Process and Device Simulation with a Generic Scientific Simulation Environment

Michael Spevak, René Heinzl, Philipp Schwaha, Tibor Grasser

Abstract— **We present a high performance environment for scientific simulation applications (GSSE). This environment is based on the three orthogonal concepts of topology, geometry, and quantities. Lambda calculus is used in order to assemble various partial differential equations for TCAD and other physical equations. We present examples from device and process simulation which show the applicability of our environment. Despite the high abstraction level we can archieve high performance.**

I. Introduction

A general environment for TCAD equations has to provide methods for the solution of different physical phenomena such as carrier transport, diffusion, electromagnetic wave propagation, heat transfer, mechanical deformation, fluid flow, and quantum effects. Due to the wide range of applications it is not trivial to develop an environment which is capable of handling all equations within a homogenous environment. In the field of TCAD coupled partial differential equations and multi quantity equations are often employed. For the solution of these equations we use discretization schemes such as the finite element method, the finite volume method, the finite difference method or the boundary element method.

Each of these schemes has its merits and shortcomings and is therefore more or less suited for different classes of equations. All of these methods have in common that they require a proper tessellation and adaption of the simulation domain [1], [2]. Due to the diversity of the mathematical structures themselves, combined with efficiency considerations, in particular in three dimensions, the development of simulation software is quite difficult. Supporting libraries for numerical simulations exist, but no complete environment for scientific computing to tackle the following issues:

- Support for different geometries and topologies
- Complete and tested discretization schemes
- Support for mathematical modeling
- High performance calculation
- (Real-time) visualization

Hence, the simulation tools are typically written by experts specialized in other fields. In the extreme case all areas of simulator development, like software design, programming, testing, and evaluation are done by one person. In the last decade our institute has developed different simulators and libraries, like SAP [3], Wafer-State-Server [4], and Minimos-NT [5]. However, none of them has shown to be perfect for the rapid progress in scientific software development. Even the reuse of simple code parts is difficult, due to the non-generic library approach.

Therefore a scientific environment with high flexibility and adaptivity of meshes combined with great flexibility in numerical treatment and discretization schemes in all dimensions is called for. It should be possible to use a common code base which is easily adaptable to special requirements but does not require specialized features for different discretization schemes such as element matrices like many other specialized finite element simulation environments.

On that account, we have extracted the main concepts from all of our simulation tools and developed the generic and lightweight environment GSSE which suits scientific requirements. On the one side, generic library means that each part of GSSE can be used separately. The complete GSSE is based on header files only and therefore the required mechanisms can be included without incurring additional dependencies. On the other hand, generic means that all data types are parameterized and can be exchanged easily, for instance the numerical data type for quantity storage.

GSSE is designed for rapid development of simulation software. One of the most important facts is that errors can be easily found and even prevented. As errors are often not obvious to detect it may already take a lot of experience to decide if the result from a simulation is erroneous or not. If the result is not correct, it might be reduced to a programming bug, a logical error in the program flow, or a badly chosen parameter. Due to this reason it is necessary to locate an error quickly. Each of the data structures which we provide can itself be tested before compound data structures are tested. Thus the development effort for the final code can be reduced enormously.

In the following we will present some examples of our simulations from the field of semiconductor device and process simulation. First we solve a simple device simulation example. Then we benchmark a finite element example with the electromagnetic simulation tool SAP.

M. Spevak is with the Institute for Microelectronics, Technical University Vienna, E-mail: spevak@iue.tuwien.ac.at

R. Heinzl, P. Schwaha, and T.Grasser are with the Christian Doppler Laboratory for TCAD in Microelectronics at the Institute for Microelectronics, Technical University Vienna, E-mail: {heinzl|schwaha|grasser}@iue.tuwien.ac.at

Both examples will show that our environment can perform fast calculations.

II. The Generic Scientific Simulation Environment

Due to the large variety of available models and differential equations, discretization schemes, and simulation domains we had to extract the base concepts of a simulation environment. The main aspect of software design is orthogonality as well as modularity. Each component should be usable without any dependences to another. If a higher structure combines two base structures (e.g. quantities on a topology) it does not depend on the lower structures but it only relies on the concept.

