Concepts and Implementation of an Advanced Equation Assembly Module

Stephan Wagner[⋆], Tibor Grasser[⋆], Claus Fischer[†], and Siegfried Selberherr[∘]

[⋆]Christian Doppler Laboratory for TCAD in Microelectronics at the Institute for Microelectronics
[∘] Institute for Microelectronics, TU Vienna, Gußhausstraße 27–29, A-1040 Wien, Austria
Phone: +43-1-58801/36037, Fax: +43-1-58801/36099, Email: Wagner@iue.tuwien.ac.at
[†]Firma Dr. Claus Fischer
Gustav Fuhrichweg 24/1, A-2201 Gerasdorf bei Wien Austria

ABSTRACT

We present concepts and an implementation of a generally applicable module which allows to efficiently assemble equations. Such a module is required for solving a system of nonlinear partial differential equations discretized on a grid. Since the nonlinear problem is then usually solved by a damped Newton algorithm the solution of a linear equation system has to be obtained at each step. Our assembly approach for these systems has been originally developed for the simulation of semiconductor devices based on the Finite Boxes discretization scheme. The implemented module which is not restricted to this field of application, is currently employed in two numerical simulators developed at the Institute for Microelectronics. In addition to the equation assembly, it provides several functionalities relevant for the simulation process and numerical stability, namely the representation of boundary and interface conditions, physically motivated variable transformation, and numerical conditioning of the system matrices.

Keywords: Linear Equation Systems, Numerical Simulator Development, Linear Solvers, Sorting, Scaling

MOTIVATION

Handling equations is a fundamental but involved task required in many numerical simulators. Basically, all models calculate and store their contributions to linear equation systems which are passed to a solver module in order to get the solution vector. In the following several motivations are given, why the separation of respective simulator functions and the modules responsible for linear systems functionality is highly desirable.

Due to specific properties of the assembled linear equation systems (e.g. equations that are not diagonal-dominant), they cannot be generally solved in reasonable time. Thus, they require additional conditioning measures which can be performed independently and after the assembly process. For example, the assembly module can provide facilities to transform these systems to improve the subsequent solving process in terms of convergence, time, and memory consumption. Furthermore, the interface functions of different solver modules lack a standard definition. The resulting problems, for example the matrix storage format, can be addressed by an independent module with completely encapsulated data structures. For that reason the coupling of a new solver module requires an adapted or new assembly-solver interface, but the simulator-assembly interface remains unaffected. In an abstract way the simulator can be seen as a client process using the services provided by a numerical server. On the other side, the respective model (client) developer can use an abstract application programming interface (API of the server) and is exempted of special considerations regarding the linear equation systems.

In this work we start with a short introduction into the target field of application. Therefore, we define an analytical problem and sketch its discretization in order to present the functionality of the module in the context of a typical example. The module itself is available as a shared library for various platforms.

INTRODUCTION

The Finite Boxes discretization method [1] is employed in various kinds of numerical tools and simulators for the fast and accurate solution of nonlinear partial differential equation (PDE) systems. The resulting 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 the core modules responsible for handling the linear equation systems. We present an advanced equation assembly module which is currently employed in the device and circuit simulator MINIMOS-NT [2] and in the Finite Elements Diffusion and Oxidation Simulator FEDOS [3]. The simulation of semiconductor devices with MINIMOS-NT is used to illustrate the various concepts hereinafter.

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, several different types of transport equations can be solved. Among these are the hydrodynamic equations which capture hot-carrier transport [4, 5] the lattice heat flow equation to cover thermal effects like self-heating, and the circuit equations to connect single devices to a circuit [6], 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 have been identified and generalized.

A crucial aspect is also the requirement of assembling and solving complex-valued linear equation systems which are needed during small-signal and noise simulations directly in the frequency domain.

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 the so-called method of moments this equation can be transformed to an infinite series of equations. Keeping only the zero and first order moment equations (with proper closure assumptions) yields the basic semiconductor equations (drift-diffusion model).

