A SIMULATOR MODULE FOR ADVANCED EQUATION ASSEMBLING

Stephan Wagner[1], Tibor Grasser[1], Claus Fischer[*], and Siegfried Selberherr[2]

[1]Christian-Doppler-Laboratory for TCAD in Microelectronics
at the Institute for Microelectronics

[2]Institute for Microelectronics
Technical University Vienna
Gusshausstr. 27–29, A-1040 Vienna, Austria
E-mail: Wagner@iue.tuwien.ac.at

[*]Firma Dr. Claus Fischer
Gustav Fuhrichweg 24/1, A-2201 Gerasdorf bei Wien, Austria

KEYWORDS

Finite Boxes, Linear Equation Systems, Contact Current, Boundary Conditions, Interface Conditions

ABSTRACT

We present a generally applicable simulator module which provides an advanced equation assembly system. The module has been originally developed for the simulation of semiconductor devices based on the Finite Boxes discretization scheme and is currently used in the general purpose device and circuit simulator MINIMOS-NT. In general, such simulations require the solution of a specific set of nonlinear partial differential equations which are discretized on a grid. Since the resulting nonlinear problem is solved by a damped Newton algorithm the solution of a linear equation system has to be obtained at each step. The module is responsible for assembling these systems and takes several requirements of the simulation process, namely the representation of boundary conditions, physically motivated variable transformation, preelimination and numerical conditioning, into account.

INTRODUCTION

The Finite Boxes discretization method is employed in various kinds of numerical tools and simulators for fast and accurate solving of nonlinear partial differential equation (PDE) systems. The discretized problem is then usually solved by damped Newton iterations which require the solution of a linear equation system at each step. The extensibility and effectiveness of any simulator highly depends on the capabilities of its core modules responsible for handling the linear equation systems. We present an advanced equation assembly module successfully coupled to MINIMOS-NT.

MINIMOS-NT is a general purpose device and circuit simulator that has been developed at the Institute for Microelectronics for twelve years. Besides the basic semiconductor equations (Selberherr 1984), several different types of transport equations can be solved. Among these are the hydrodynamic equations which capture hot-carrier transport (Stratton 1962, Bløtekjær 1970), the lattice heat flow equation to cover thermal effects like self-heating (Wachutka 1990), and the circuit equations to connect single devices to a circuit (Grasser and Selberherr 2001), both electrically and thermally. Furthermore, various interface and boundary conditions are taken care of, which include Ohmic and Schottky contacts, thermionic field emission over and tunneling through various kinds of barriers. This demands a sophisticated system handling the equation assembly, in order to keep the simulator design flexible. To implement such a system, the requirements will be identified and generalized.

A crucial aspect is also the requirement of assembling and solving complex-valued linear equation systems. For that reason the module is able to handle both real-valued and complex-valued contributions and systems.

THE ANALYTICAL PROBLEM

In order to analyze the electronic properties of an arbitrary semiconductor structure under all kinds of operating conditions, the effects related to the transport of charge carriers under the influence of external fields must be modeled. In MINIMOS-NT carrier transport can be treated by the drift-diffusion and the hydrodynamic transport models.

Both models are based on the semiclassical Boltzmann transport equation which is a time-dependent partial integro-differential equation in the six-dimensional phase space. By

Proceedings 15th European Simulation Symposium
Alexander Verbraeck, Vlatka Hlupic (Eds.)
(c) SCS European Council / SCS Europe BVBA, 2003
ISBN 3-936150-28-1 (book) / 3-936150-29-X (CD)

the so-called method of moments this equation can be transformed in an infinite series of equations. Keeping only the zero and first order moment equations (with proper closure assumptions) yields the basic semiconductor equations. Considering two additional moments gives the hydrodynamic model (Grasser et al. 2003).

The basic semiconductor equations, as given by VanRoosbroeck (VanRoosbroeck 1950), consist of the Poisson equation, the continuity equations for electron and holes as well as the current relations for both carrier types. The unknown quantities of this equation system are the electrostatic potential, ψ, and the electron and hole concentrations, n and p, respectively. C denotes the net concentration of the ionized dopants, ε is the dielectric permittivity of the semiconductor, and R is the net recombination rate. The heat-flow equation and thus the lattice temperature T_L is added to account for thermal effects. In the hydrodynamic case the carrier temperatures are allowed to be different from the lattice temperature, adding two more quantities, which are the electron and hole temperatures T_n and T_p.

Basically, a device structure can be divided into several segments to enable simulation of advanced heterostructures and to properly account for all conditions (which may cause very abrupt changes) at the contacts and interfaces between these segments, respectively. See Fig. 1 for an illustration of this concept. Every segment represents an independent domain D in one, two, or three dimensions where the PDEs are posed. The equations are implicitly formulated for a quantity x as $f_{(x)} = 0$ and termed control functions. In order to fully define the mathematical problem, suitable boundary conditions for contacts, interfaces, and external surfaces have to be applied.

Generally, such a system cannot be solved analytically, and the solution must be calculated by means of numerical methods. This approach normally consists of three tasks:

1. The domain D is partitioned into a finite number of subdomains D_i, in which the solution can be approximated with a desired accuracy.

2. The PDE system is approximated in each of the subdomains by algebraic equations. The unknown functions are approximated by functions with a given structure. Hence, the unknowns of the algebraic equations are approximations of the continuous solutions at discrete grid points in the domain. Thus, generally a large system of nonlinear, algebraic equations is obtained with unknowns comprised of approximations of the unknown functions at discrete points.

3. The third task is to derive a solution of the unknowns of the nonlinear algebraic system. In the best case an exact solution of this system can be obtained, which represents a good approximation of the solution of the analytically formulated problem (which cannot be solved exactly). The quality of the approximation depends on the fineness of the partitioning into subdomains as well as on the suitability of the approximating functions.