Topology. Within a scientific simulation environment it is crucial to have neighborhood information of vertices, edges, faces, and cells available within a constant time. For this reason we have implemented a data structure which covers only topological information of vertex cell incidence and cell vertex incidence. Even though storing one of these incidence functions is redundant it is necessary to guarantee constant time for traversal. Based on this incidence information we generate inter-dimensional objects such as edges and faces. The incidence information of edges and faces does not need to be stored explicitly because it can be derived from the base traversal functions and the archetype information.

The archetype concept implies that each cell of the tessellation has the same topological shape or very few different shapes. Therefore we need not store all edge and face information but we can derive it from an archetype. For this reason the unstructured topology is highly flexible. We can use archetypes of different dimensions and shapes.

Geometry. Even though we have a convenient method for describing the topology of a simulation domain this does not imply that we store the coordinate information on the vertices and cells. Topology tells us for instance if two vertices are connected by an edge. It however does not provide any information about the real geometry of the curve. This is the task of the geometry concept. The basic geometry concept is the point list. The point list contains the geometrical point coordinates associated to the topological vertices. From this information we can derive the geometry of all edges, faces and cells. We can perform orientation tests, volume measurements, and the calculation of the voronoi information.

Quantity. Quantities are numerical attributes which can be stored on all topological elements using their handles. Using an associative storage concept we can store values with respect to an associative key of the topology and a quantity descriptor key (which might be a string). The value types of the quantities can be chosen differently. The simplest case is a scalar floating point value. We also provide vector and tensor quantities as well as string quantities. The quantity data type can be parameterized on the data type so that it is possible to use any type as data type for the quantity.

Based on these core concepts, a mathematical function layer was implemented to provide easy development of all different modeling issues on the one side and a high performance computing on the other side. This layer includes all necessary functions for a convenient numerical analysis as well as accumulation functions which will be discussed later on in the example section. We even have the possibility to work with linearized functions which provide direct access to the system matrix for line-wise entry as well as finite element stencils.

As we have parameterized data types for numerical calculations it is possible to introduce abstract matrix data types with lazy evaluation concepts which can reduce the execution time as well as the number of temporary objects. For this reason we use expression templates [6], [7] as well as lambda expressions [8].

A solver interface is integrated in this environment for the use of all different kinds of state-of-the-art solvers called TRILINOS [9]. For the important visualization purpose within scientific computing, IBM's data explorer [10] is integrated with a few modifications to make a real-time visualization possible.

III. Device Simulation

The field of device simulation requires the use of many different numerical techniques. Macroscopic transport [11] models are among the most widely employed calculations. These models can be derived by applying the moment method on the Boltzmann equation. Together with the Poisson equation they form a system of partial differential equations which are capable of describing the behavior of semiconductor devices. We use the simplest macroscopic model, the drift diffusion transport equations. In the following we show how the equations can be discretized in our environment by the means of the finite volume method. The discretization formula for the Poisson equation yields

$$\sum_{\text{edge vertex}} (\Psi_j - \Psi_i) \frac{A}{d} = V (n - p + N_\text{A} - N_\text{D}) \, \text{q} \qquad (1)$$

Based on this discretization formula we obtain the final discretization routine (1) we can write the following statement in the GSSE.

```
eqn = (sum<vertex_edge>
[diff<edge_vertex>(0.0)[potential] * A / d]
  - q * volume * (n - p + N_A - N_D))(v);
```

The same assembly routine can be applied to the current relations using the Scharfetter-Gummel discretiza-

tion [12] (generation and recombination rates are omitted here).

$$J_w = \frac{1}{\Lambda}\left(n_j \mathcal{B}\left(\Lambda\right) - n_i \mathcal{B}\left(-\Lambda\right)\right) \ , \qquad (2)$$

where $\mathcal{B}$ is the Bernoulli function. Using the method of finite volumes we have discretize the current integral on the boundary of the control volumes. If we consider the stationary case as well as zero recombination we obtain the following code snippet for our discretization scheme.

```
eqn = sum<vertex_edge>(0.0)
 [area / distance * diff<edge_vertex>()
  [n * bern(d_psi/u_th)]
  [n * bern(-d_psi/u_th)]]
(v);
```