These equations as first given by VanRoosbroeck [7] are the Poisson equation (1) (derived from Maxwell's equations), the continuity equations for electrons (2) and holes (3) including a drift and diffusion term:

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

$$\mathrm{div}\left(D_n \cdot \mathrm{grad}\, n - \mu_n \cdot n \cdot \mathrm{grad}\, \psi\right) = R + \frac{\partial n}{\partial t} \tag{2}$$

$$\mathrm{div}\left(D_p \cdot \mathrm{grad}\, p + \mu_p \cdot p \cdot \mathrm{grad}\, \psi\right) = R + \frac{\partial p}{\partial t} \tag{3}$$

The unknown quantities of this equation system are the electrostatic potential ψ, and the electron and hole concentrations n and p, respectively. ε is the dielectric permittivity of the semiconductor, ρ denotes the space charge density, D_n and D_p are the diffusion coefficients, μ_n and μ_p stand for the carrier mobilities, and R describes the net recombination rate. These variables depend on the unknown quantities ψ, n, and p and have to be modeled properly [8]. The equation system is rendered by these models in a nonlinear form.

The heat flow equation (4) may be added to account for thermal effects in a device:

$$\mathrm{div}(\kappa_\mathrm{L} \cdot \mathrm{grad} T_\mathrm{L}) = \rho_\mathrm{L} \cdot c_\mathrm{L} \tag{4}$$

This equation requires proper modeling of the thermal conductivity κ_L, the mass density ρ_L, and the heat capacity c_L. The parameters of equations (1) to (3) depend also on the lattice temperature T_L and have again to be modeled properly.

Considering two additional moments gives the hydrodynamic model [5], where the carrier temperatures are allowed to be different from the lattice temperature. Since the current densities depend then on the respective carrier temperature, two additional unknowns, the electron temperature T_n and the hole temperature T_p, are added.

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. 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 partioned into a finite number of sub-domains D_i, in which the solution can be approximated with a desired accuracy.

2. The PDE system is approximated in each of the sub-domains 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 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. A solution of the unknowns of the nonlinear algebraic system must be computed. 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 resolution of the partitioning into sub-domains as well as on the suitability of the approximating functions.

THE DISCRETIZED PROBLEM

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.

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 time-dependent 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 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 [1] the equations are integrated over a control volume (subdomain, usually obtained by a Voronoi tesselation). Grid points on the boundary ∂D are defined as having neighbor grid points in other segments. Thus, the segment models are not able to assemble the complete control functions, 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 ∂D, the Neumann condition which specifies the normal derivative, and the linear combination of these conditions giving an intermediate type:

$$\mathbf{n} \cdot \mathrm{grad} x + \sigma x = \delta \tag{5}$$

Generally, the form of these conditions depends on the respective boundary models. 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 segment 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.

It was shown [9] that more complex models with exponential interdependence between the solution variables such as thermionic field emission interface conditions can also be implemented.

The method which has been developed allows to implement the segment models which describe 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. For a detailed discussion of the interface and boundary conditions of this approach see [10].

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 from the general segment models. Hence, also completely independent assembly structures for the boundary and segment system are used.

Thus, the system matrix **A** (the Jacobian matrix in the 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 **A**. The right hand side vector **b** is treated the same way:

$$\mathbf{A} = \mathbf{A}_\mathrm{b} + \mathbf{T}_\mathrm{b} \cdot \mathbf{A}_\mathrm{s} \tag{6}$$

$$\mathbf{b} = \mathbf{b}_\mathrm{b} + \mathbf{T}_\mathrm{b} \cdot \mathbf{b}_\mathrm{s} \tag{7}$$

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{8}$$

Although in principle every model is allowed to add entries to all components, the assembly module checks two pre-requisites 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 fourth matrix $\mathbf{T}_\mathrm{v}$ is assembled which contains information for an additional variable transformation.

THE ASSEMBLY MODULE

As stated before, two separate modules are provided 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 pre-elimination system. In addition, sorting and scaling plug-ins can be called.

