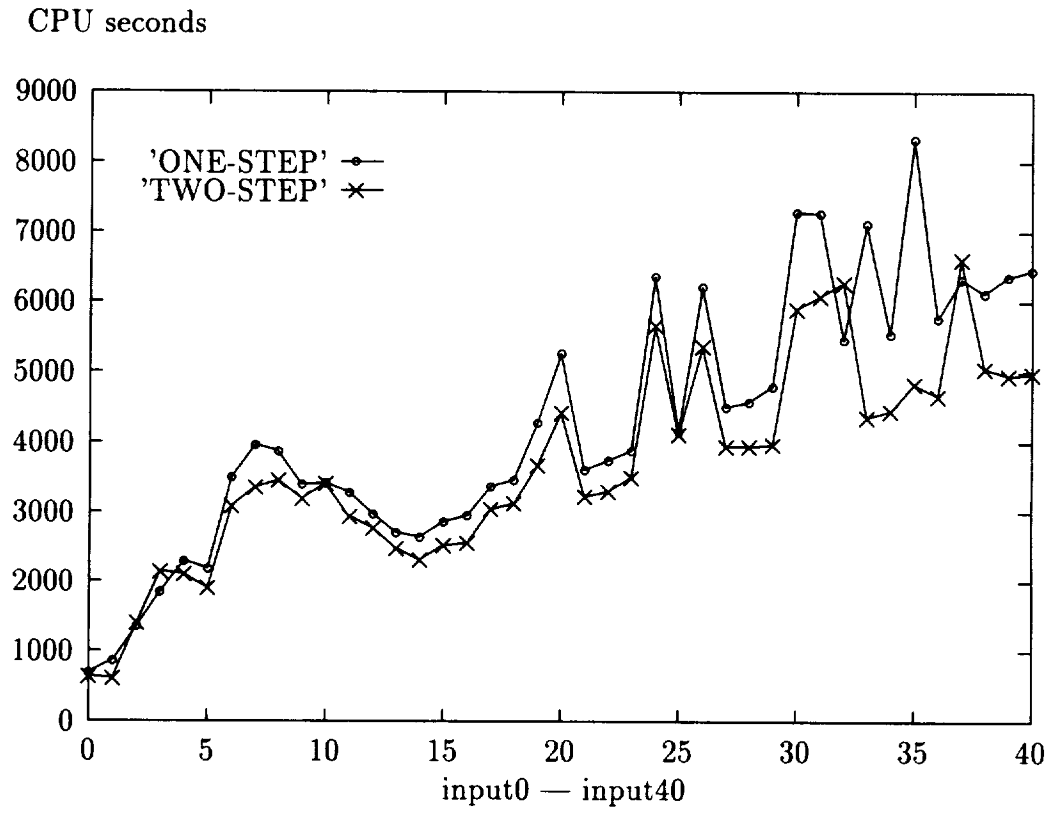

FIG. 5.4. *Comparison of one-step and two-step preconditioners.*

TABLE 5.3
Comparison of one-step and two-step preconditioner in Bi-CGSTAB

Test problem	Bi-CGSTAB (level:step)	Newton iter #	Linear iter #	Total CPU linear time	Total CPU time
problem 1	Bi-CGSTAB(1:1)	20	459	1051.62	1360.65
	Bi-CGSTAB(1:2)	20	260	1091.46	1407.89
	SPARS	20			3039.95
problem 2	Bi-CGSTAB(1:1)	31	1468	3014.96	3494.40
	Bi-CGSTAB(1:2)	31	729	2600.20	3077.40
	SPARS	31			4701.80
problem 3	Bi-CGSTAB(1:1)	48	2663	5489.89	6222.92
	Bi-CGSTAB(1:2)	48	1219	4585.40	5369.92
	SPARS	48			7348.77
problem 4	Bi-CGSTAB(1:1)	50	2833	5704.77	6461.42
	Bi-CGSTAB(1:2)	50	1194	4222.25	4985.08
	SPARS	50			7883.16
problem 5	Bi-CGSTAB(1:1)	57	3203	6397.02	7279.58
	Bi-CGSTAB(1:2)	57	1427	5050.29	5899.39
	SPARS	57			8809.89

6. Conclusion.

REFERENCES

[1] J. M. C. AARDEN AND K.-E. KARLSSON, *Preconditioned CG-type methods for solving the coupled system of fundamental semiconductor equations*, Bit, 29 (1989), pp. 916–937.

[2] O. AXELSSON, *A survey of preconditioned iterative methods for linear systems of equations*, BIT, 25 (1985), pp. 166–187.

[3] R. E. BANK, T. F. CHAN, W. M. COUGHRAN, AND R. K. SMITH, *The alternate block factorization procedure for systems of partial differential equations*, BIT, 29 (1989), pp. 938–954.

[4] R. E. BANK, W. COUGHRAN, R. S. M.A. DRISCOLL, AND W. FICHTNER, *Iterative methods in semiconductor device simulation*, Comp. Phy. Comm., 53 (1989), pp. 201–212.

[5] G. BRUSSINO AND V. SONNAD, *A comparison of direct and preconditioned iterative techniques for sparse unsymmetric systems of linear equations*, Internation Journal on Numerical Methods in Engineering, 28 (1989), pp. 801–815.