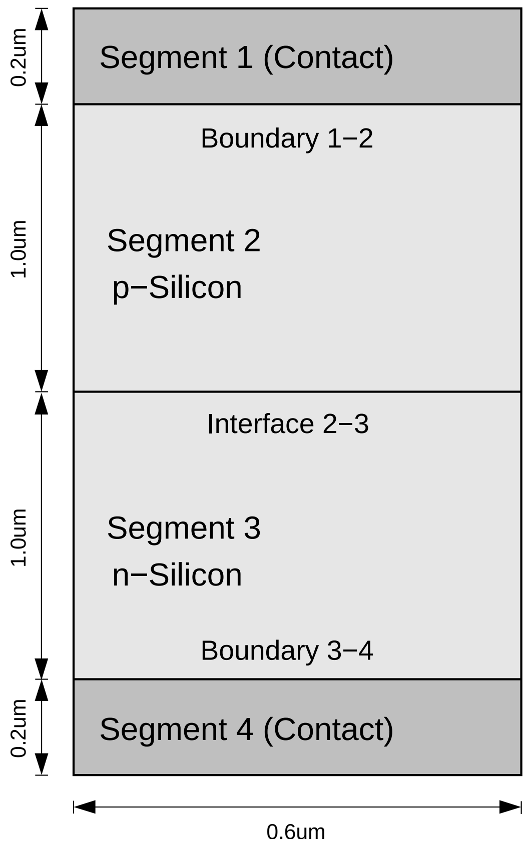

Figure 1: Illustration of the segment concept: a simple diode

For the derivation of the discrete problem several methods can be applied. We deal here with point residual methods: the finite difference method based on rectangular grids or the finite boxes (box integration) method allowing general unstructured grids. In the case of orthogonal rectangular grids both methods yield the same discretization.

DISCRETIZATION

Nonlinear partial differential equations of second order can appear in three variants: elliptic, parabolic, and hyperbolic PDEs. The Poisson equation as well as the steady-state continuity equations form a system of elliptic PDEs, whereas the heat-flow equation is parabolic. To completely determine the solution of an elliptic PDE boundary conditions have to be specified. Since parabolic and hyperbolic PDEs describe evolutionary processes, time normally is an independent variable and an initial condition is additionally required. Hence, also the transient continuity equations are parabolic.

Applying the finite boxes discretization scheme (Selberherr 1984) the equations are integrated over a control volume (subdomain, usually obtained by a Voronoi tesselation) D_i which is associated with the grid point P_i. For this grid point a general equation for the quantity x is implicitly given as

$$f^{\mathrm{S}}_{x_i} = \sum_j F_{x_{i,j}} + G_i = 0 \tag{1}$$

where j runs over all neighboring grid points in the same segment, $F_{x_{i,j}}$ is the flux between points i and j, and G_i is the source term (see Fig. 2).

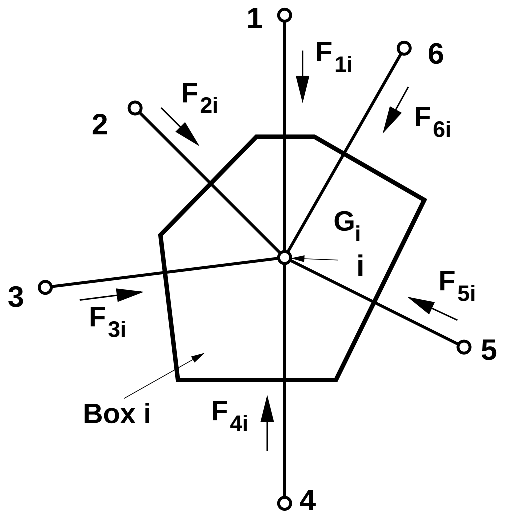

Figure 2: Box i with 6 neighbors

Grid points on the boundary $\partial\mathrm{D}$ are defined as having neighbor grid points in other segments. Thus, (1) does not represent the complete control function f_x, since all contributions of fluxes into the contact or the other segment are missing. For that reason, the information for these boxes has to be completed by taking the boundary conditions into account. Common boundary conditions are the Dirichlet condition, which specifies the solution on the boundary $\partial\mathrm{D}$, the Neumann condition, which specifies the normal derivative, and the linear combination of these conditions giving an intermediate type:

$$\mathbf{n} \cdot \mathrm{grad} x + \sigma x = \delta \tag{2}$$

Generally, the form of these conditions depends on the respective boundary models, and the conditions of which depend on the interior information. For that reason, the equation assembly is often performed in a coupled way, causing complicated modules. For instance, it is absolutely necessary to differ between interior and boundary points. Considering a general tetrahedron, there exist many kinds of boundary points (depending on the number of edges involved), which have to be treated separately. This leads to a complicated implementation of the models and can make simplifications necessary. Thus, due to organizational and implementational issues this form of coupling should be avoided.

More complex models with exponential interdependency between the solution variables such as thermionic field emission interface conditions (Schroeder 1994, Simlinger 1996) have also been implemented.

A method has been under development to implement segment models calculating the interior fluxes and their derivatives independently from the boundary models. The segment models do not have to differentiate the point type, they do not even have to care about the boundary model used. The assembly system is responsible for combining all relevant contributions by using the information given by the boundary models.

Interface Conditions

To account for complex interface conditions, grid points located at the boundary of the segments (see Fig. 3a) have three values, one for each segment (see Fig. 3b) and a third point located directly at the interface which can be used to formulate more complicated interface conditions like e.g., interface charges. However, to simplify notation these interface values will be omitted in our discussion and only the two interface points, i and i', are used.

Basically, the two (incomplete) equations $f^{\mathrm{S}}_{x_i}$ and $f^{\mathrm{S}}_{x_{i'}}$ are completed by adding the missing boundary fluxes $F_{x_{i,i'}}$:

$$f_{x_i} = f^{\mathrm{S}}_{x_i} + F_{x_{i,i'}} = 0 \tag{3}$$

$$f_{x_{i'}} = f^{\mathrm{S}}_{x_{i'}} - F_{x_{i,i'}} = 0 \tag{4}$$

The intermediate type of interfaces (2) and thus also the two other types of interfaces are generally given in linearized form by:

$$\alpha(x_i - \beta x_{i'} + \gamma) = F_{x_{i,i'}} \tag{5}$$

α, β, and γ are linearized coefficients, $F_{x_{i,i'}}$ represents the flux over the interface. The three types of interfaces differ in the magnitude of α.

In the case of an arbitrary splitting of a homogeneous region into different segments, the boundary models have to ensure that the simulation results remain unchanged. By adding (4) to (3), the box of grid point P_i can be completed and the boundary flux is eliminated. The merged box is now valid for both grid points, for that reason the respective equation can not only be used for grid point P_i, but also for $\mathrm{P}_{i'}$.