Both discretization terms need quantities which are located on topological elements (Fig. 1). The potential and charge terms as well as the box volume are stored on the vertices. The distance (d) and area (a) as well as the potential difference d_ψ are stored on the edges. The sum as well as the difference operations change the locality. The expression within the brackets gives the kind of traversal; `vertex_edge` iterates over all edges which cover a vertex, whereas `edge_vertex` iterates over all vertices which are covered by an edge. The formula in the square brackets are evaluated on all elements of the traversal and accumulated. `bern` denotes the Bernoulli function $\mathcal{B}$ which is used in the Scharfetter-Gummel current relation. From this specification the Jacobian

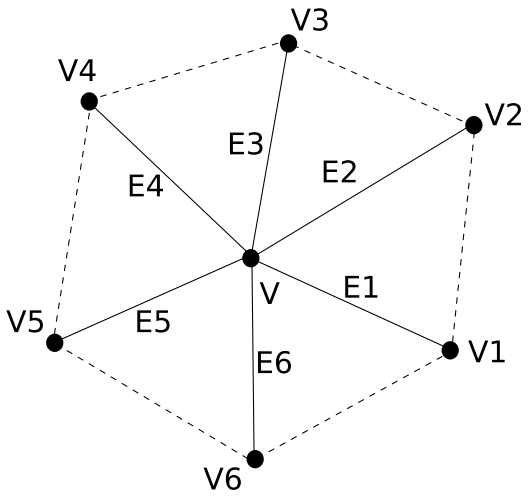

Fig. 1. The local patch of a vertex. We outlined the vertices as well as the edges.

matrix and the right hand side are assembled automatically. An automatic derivation of the linearized function, which is very error prone if done by hand, can be achieved by using an associative data structure. This data structure contains matrix entries as well as a right hand side value. For structures which are called linearized equations, we have implemented basic operations such as addition and multiplication as well as other numerical functions such as the already demonstrated Bernoulli function. This method can be used in order to assemble linear as well as nonlinear equations which include discrete couplings between single topological elements. The complete application does not need more than 100 lines of code, the core is only about 25 of these lines.

IV. Process Simulation

In general, three-dimensional process simulation steps need special surface treatment and must provide the ability to handle surface elements of arbitrary complexity containing degenerated or even faulty elements.

For the solution of problems in process simulation as well as interconnect simulation finite element methods are commonly used. In the following example we apply the finite element method to the Laplace equation in order to calculate capacitances as well as resistances of interconnect structures. In the following we will calculate two simple structures in order to evaluate the correctness of our simulation and to show the performance of the environment. The example is a single interconnect line. Even though it is very simple it shows the applicability of the GSSE as well as the performance.

V. Runtime Efficiency

To compare the runtime efficiency of the generic scientific environment we used the fastest (Poisson only) simulation tools from our institute (SAP) and run different application benchmarks with an automatic benchmark system.

As our environment is on a high semantic level and also does not impose a high abstraction penalty it is easy to make special optimizations if simpler partial differential equations have to be discretized:

- There is only one solution quantity
- We only apply one kind of differential equation

Even though these conditions are not always met we can gain performance if any of them is fulfilled. The genericity of our environment allows us to use these simplifications for all kinds of differential equations.

For the testing of the finite element code we divide the simulation time into the following parts. Each of the parts is measured independently in order to show the differences.

- Preparation (I/O, mamory allocation)
- Assembly (Jacobian matrix and RHS)
- Solution

Our performance test example has to be very simple because we have to eliminate implication which result from problems with complicated geometries. Our test

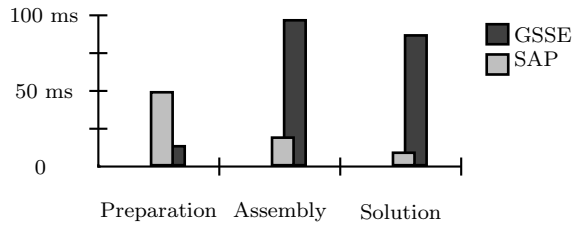

Fig. 2. Structure with 8.700 tetrahedra and 2.300 vertices.