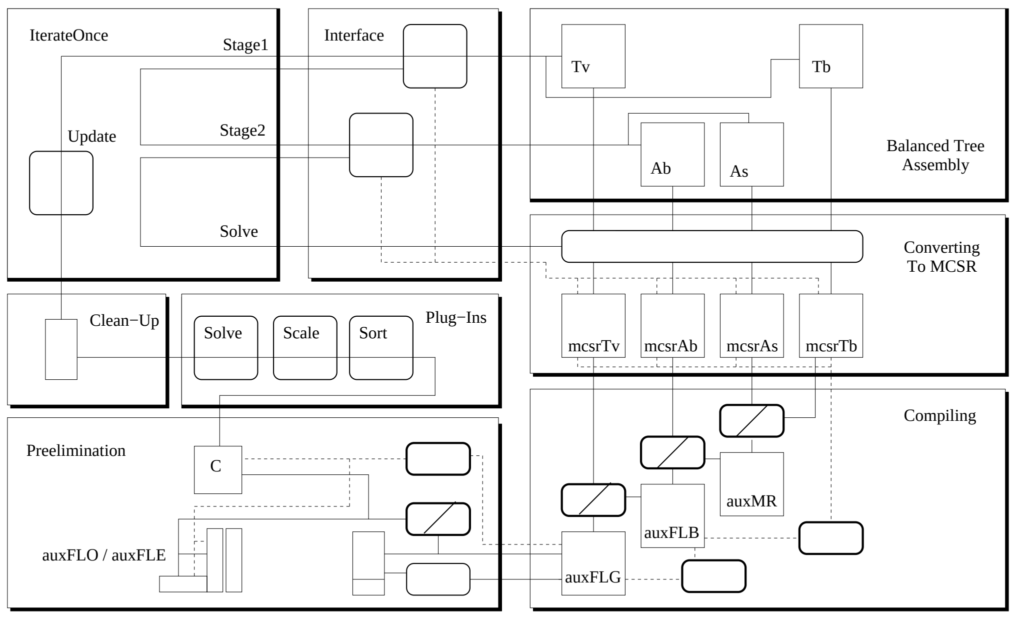

Figure 1: Schematic overview.

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

The Application Programming Interface provides methods for

- adding values to the segment system
- adding values to the boundary system
- adding values to the transformation matrix
- deleting equations
- setting elimination flags
- administration of priority information

In addition, a rigorous input/output system is provided in order to support the developer with detailed information on the matrices and the solving process. All matrices or the equation systems can be written to files (frequently needed for model debugging purposes) or read from files to test alternative solver configurations. An additional functionality, employed for the matrix structure figures in this paper, allows to create input matrices for the Open Visualization Data Explorer [11].

A schematic overview of the complete concept is given in Fig. 1. In the upper left corner the Newton iteration control function *IterateOnce* of the simulator is represented. Following the solid lines beginning at the interface, four matrices (segment, boundary, and two transformation matrices) 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. Note, that the right-hand-side vectors are analogously assembled, compiled and transformed, which is not separately shown in Fig. 1.

These structures are then converted to the *Modified Sparse Compressed Row* (MCSR) format [12] and are compiled resulting in the complete linear system (auxFLG in Fig. 1). The row transformation performs a linear combination of rows to extinguish large entries.

The variable transformation is used to reduce the coupling of the semiconductor equations. 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. See Fig. 2 for a completely compiled matrix arising from the discretization of a two-dimensional MOS transistor structure.

The preelimination is required to eliminate problematic equations (e.g. those with large off-diagonal entries or structurally different equations of the boundary conditions) by Gaussian elimination in order to improve the condition of the inner system matrix. 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. See Fig. 3 how the preelimination affects the system matrix.

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 mode (bold boxes with slash). A related feature incorporates the sorting of the inner system matrix into the reordering phase of the preelimination process. All these features together achieve a performance gain up to 11%.

But there are various additional speed-up features: in some simulations, e.g. the calculation of the complex-valued admittance matrix, the system matrix remains constant while several sets of different boundary conditions only affect the right-hand-side vectors. In such a case the effort for assembling, compiling, preeliminating, sorting, scaling and factorizing of the system matrix actually has to be spent only once - and this factored matrix could then be used for all right-hand-side vectors. The solution vectors are multiply calculated during the relatively fast back-substitution step. In addition, the assembly of the real-valued part of the system matrix can be skipped during a frequency stepping which affects the imaginary contributions only.