Whereas the segment models assemble the so-called segment matrix, the interface models are responsible for assembling and configuring the interface system consisting of a boundary and special-purpose transformation matrix. New equations based on (5) can be introduced into the boundary matrix without any limitations on α, thus from 0 (Neumann) to ∞ (Dirichlet). The interface models are also responsible for configuring the transformation matrix to combine the segment and boundary matrix correctly. Depending on the interface type there are two possibilities:

- Dirichlet boundaries are characterized by $\alpha \rightarrow \infty$. Thus, the implicit equation $x_i = \beta x_{i'} - \gamma$ can be used as a substitute equation. As these equations are normally not diagonally dominant they have a negative impact on the condition number and are configured to be preeliminated (see Section).
- For the other types (explicit boundary conditions) the boundary flux is simply added to the segment fluxes. In the case of a large α, the transformation matrix could be used to scale the entries by $1/\alpha$ because of the preconditioner used in the solver module.

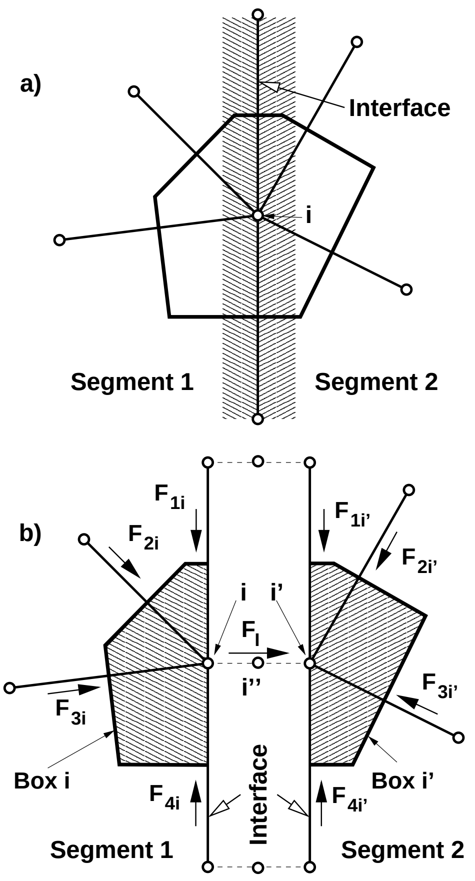

Figure 3: Splitting of interface points: Interface points as given in a) are split into three different points having the same geometrical coordinates b)

Note, that all interface-dependent information is administrated by the respective interface model only.

As an additional feature, the transformation matrix can be used to calculate several independent boundary quantities by combining the specific boundary value with the segment entries (also in the case of Dirichlet boundaries). For example, the dielectric flux over the interface is calculated as $\sum_i f^{\mathrm{S}}_{x_i}$ and introduced as a solution variable because some interface models require the cross-interface electric field strength to determine tunnel processes. Calculation of the normal electric field is thus trivial. Note that this is not the case when the normal component of the electric field $\vec{E}_n$ has to be calculated using neighboring points when unstructured two- or three-dimensional grids are used.

See Fig. 4 for an illustration of these concepts. The transformations are set up to combine the various segment contributions with the boundary system.

Boundary Conditions

Contacts are handled in a similar way to interfaces. However, in the contact segment there is only one variable available for each solution quantity (x_{C}). Note that contacts are represented by spacial multi-dimensional segments. Furthermore, all fluxes over the boundary are handled as additional solution variables F_{C} (e.g., contact charge Q_{C} for Poisson equation, contact electron current $I_{n_{\mathrm{C}}}$ for the electron continuity equations, or H_{C} as the contact heat flow).

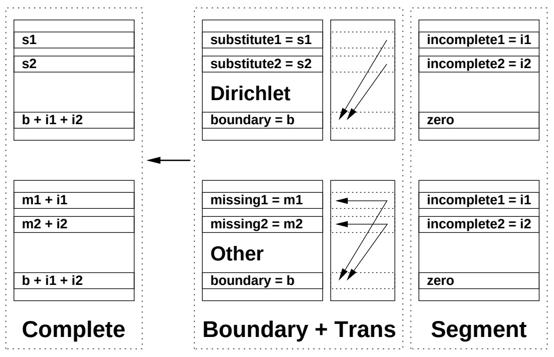

Figure 4: The complete equations are a combination of the boundary and the segment system. This combination is controlled by the transformation matrix and depends on the interface type.

For explicit boundary conditions one gets

$$\begin{aligned} f_{x_i} = \quad & f_{x_i}^{\mathrm{S}} + F_{x_i,\mathrm{C}} & = 0 \qquad (6) \\ f_{F_\mathrm{C}} = \quad & F_\mathrm{C} + \textstyle\sum_i f_{x_i}^{\mathrm{S}} & = 0 \qquad (7) \end{aligned}$$

with i running over all segment grid points. At Schottky contacts explicit boundary conditions apply. The semiconductor contact potential ψ_s is fixed and given as the difference of the metal quasi-Fermi level (which is specified by the contact voltage ψ_C) and the metal workfunction difference potential ψ_wf.

$$\psi_\mathrm{s} = \psi_\mathrm{C} - \psi_\mathrm{wf}, \qquad \text{where} \qquad \psi_\mathrm{wf} = -\frac{E_\mathrm{w}}{\mathrm{q}} \qquad (8)$$

The difference between the conduction band energy E_C and the metal workfunction energy gives the workfunction difference energy E_w which is the barrier height of the Schottky contact.