[6] P. CHIN, E. F. D'AZEVEDO, P. A. FORSYTH, AND W.-P. TANG, *Preconditioned conjugate gradient methods for the incompressible Navier-Stokes equations*, International Journal for Numerical Methods in Fluids, 15 (1992), pp. 273–295.

[7] E. CHOW AND W.-P. TANG, *Storage schemes for sparse matrices with variable block size*, workshop report, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1., April 1993.

[8] S. S. CLIFT AND W.-P. TAN, *Weighted graph based ordering techniques for preconditioned conjugate gradient methods*, Technical report CS-93-41, University of Waterloo, August 1993.

[9] E. F. D'AZEVEDO, P. A. FORSYTH, AND W.-P. TANG, *Drop tolerance preconditioning for incompressible viscous flow*, Inter. J. Computer Math, 44 (1992), pp. 301–312.

[10] E. F. D'AZEVEDO, P. A. FORSYTH, AND W. P. TANG, *Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems*, SIAM Journal On Matrix Analysis and Applications, 13 (1992), pp. 944–961.

TABLE 5.4

A performance comparison between different implementations

Test (neq)	Methods applied	Linear iter #	Total CPU linear time	Total CPU time	Double workspace
First device (15583)	Two step	81	411.70	597.00	970423
	Level 2	218	621.27	786.00	741208
	Level 1	292	702.52	875.00	577225
	AFB	1723	1697.67	1877.00	484400
	ILU(0.5)	741	2426.10	2579.00	660080
	ILU(0.01)	660	2370.84	2523.00	729729
	ILU(0.001)	416	2508.34	2654.00	1141275
	ILU(0.0001)	363	4071.69	4226.00	1784119
Second device (13758)	Two step	142	544.58	696.00	837372
	Level 2	242	612.72	764.00	673030
	Level 1	322	652.69	827.00	523251
	AFB	**			437217
	ILU(0.5)	1439	3700.06	3847.00	534994
	ILU(0.01)	1226	3320.64	3465.00	572442
	ILU(0.001)	1029	3505.31	3645.00	772080
	ILU(0.0001)	579	2980.67	3123.00	1108358

[11] E. F. D'AZEVEDO, P. A. FORSYTH, AND W.-P. TANG, *Towards a cost-effective ILU preconditioner with high level fill*, BIT, 32 (1992), pp. 442–463.

[12] ——, *Two variants of minimum discarded fill ordering*, in Proc. IMACS Intern. Symp. on Iterative Methods in Linear Algebra, R. Beauwens and P. de Groen, eds., North-Holland, 1992, pp. 603–612.

[13] H. A. V. DER VORST, *Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM Journal On Scientific and Statistical Computing, 13 (1992), pp. 631–644.

[14] S. DOI AND A. LICHNEWSKY, *A graph-theory approach for analyzing the effects of ordering on ILU preconditioning*, Research report 1452, INIA, Le Chesnay Cedex, France, June 1991.

[15] M. DRIESSEN AND H. A. V. DER VORST, *Bi-CGSTAB in semiconductor modelling*, Simulation of Semiconductor Devices and Processes, 4 (1991), pp. 45–54.

[16] I. S. DUFF AND G. A. MEURANT, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–657.

[17] L. C. DUTTO, *The effect of ordering on preconditioned GMRES algorithm, for solving the compressible Navier-Stokes equations*, International Journal for Numerical Methods in Engineering, (1994). to appear.

[18] C. FITZSIMONS, *An efficient implementation of Bi-CGSTAB applied to three-dimensional semiconductor device problems*, research report, Rutherford Appleton laboratory, Mathematical Software Group, Chilton, Didcot, Oxfordshire, 1992.

[19] R. W. FREUND, *A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems*, SIAM Journal On Scientific and Statistical Computing, 14 (1993), pp. 470–482.

[20] A. GEORGE AND J. W. H. LIU, *Computer solution of large sparse positive-definite systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[21] O. HEINREICHSBERGER, S. SELBERHERR, M. STIFTINGER, AND K. P. TRAAR, *Fast iterative solution of carrier continuity equations for three-dimensional device simulation*, SIAM Journal On Scientific and Statistical Computing, 13 (1992), pp. 289–306.

[22] G. HEISER, C. POMMERELL, J. WEIS, AND W. FICHTNER, *Three-dimensional numerical semiconductor device simulation: algorithms, architectures, results*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 10 (1991).

[23] Y. NOTAY, *Ordering methods for approximate factorization preconditioning*, technical report, Service de Métrologie Nucléaire, Université Libre de Bruxelles, January 1993.

[24] C. POMMERELL AND W. FICHTNER, *New developments in iterative methods for device simulation*, Simulation of Semiconductor Devices and Processes, 4 (1991), pp. 243–248.

[25] Y. SAAD, , AND M. SCHULTZ, *GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems*, SIAM Journal On Scientific and Statistical Computing, 7 (1986), pp. 856–869.

[26] P. SONNEVELD, *CGS, a fast Lanczos-type solver for nonsymmetric systems*, SIAM Journal On Scientific and Statistical Computing, 10 (1989), pp. 36–52.

[27] Z.-Y. WANG, K.-C. WU, AND R. W. DUTTON, *An approach to construct pre-conditioning matrices for block iteration of linear equations*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 11 (1992).

[28] K.-C. WU, R. F. LUCAS, Z. Y. WANG, AND R. W. DUTTON, *New approaches in a 3-d one-carrier device solver*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 8 (1989), pp. 528–537.