After the preelimination, specific plug-ins are called for sorting and scaling. 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 (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 known as fill-in [1] become non-zero. The profile

$$p(\mathbf{A}) = \sum_{i=1}^{n} m_i \quad \text{with} \quad m_i = i - \min_{a_{i,j} \neq 0}(j) \tag{9}$$

is a measure for this fill-in and the bandwidth of the matrix is $\max_i(m_i)$. Since storing of $p(\mathbf{A})$ requires additional memory, the equations are specially ordered to reduce the bandwidth and thus the profile. The standard module provided by default is based on a Cuthill-McKee algorithm [13]. It is applied in such a way that row and columns are correspondingly swapped to keep the diagonal dominance. Fig. 4 shows the effect of the reordering plug-in.

Since a threshold value (tolerance) is used to decide whether to keep or skip an entry, the ILU preconditioner requires a system matrix having entries of the same order of magnitude. To provide 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 [14]: 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.

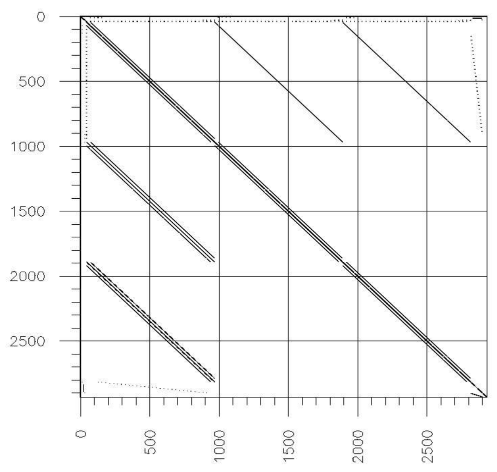

Figure 2: The completely compiled system matrix of a discretized two-dimensional MOS transistor structure assembled by MINIMOS-NT.

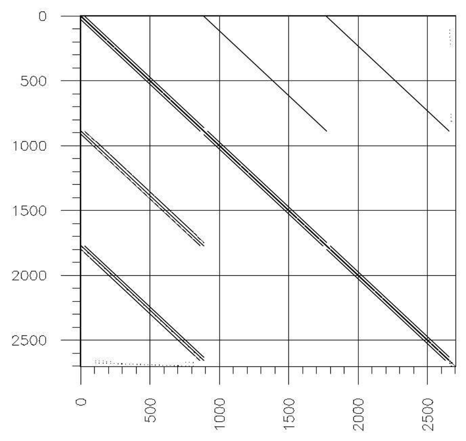

Figure 3: The inner system matrix does not contain the problematic equations any more, e.g. the equations in rows 1-44 of the complete system matrix in Fig. 2.

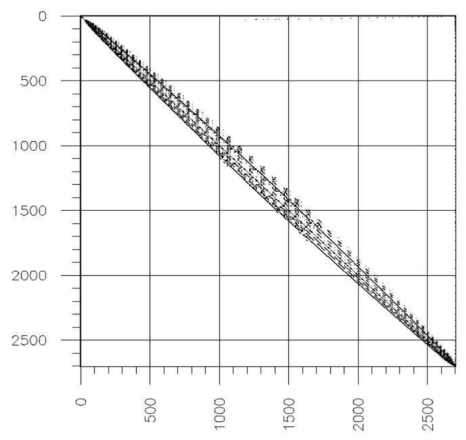

Figure 4: In comparison to the preeliminated structure, the reordering algorithm significantly reduced the bandwidth in order to reduce the factorization fill-in.

INTERFACE TO THE SOLVER MODULE

The assembly module has a standardized interface to solver modules. Thus, the scaled inner equation system is directly passed to the solver module in order to get the solution vector for that system.