For Dirichlet boundary conditions one gets

$$\begin{aligned} f_{x_i} = \quad & x_\mathrm{C} - h(x_i) & = 0 \qquad (9) \\ f_{F_\mathrm{C}} = \quad & F_\mathrm{C} + \textstyle\sum_i f_{x_i}^{\mathrm{S}} & = 0 \qquad (10) \end{aligned}$$

Here, x_C is the boundary value of the quantity, which is a solution variable, whereas (10) is used as constitutive relation for the actual flow over the boundary F_C. For example, at Ohmic contacts simple Dirichlet boundary conditions apply. The contact potential ψ_s, the carrier contact concentrations n_s and p_s, and in the hydrodynamic simulation case, the contact carrier temperatures T_n and T_p are fixed. The metal quasi-Fermi level (which is specified by the contact voltage ψ_C) is equal to the semiconductor quasi-Fermi level. With the constant built-in potential ψ_bi (calculated after (Fischer 1994)), the contact potential at the semiconductor boundary reads

$$\psi_\mathrm{s} = \psi_\mathrm{C} + \psi_\mathrm{bi}. \qquad (11)$$

For Neumann boundaries the flux over the boundary is zero hence the equation assembled by the segment model is already complete.

Separate Contact Variables

Having a separate solution variable for the contact voltage avoids numerical problems with large arguments of the Bernoulli function B. Using a Scharfetter-Gummel discretization scheme (Scharfetter and Gummel 1969) the expression for the current between two grid points i and j reads

$$\begin{aligned} I_{i,j} = C_1(B(\Delta)n_j - B(-\Delta)n_i)\,x \qquad (12) \\ \text{with} \quad \Delta = C_2(\psi_j - \psi_i) + C_3 \qquad (13) \end{aligned}$$

with C_i being material parameters. Applying the contact voltage directly to the boundary grid point could cause large arguments of B and hence numerical problems. This is avoided by having a separate variable for the contact voltage. At the beginning of the iteration procedure the constitutive relation for ψ_C is violated and will only successively be adapted which guarantees numerical stability. The generalized boundary condition is the constitutive relation for the contact potential ψ_C and reads:

$$f_{\psi_\mathrm{C}} = \alpha\psi_\mathrm{C} + \beta I_\mathrm{C} + \gamma Q_\mathrm{C} - \delta = 0 \qquad (14)$$

where Q_C is the contact charge and $I_\mathrm{C} = I_{n_\mathrm{C}} + I_{p_\mathrm{C}} + \frac{\partial Q_\mathrm{C}}{\partial t}$ the contact current. It should be noted that all these quantities are solution variables which are directly available.

SOLVING OF THE NONLINEAR SYSTEM

MINIMOS-NT organizes the solving of the nonlinear, but discretized control functions $\mathbf{f} = \mathbf{0}$ using a damped Newton algorithm (k is the number of the iteration step) (Selberherr 1984):

$$\begin{aligned} \mathbf{J}^k \cdot \mathbf{x}^{k+1} &= \mathbf{f}(\mathbf{v}^k) \qquad (15) \\ \mathbf{v}^{k+1} &= \mathbf{v}^k + F_\mathrm{d}\mathbf{x}^{k+1} \qquad (16) \\ \mathbf{J} &= -\frac{\partial \mathbf{f}}{\partial \mathbf{v}} \qquad (17) \end{aligned}$$

where $\mathbf{J}$ is the Jacobian matrix, $\mathbf{f}(\mathbf{v})$ the residual and $\mathbf{x}$ the update or correction vector (solution vector of the linear system) that is then used to calculate the next solution vector $\mathbf{v}$ of the Newton approximation.

To avoid overshoot of the solution several damping schemes suggested by Deuflhard (Deuflhard 1974) or Bank and Rose (Bank and Rose 1981) are providing a damping factor F_d. For each Newton iteration step a linear equation system $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ has to be assembled and solved.

THE ASSEMBLY MODULE

MINIMOS-NT consists of two separate modules responsible for assembling and solving linear equation systems:

1. the assembly module which is directly accessed by the implemented physical models of the simulator, provides an effective application programming interface, various transformation algorithms and the preelimination system. In addition, sorting and scaling plug-ins can be called.

2. the solver module which is plugged into the assembly module, is responsible for solving the so-called inner linear equation system. The module currently used

provides a direct (Gaussian) method and two iterative solver schemes.

The key demands on the assembly module (class) can be summarized as follows:

1. Application Programming Interface providing methods for
   - adding contributions to the segment system
   - adding contributions to the boundary system
   - adding contributions to the transformation matrix
   - deleting equations
   - setting elimination flags
   - administration of priority information
2. Row transformation: linear combination of rows to extinguish large entries (see Section).
3. Variable transformation: reduce the coupling of the semiconductor equations (see Section).
4. Preelimination: eliminate problematic equations by Gaussian elimination to improve the condition of the inner system matrix (see Section)
5. Call of specific plug-ins (see Section) for
   - Scaling: Since a threshold value (tolerance) is used to decide whether to keep or skip an entry, the preconditioner used (Incomplete-LU factorization) requires a system matrix having entries of the same order of magnitude.
   - Sorting: Reduction of the bandwidth of a matrix to reduce the fill-in.
   - Solving: Calculate the solution vector of the linear equation system.

The input of the assembly module are the contributions of the various segment and boundary models implemented in the simulator. The assembly module compiles these values to a linear equation system, which is subsequently transformed in order to improve the condition of the system matrix.

For some simulations, for example derivation of the complex-valued admittance matrix, several linear equation systems differ only in the right-hand-side vector. Thus, the effort for assembling, compiling, preeliminating, sorting, scaling and factorizing of the system matrix actually has to be done only once - and this factored matrix could then be used for all RHS-vectors. For that reason the module is able to simultaneously assemble several RHS vectors.

A plug-in concept has been implemented for scaling, sorting and solving the inner linear equation system, making it possible to adapt or replace these modules easily. The sorting and scaling modules get the system matrix on input and return the sorting and scaling (diagonal) matrices which are then applied by the assembly module. The solver module gets the system matrix and all RHS vectors on input and returns the solution vectors of all inner linear equation systems on output.

After reverting all transformations and backsubstituting the preeliminated equations, the output of the assembly module are the complete solution vector (or vectors in case of multiple right-hand-side vectors). In addition, the right-hand-side vector(s) are returned which can be used for various norm calculations.

A schematic overview of the complete concept is given in Fig. 5. In the upper left corner the Newton iteration control function IterateOnce is represented, which is divided into two stages and uses an interface class to access the assembly module. Following the solid lines beginning at the interface, the four matrices $\mathbf{T}_\mathrm{b}$, $\mathbf{A}_\mathrm{s}$, $\mathbf{A}_\mathrm{b}$, and $\mathbf{T}_\mathrm{v}$ (see Section) are assembled by using a specific storage class. All diagonal elements are stored in one array, and the off-diagonals, the positions of which are not known in advance, in a balanced binary tree for each row, sorted by column. This allows flexible adding of all entries of the sparse matrices.