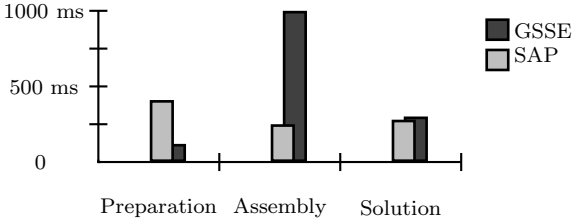

Fig. 3. Structure with 64.000 tetrahedra and 14.000 vertices.

sample is a simple via line with a length of 10μm and a square cross-section of $1\mu\text{m}^2$. For the test case we apply a potential difference of 1V. For both examples we use meshes which have been generated with `vgmodeler` [1] with 8.700, 14.000 and 97.000 tetrahedra. The comparison of the run time on an Intel Pentium 4 with 2.80GHz shows the following results (Fig. 2, Fig. 3, Fig. 4):

The benchmarks (Fig. 2, Fig. 3, Fig. 4) show that the GSSE does not have a high time consumption for the preparation of the quantities. However in assembly time as well as in solution time the highly optimized code is faster but within the same order of magnitude. The solution time shows that the TRILINOS solver package is well designed for large matrices.

VI. Conclusion

A generic environment for scientific computing has been presented. It can handle a large variety of differential equations which can be specified with different discretization schemes such as finite elements, finite volumes and finite differences. We have shown that a high semantic level does not necessarily imply an abstraction penalty so that the performance is comparable to highly optimized programs.

Even though we have shown the applicability of our environment on very simple structures it is possible to extend the features very conveniently. First the simulation domain can be taken from any meshing output. The partial differential equations can be extended to more complex models using a C++ embedded language.

Using this environment it is possible for scientists to formulate PDE problems with a full topological and geometrical support for the development of applications with minimal in-depth knowledge of internal data structures.

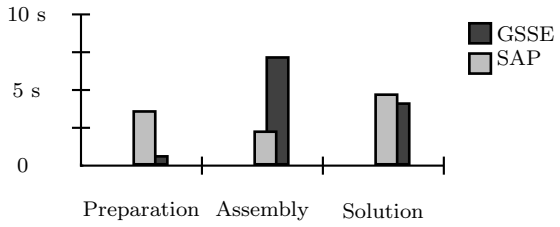

Fig. 4. Structure with 520.000 tetrahedra and 97.000 vertices.

References

[1] R. Heinzl and T. Grasser, in *Proc. Conf. in Sim. of Semiconductor Processes and Devices* (Tokio, 2005), pp. 211–214.
[2] P. Schwaha, R. Heinzl, M. Spevak, and T. Grasser, in *Proc. Conf. in Sim. of Semiconductor Processes and Devices* (Tokio, 2005), pp. 235–238.
[3] R. Sabelka and S. Selberherr, in *Proc. Intl. Interconnect Techn. Conf.* (Burlingame, California, 1998), pp. 250–252.
[4] A. Hössinger, R. Minixhofer, and S. Selberherr, in *Proc. Conf. in Sim. of Semiconductor Processes and Devices* (2004), pp. 129–132.
[5] IμE, *MINIMOS-NT 2.1 User's Guide*, Institut für Mikroelektronik, Technische Universität Wien, Austria, 2004, http://www.iue.tuwien.ac.at/software/minimos-nt.
[6] T. L. Veldhuizen, in *Proc. of PEPM'99.* (University of Aarhus, Dept. of Computer Science, 1999), pp. 13–18.
[7] J. G. Siek and A. Lumsdaine, in *ECOOP Workshop* (1998), pp. 466–467.
[8] in *Lambda-Calculus and Computer Science Theory*, Vol. 37 of *Springer series Lecture Notes in Computational Sciene (LNCS)*, edited by C. Böhm (Springer, Rome, 1975).
[9] M. A. Heroux *et al.*, ACM Trans. on Math. Software (2005), for TOMS special issue on the ACTS Collection.
[10] *IBM Visualization Data Explorer*, 3rd ed., IBM Corporation, Yorktown Heights, NY, USA, 1993.
[11] T. Grasser, T. Tang, H. Kosina, and S. Selberherr, Proc. IEEE **91**, 251 (2003).
[12] D. Scharfetter and H. Gummel, IEEE Trans. Electron Dev. **16**, 64 (1969).