This module offers two iterative solvers, BICG-STAB and GMRES(m), in combination with a preconditioner based on Incomplete-LU factorization. In addition, a direct Gaussian LU factorization is provided, but also external solver modules can be coupled by the interface. At the moment, two external modules can be employed: First, the Parallel Sparse Direct Linear Solver PARDISO [15], which provides a multi-threaded direct solver as well as a LU-CGS iterative solver implementation. Second, the Algebraic Multigrid Methods for Systems (SAMG) [16], which provides multi-level algorithms. Both external packages are written in Fortran. Their only computational overhead is the required matrix storage format conversion. As this interface is part of the assembly module, the simulator remains completely unaffected, but has the opportunity to switch simply between the offered modules.

The solver module in charge is expected to return the solution vector of the inner equation system, which is further processed by the assembly module. After reverting all transformations and back-substituting the preeliminated equations, the output of the assembly module is the complete solution vector. In addition, the right-hand-side vector is returned which can be used for various norm calculations.

CONCLUSION

We presented the concept and implementation of an advanced assembly approach successfully applied in the simulators MINIMOS-NT and FEDOS. All conceptional and numerical features required for assembling and solving linear systems arising from semiconductor device, circuit, and process simulation are provided. We developed a formulation which allows to independently treat segments, boundaries, and interface models. As a consequence, all fluxes over boundaries are available as solution variables, which simplifies the formulation of boundary conditions and circuit equations.

A rigorous application programming interface is provided. The assembly and solver module is controlled with a a large set of parameter, which are incorporated in a single parameter class to centralize the input and output information. In addition, the simulator developer is supported by error and input/output systems.

The presented concepts result in superior stability of the simulators 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.

REFERENCES

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

[2] Institut für Mikroelektronik, Technische Universität Wien, Austria, “Minimos-NT 2.0 User’s Guide.” http://www.iue.tuwien.ac.at/software/minimos-nt, 2002.

[3] Institut für Mikroelektronik, Technische Universität Wien, Austria, “Fedos User’s Guide.” http://www.iue.tuwien.ac.at, 2003.

[4] R. Stratton, “Diffusion of Hot and Cold Electrons in Semiconductor Barriers,” *Physical Review*, vol. 126, no. 6, pp. 2002–2014, 1962.

[5] T. Grasser, T. Tang, H. Kosina, and S. Selberherr, “A Review of Hydrodynamic and Energy-Transport Models for Semiconductor Device Simulation,” *Proc.IEEE*, vol. 91, no. 2, pp. 251–274, 2003.

[6] T. Grasser and S. Selberherr, “Fully-Coupled Electro-Thermal Mixed-Mode Device Simulation of SiGe HBT Circuits,” *IEEE Trans.Electron Devices*, vol. 48, no. 7, pp. 1421–1427, 2001.

[7] W. VanRoosbroeck, “Theory of Flow of Electrons and Holes in Germanium and Other Semiconductors,” *Bell Syst.Techn.J.*, vol. 29, pp. 560–607, 1950.

[8] C. Snowden, *Semiconductor Device Modelling*. Springer, 1989.

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

[10] S. Wagner, T. Grasser, C. Fischer, and S. Selberherr, “A Generally Applicable Approach for Advanced Equation Assembling,” in *Proc. International Conference on Software Engineering and Applications SEA*, (Marina del Rey, CA), pp. 494–499, 2003.

[11] The Open Source Software Project Based on IBM’s Visualization Data Explorer, “Open Visualization Data Explorer.” http://www.opendx.org, 2003.

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

[13] E. Cuthill and J. McKee, “Reducing the Bandwidth of Sparse Symmetric Matrices,” in *ACM Conf.*, pp. 157–172, 1969.

[14] C. Fischer and S. Selberherr, “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*, (Honolulu), pp. 123–126, 1994.

[15] O. Schenk and K. Gärtner, “Solving Unsymmetric Sparse Systems of Linear Equations with PARDISO,” *Future Generation Computer Systems*, 2003. Accepted, in press.

[16] T. Clees and K. Stüben, “Algebraic Multigrid for Industrial Semiconductor Device Simulation,” in *Challenges in Scientific Computing - CISC 2002. Proceedings of the Conference ”Challenges in Scientific Computing”, Berlin, October 2-5, 2002* (E. Bänsch, ed.), vol. 35 of *Lecture Notes in Computational Science and Engineering*, pp. 110–130, Springer, Berlin, 2003.