These structures are then converted to the *Modified Sparse Compressed Row* (MCSR) format (Saad 1990) and are compiled resulting in the complete linear system (auxFLG), which is preeliminated to get the inner and the outer linear equation system. The inner one (represented by C) is passed to the sorting and scaling plug-ins and finally solved by the solver module. After the solution has been calculated, scaling and sorting have to be reversed and the preeliminated equations are solved back.

The Newton adjustment levels (dashed lines) reuse already existing MCSR structures, which reduces the assembling effort: the balanced trees may be skipped completely, and during compiling and preelimination much simpler functions (bold boxes) can be used than in the conventional assembly mode (bold boxes with slash).

Assembly of the Complete Linear Equation System

The semiconductor device is divided into several segments that are geometrical regions employing a distinct set of models. The implementation of each model is completely independent from other models and each model is basically allowed to enter its contributions to the linear equation system. All boundary and interface issues are completely separated

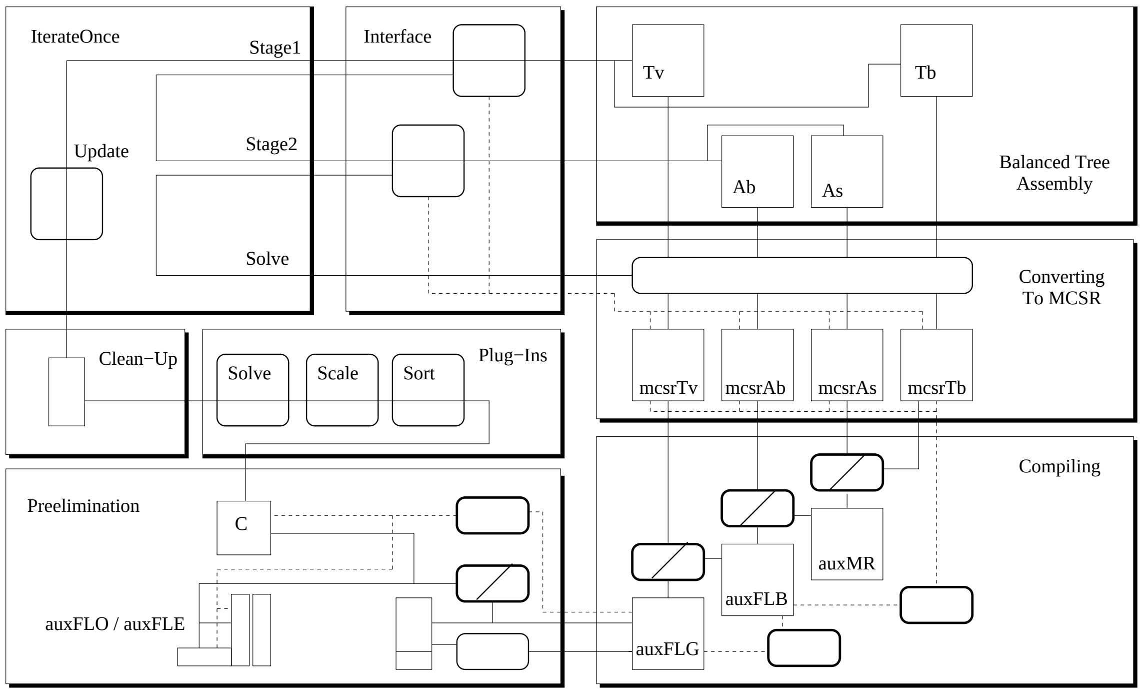

Figure 5: Schematic assembly overview

from the general segment models, which is represented by assembly structures for the boundary system which are independent from the segment ones.

Thus, the system matrix $\mathbf{A}$ (the Jacobian matrix in Newton approximation) will be assembled from two parts, namely the direct part $\mathbf{A}_\mathrm{b}$ (boundary models) and the transformed part $\mathbf{A}_\mathrm{s}$ (segment models). The latter is multiplied by the row transformation matrix $\mathbf{T}_\mathrm{b}$ from the left before contributing to the system matrix $\mathbf{A}$. The right hand side vector $\mathbf{b}$ is treated the same way:

$$\mathbf{A} = \mathbf{A}_\mathrm{b} + \mathbf{T}_\mathrm{b} \cdot \mathbf{A}_\mathrm{s} \tag{18}$$

$$\mathbf{b} = \mathbf{b}_\mathrm{b} + \mathbf{T}_\mathrm{b} \cdot \mathbf{b}_\mathrm{s} \tag{19}$$

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{20}$$

Although in principle every model is allowed to add entries to all components, the assembly module checks two prerequisites before actually entering the value: first, the quantity the value belongs to is marked to be solved (the user may request only a subset of all provided models) and secondly the priority of the model is high enough to modify the row transformation properties. As stated before, the row transformation is used to complete missing fluxes in boundary boxes. Since a grid point can be part of more than two segments, a ranking using a priority has been introduced. For example, contact models have usually the highest priority and thus their contributions are always used for completion. All three matrices $\mathbf{A}_\mathrm{b}$, $\mathbf{A}_\mathrm{s}$, and $\mathbf{T}_\mathrm{b}$ and the two vectors $\mathbf{b}_\mathrm{b}$ and $\mathbf{b}_\mathrm{s}$ may be assembled simultaneously, so no assembly sequence must be adhered to. In addition, a forth matrix $\mathbf{T}_\mathrm{v}$ is assembled which contains information for an additional variable transformation (see Section).

During the assembling process, all contributions are added to values stored in balanced binary trees. After the assembly is finished, these trees are converted to the sparse matrix format MCSR since all necessary mathematical operations are defined for these structures. The analogous, column oriented MCSC format is used to speed up column deleting required by the transformation matrix.

During the Newton iterations the structural configuration of these matrices is not modified very often (e.g., on enabling more derivatives), thus, the tree assembly may be skipped and the variables may be entered directly in the already existing MCSR structures. Hence, the effort for deleting, tree assembling, reallocating and converting can be saved which drastically speeds up the assembly process. However, if an entry in the structure is missing, the conventional assembly procedure can be easily restarted. The so-called Newton adjustment addresses not only the assembly matrices, but also the resulting structures of the compilation and preelimination process.

Row Transformation

The complete linear equation system is built from an original system (segment system), which is the main matrix $\mathbf{A}_\mathrm{s}$ and the main right hand side vector $\mathbf{b}_\mathrm{s}$, both of them representing cumulated fluxes and their derivatives to the system variables. Basically, the fluxes are calculated from segment models which are the models for the interior of discretized regions. The matrix is a linear superposition of very small matrices, one for each flux, with a few non-zero elements only. Consequently, the same superposition applies for the vector $\mathbf{b}_\mathrm{s}$. All fluxes are assigned to boxes, a box is in turn assigned to each variable.

As the control function for a box is defined by the user, for example being the sum of all fluxes leaving the box, the fluxes leaving the boxes are entered into the vector $\mathbf{b}_\mathrm{s}$ in the places appropriate for the variables that are assigned to the boxes. In context of the Newton method, matrix $\mathbf{A}_\mathrm{s}$ contains the negative derivatives of the values in $\mathbf{b}_\mathrm{s}$ to the system variables, so that the change $\mathrm{d}\mathbf{x}$ in the variables leads to a change $-\mathbf{A}_\mathrm{s} \cdot \mathrm{d}\mathbf{v} = \mathrm{d}\mathbf{b}_\mathrm{s}$ in the right hand side. Considering $\mathbf{b}_\mathrm{s}$ a function of the variable vector $\mathbf{v}$, one can write:

$$\mathbf{b}_\mathrm{s} = \mathbf{b}_\mathrm{s}(\mathbf{v}) \tag{21}$$

$$\mathbf{A}_\mathrm{s} = -\frac{\mathrm{d}\mathbf{b}_\mathrm{s}}{\mathrm{d}\mathbf{v}} \tag{22}$$

The boundary conditions will enforce some special physical conditions at the boundaries. The control functions of boxes along the boundary will usually be completed by the boundary conditions. For example, a Dirichlet boundary condition will use the dielectric flux cumulated in the boundary box to calculate the surface charge on the surface of the adjacent material. The equation used to calculate the value of the boundary variable, however, will not always make use of the fluxes accumulated in the main system.

The boundary conditions are therefore implemented by two elements: a boundary system ($\mathbf{A}_\mathrm{b}$ and $\mathbf{b}_\mathrm{b}$) and a transformation matrix $\mathbf{T}_\mathrm{b}$. The purpose of the matrix $\mathbf{T}_\mathrm{b}$ is the forwarding of the fluxes of the main system to their final destination or their resetting if they are not required. The system of $\mathbf{A}_\mathrm{b}$ and $\mathbf{b}_\mathrm{b}$ represents additional or substitutional parts of the final equation for the variables at the boundaries. Again, the entries in the matrix $\mathbf{A}_\mathrm{s}$ are the negative derivatives of the right hand side vector $\mathbf{b}_\mathrm{b}$ to the variable vector $\mathbf{v}$:

$$\mathbf{b}_\mathrm{b} = \mathbf{b}_\mathrm{b}(\mathbf{v}) \tag{23}$$

$$\mathbf{A}_\mathrm{b} = -\frac{\mathrm{d}\mathbf{b}_\mathrm{b}}{\mathrm{d}\mathbf{v}} \tag{24}$$

The Complete Linear System

The full system is assembled from the segment system and the boundary system in the following way:

$$\mathbf{b} = \mathbf{b}_\mathrm{b} + \mathbf{T}_\mathrm{b} \cdot \mathbf{b}_\mathrm{s} \tag{25}$$

$$\mathbf{A} = \mathbf{A}_\mathrm{b} + \mathbf{T}_\mathrm{b} \cdot \mathbf{A}_\mathrm{s} \tag{26}$$

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{27}$$

$$(\mathbf{A}_\mathrm{b} + \mathbf{T}_\mathrm{b} \cdot \mathbf{A}_\mathrm{s}) \cdot \mathbf{x} = \mathbf{b}_\mathrm{b} + \mathbf{T}_\mathrm{b} \cdot \mathbf{b}_\mathrm{s} \tag{28}$$

As stated above, vector $\mathbf{x}$ represents a linear change of the variables vector $\mathbf{v}$. With equation (29) and the new variables vector $\mathbf{v}_n$ from equation (30), the new value $\mathbf{b}_n$ of the vector function $\mathbf{b}(\mathbf{v})$ will be described by the linear approximation in equation (31).

$$\mathbf{x} = \mathrm{d}\mathbf{v} \tag{29}$$

$$\mathbf{v}_n = \mathbf{v} + \mathbf{x} \tag{30}$$

$$\mathbf{b}_n(\mathbf{v}_n) = \mathbf{b} + \frac{\mathrm{d}\mathbf{b}}{\mathrm{d}\mathbf{v}} \cdot \mathrm{d}\mathbf{v} = \mathbf{b} - \mathbf{A} \cdot \mathbf{x} = 0 \tag{31}$$

Variable Transformation

Especially in the case of mixed quantities in the solution vector, a variable transformation is sometimes helpful to improve the condition of the linear system. The representation chosen here allows to specify fairly arbitrary variable transformations to be applied to the system. Basically, a matrix $\mathbf{T}_\mathrm{v}$ is assembled and multiplied with the system matrix.

For example, to reduce the coupling of the semiconductor equations and thus improve the condition of the system matrix, a transformation of the stationary drift-diffusion model is suggested in (Ascher et al. 1986).

The transformation expressed by matrix $\mathbf{T}_\mathrm{v}$ is given by equation (32):

$$\begin{aligned}((\mathbf{A}_\mathrm{b} + \mathbf{T}_\mathrm{b} \cdot \mathbf{A}_\mathrm{s}) \cdot \mathbf{T}_\mathrm{v}) \cdot (\mathbf{T}_\mathrm{v}^{-1} \cdot \mathbf{x}) &= \\ &= (\mathbf{b}_\mathrm{b} + \mathbf{T}_\mathrm{b} \cdot \mathbf{b}_\mathrm{s})\end{aligned} \tag{32}$$

For compactness the following substitutions will be used hereinafter:

$$\tilde{\mathbf{A}} = ((\mathbf{A}_\mathrm{b} + \mathbf{T}_\mathrm{b} \cdot \mathbf{A}_\mathrm{s}) \cdot \mathbf{T}_\mathrm{v}) \tag{33}$$

$$\tilde{\mathbf{x}} = (\mathbf{T}_\mathrm{v}^{-1} \cdot \mathbf{x}) \tag{34}$$

$$\tilde{\mathbf{b}} = (\mathbf{b}_\mathrm{b} + \mathbf{T}_\mathrm{b} \cdot \mathbf{b}_\mathrm{s}) \tag{35}$$

Preelimination

The main matrix $\mathbf{A}_\mathrm{s}$ consists of fluxes that will (if the control functions are correctly assigned to the variables) satisfy the criterion of diagonal-dominance that is necessary to make the linear equation system solvable with an iterative solver. The transformations and additional terms imposed by the boundary conditions may heavily disrupt this feature both in structural and numerical aspects. Some of the boundary or interface conditions can make the full system matrix so ill-conditioned that this simply prevents iterative linear solvers from converging.

One solution to this problem which occurs only at the boundary variables that are affected in this way by the boundary conditions, is to apply Gaussian elimination to these variables/equations before the system is passed on to the linear solver. After the iterative solver has converged, the eliminated variables are calculated by backsubstitution into the eliminated equations.

Before they can be eliminated, the equations of this type are sorted to the back of the matrix, together with their assigned variables. This is done by applying a permutation matrix $\mathbf{P}$ to the linear equation system. The permutation matrix is calculated automatically on solving the system. The equation causing a possibly ill condition have to be marked for preelimination. The outer system is removed from the linear equation system and later solved by Gaussian elimination, the inner system is passed on to an iterative solver. See Fig. 6 (Fischer 1994) for an illustration of this concept.

The resulting system is given by (36):

$$(\mathbf{E}\cdot\mathbf{P}\cdot\tilde{\mathbf{A}}\cdot\mathbf{P}^{\mathrm{T}})\cdot(\mathbf{P}\cdot\tilde{\mathbf{x}}) = (\mathbf{E}\cdot\mathbf{P}\cdot\tilde{\mathbf{b}}) \qquad (36)$$

Here, $\mathbf{P}$ is the permutation matrix with its inverse equal to its transposed matrix $\mathbf{P}^{\mathrm{T}}$. $\mathbf{E}$ is a matrix of elimination coefficients obtained as the lower matrix $\mathbf{L}$ of a Gaussian elimination of the permuted system matrix. $\mathbf{E}$ contains non-zero off-diagonals in the outer parts only, the inner matrix up to the row/column index that narrows the section passed to the linear solver is a strict unity matrix.

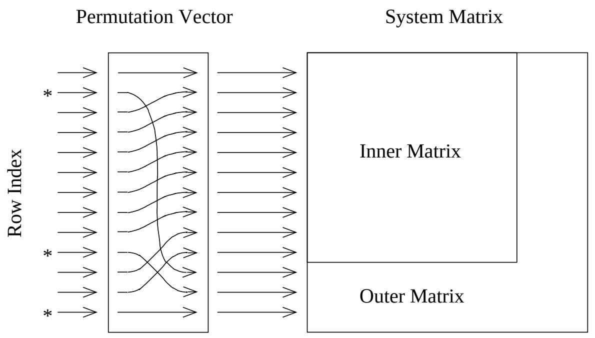

Figure 6: All equations marked for preelimination (*) are moved to the outer system matrix, the others remain in the inner one.

Call of Specific Plug-Ins

Matrices arising from the discretization of differential operators are sparse, because only neighbor points are considered. To reduce memory consumption, only the non-zero elements are stored (see MCSR format). During a factorization of $\mathbf{A}$ into an upper and lower triangular matrix $\mathbf{A} = \mathbf{L}\cdot\mathbf{U}$, additional matrix elements termed fill-in (Selberherr 1984) become non-zero and require additional memory. In order to minimize the fill-in, the system matrix is usually sorted. The standard module provided by default obtains the sorting matrix $\mathbf{R}_{\mathrm{s}}$ (similar to $\mathbf{P}$) by a Cuthill-McKee-based algorithm.

To provide the (ILU-) preconditioner with a normalized representation of the matrix, a scaling of all values has to be performed. The standard algorithm used by default works with a two-stage strategy (Fischer and Selberherr 1994): In the first stage, the matrix is scaled such that the diagonal elements are one. The second stage attempts to suppress the off-diagonals while keeping the diagonals at unity. The resulting scaling matrices $\mathbf{S}_{\mathrm{r}}$ and $\mathbf{S}_{\mathrm{c}}$ are diagonal matrices, and $\mathbf{R}_{\mathrm{s}}^{\mathrm{T}}$ equals $\mathbf{R}_{\mathrm{s}}^{-1}$. With $\mathbf{A}_i\cdot\mathbf{x}_i = \mathbf{b}_i$ as the inner system, the effect of sorting and scaling is given in (37):

$$\begin{aligned}(\mathbf{S}_{\mathrm{r}}\cdot(\mathbf{R}_{\mathrm{s}}^{\mathrm{T}}\cdot\mathbf{A}_i\cdot\mathbf{R}_{\mathrm{s}})\cdot\mathbf{S}_{\mathrm{c}})\cdot(\mathbf{S}_{\mathrm{c}}^{-1}\cdot(\mathbf{R}_{\mathrm{s}}^{\mathrm{T}}\cdot\mathbf{x}_i)) &= \\ = \mathbf{S}_{\mathrm{r}}\cdot(\mathbf{R}_{\mathrm{s}}^{\mathrm{T}}\cdot\mathbf{b}_i)\end{aligned} \qquad (37)$$

The assembly module is finally responsible for passing the inner linear equation system to the solver module. There are several approaches to obtain the solution vector: a Gaussian solver factorizes the matrix with a complete LU factorization, followed by a forward- and a backward substitution. Iterative solvers use successive approximations of this vector to obtain more accurate solutions to a linear equation system at each step (Barrett et al. 1994). In addition, the solver module provides a general interface to alternative solvers. After returning from the solver module, scaling and sorting are reverted and the preeliminated equations are backsubstituted.

CONCLUSION

We presented the concept and implementation of an advanced assembly module successfully applied in the device and circuit simulator MINIMOS-NT. The generally applicable module provides all conceptional and numerical features required for assembling and solving linear systems arising from discretized PDEs. The presented concepts result in superior stability of MINIMOS-NT without restricting model implementation and further development. The general approach for treating boundary conditions yields in combination with several preconditioning measures diagonal-dominant linear equation systems well prepared for advanced solver algorithms. As a result, boundary conditions for specific operating points can be directly applied without stepping to the desired value as is very common even in commercial simulators.

AUTHOR BIOGRAPHIES

STEPHAN WAGNER was born in Vienna, Austria, in 1976. He studied electrical engineering at the Technical University Vienna, where he received the master degree of ”Diplomingenieur” in 2001. As a member of the MINIMOS-NT development group he joined the Institute for Microelectronics in November 2001, where he is currently working on his doctoral degree. His scientific interests include device and circuit simulation, numerical aspects and software technology.

TIBOR GRASSER was born in Vienna, Austria, in 1970. He studied communications engineering at the Technische Universität Wien where he received the ‘Diplomingenieur’ and the doctoral degree in technical sciences in 1995 and 1999, respectively. He joined the Institut für Mikroelektronik in April 1996 where he is currently employed as an Assistant. In 2002 he received the venia docendi on Microelectronics. Since 1997 he has been heading the MINIMOS-NT development group, working on the successor of the highly successful MINIMOS program.

CLAUS FISCHER Claus Fischer was born in Vienna, Austria, in 1967. He received the doctoral degree in technical sciences from the ’Technische Universität Wien’ in 1994. He started the Minimos NT project in order to extend the proven concepts and numerical methods of Minimos to complex geometrical and topological structures and introduced a generalized physical interface treatment. After Ph.D., he worked in the semiconductor industry for three years. He now runs his own software engineering company in Austria. His interests cover a range of software engineering topics for industrial use, from technical to business applications.

SIEGFRIED SELBERHERR was born in Klosterneuburg, Austria, in 1955. He received the degree of ‘Diplomingenieur’ in electrical engineering and the doctoral degree in technical sciences from the ‘Technische Universität Wien’ in 1978 and 1981, respectively. Dr. Selberherr has been holding the ‘venia docendi’ on ‘Computer-Aided Design’ since 1984. Since 1988 he has been the head of the ‘Institut für Mikroelektronik’ and since 1999 he has been dean of the ‘Fakultät für Elektrotechnik’. His current topics are modeling and simulation of problems for microelectronics engineering.

REFERENCES

Ascher, U.; P.A. Markowich; C. Schmeiser; H. Steinrück; and R. Weiss. 1986. Conditioning of the Steady State Semiconductor Device Problem. Technical Report 86-18, University of British Columbia.

Bank, R.E. and D.J. Rose. 1981. ”Global Approximate Newton Methods”. *Numer.Math.*, 37:279–295.

Barrett, R.; M. Berry T.F. Chan; J. Demmel; J. Donato; J. Dongarra; V. Eijkhout; R. Pozo; C. Romine; and H. Van der Vorst. 1994. *Templates for the Solution of Linear Systems: Building Blocks of Iterative Methods*. SIAM, Philadelphia, PA.

Bløtekjær, K. 1970. ”Transport Equations for Electrons in Two-Valley Semiconductors”. *IEEE Trans.Electron Devices*, ED-17(1):38–47.

Deuflhard, P. 1974. ”A Modified Newton Method for the Solution of Ill-Conditioned Systems of Nonlinear Equations with Application to Multiple Shooting”. *Numer.Math.*, 22:289–315.

Fischer, C. 1994. *Bauelementsimulation in einer computergestützten Entwurfsumgebung*. Dissertation, Technische Universität Wien. http://www.iue.tuwien.ac.at.

Fischer, C. and S. Selberherr. 1994. ”Optimum Scaling of Non-Symmetric Jacobian Matrices for Threshold Pivoting Preconditioners”. In *Intl. Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits NUPAD V*, pages 123–126, Honolulu.

Grasser, T.; H. Kosina; M. Gritsch; and S. Selberherr, 2001. ”Using Six Moments of Boltzmann’s Transport Equation for Device Simulation”. *J.Appl.Phys.*, 90(5):2389–2396.

Grasser, T.; T.-w. Tang; H. Kosina; and S. Selberherr. 2003. ”A Review of Hydrodynamic and Energy-Transport Models for Semiconductor Device Simulation”. *Proc.IEEE*, 91(2):251–274.

Grasser, T. and S. Selberherr. 2001. ”Fully-Coupled Electro-Thermal Mixed-Mode Device Simulation of SiGe HBT Circuits”. *IEEE Trans.Electron Devices*, 48(7):1421–1427.

Saad, Y. 1990. *A Basic Tool Kit for Sparse Matrix Computations*. Technical Report, RIACS, NASA Ames Research Center, Moffett Field, CA 94035.

Scharfetter, D.L. and H.K. Gummel. 1969. ”Large-Signal Analysis of a Silicon Read Diode Oscillator”. *IEEE Trans.Electron Devices*, ED-16(1):64–77.

Schroeder, D. 1994. *Modelling of Interface Carrier Transport for Device Simulation*. Springer.

Selberherr, S. 1984. *Analysis and Simulation of Semiconductor Devices*. Springer, Wien–New York.

Simlinger, T. 1996. *Simulation von Heterostruktur-Feldeffekttransistoren*. Dissertation, Technische Universität Wien. http://www.iue.tuwien.ac.at.

Stratton, R. 1962. ”Diffusion of Hot and Cold Electrons in Semiconductor Barriers”. *Physical Review*, 126(6):2002–2014.

VanRoosbroeck, W.V. 1950. ”Theory of Flow of Electrons and Holes in Germanium and Other Semiconductors”. *Bell Syst.Techn.J.*, 29:560–607.

Wachutka, G.K. 1990. ”Rigorous Thermodynamic Treatment of Heat Generation and Conduction in Semiconductor Device Modeling”. *IEEE Trans.Computer-Aided Design*, 9(11):1141–1149.