Proceedings of the 6th International Caribbean Conference on Devices, Circuits and Systems, Mexico, Apr. 26-28, 2006

Multidimensional and Multitopological TCAD with a Generic Scientific Simulation Environment

René Heinzl[△], Michael Spevak[°], Philipp Schwaha[△], Tibor Grasser[△]

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

[°]Institute for Microelectronics, Technical University Vienna,
Gußhausstraße 27-29/E360, A-1040 Vienna, Austria
E-mail: {heinzl|spevak|schwaha|grasser}@iue.tuwien.ac.at

Abstract

A new approach for solving TCAD problems in a multidimensional and multitopological way based on a generic scientific simulation environment is presented. By embedding functional programming, the environment imposes no restrictions on geometry, topology, or discretization schemes. Therewith equations and even complete models can be implemented easily. The generic and functional programming paradigms and the applicability in the area of TCAD are presented.

1. Introduction

The scientific computing approach is used to gain understanding of scientific and engineering problems by the analysis of mathematical models implemented in computer programs and solved by numerical techniques. Due to the diversity of the mathematical structures, combined with efficiency considerations, in particular in three dimensions, the development of high performance simulation software is quite challenging. With modern programming techniques and new programming paradigms many tasks can be completely automated, for instance the calculation of different dimension dependent data structures.

In the field of TCAD the numerical simulation results are based on different discretization schemes such as finite differences, finite elements, and finite volumes. 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 of the simulation domain [1], so-called (unstructured) meshes or (structured) grids. For automatic mesh adaptation steps different techniques can be used to enhance the accuracy of the simulation result [2, 3].

Major problems in the development process of software for numerical problems are *testing* and *validation*. Errors are often not obvious to detect. It may already require a lot of experience to decide if a result from a simulation is erroneous or not. If the result is not correct, it may be due to a great number of reasons, e.g. a programming bug, a logical error in the program flow, or a badly chosen parameter. Therefore the availability of already tested and proven modules can not be underestimated.

These challenges are becoming more difficult to meet, when the purpose of the software is to validate novel algorithms and complex methods, or to investigate physical phenomena that are not yet understood.

We have identified the following issues for an environment suitable for scientific computing:

- Sophisticated realtime visualization possibilities
- High performance
- Abstract solver interface
- Automatic treatment of discretization schemes

To deal with all of these issues, our institute has developed different simulation environments, libraries, and applications during the last decade:

- Wafer-State-Server [4] is a complete geometrical and topological library with geometrical algorithms, interpolation mechanism and consistency checks for simplex objects, especially in three dimensions.
- STAP [5] is based on a set of high-speed simulation programs for two- and three-dimensional analysis of interconnect structures. The simulators are based on the finite element method and can be used for highly accurate capacitance extraction, resistance calculation, transient electric and coupled electro-thermal simulations.
- Minimos-NT [6] is a general-purpose semiconductor device simulator providing steady-state, transient, and small-signal analysis of arbitrary device structures. In addition, Minimos-NT offers mixed-mode device/circuit simulation to embed numerically simulated devices in circuits with compact models.
- FEDOS [7] is a finite-element based simulator for oxidation and diffusion phenomena with integrated mesh adaptation.

However, none of these simulators has proven 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.

Various research groups have put a lot of effort into the development of libraries for scientific computing or for sub-problems occurring in scientific computing like Blitz++ [8], Matrix Template Library (MTL) [9], Boost Graph Library (BGL) [10], CGAL [11], GrAL [12], Prophet [13] and deal.II [14]. During the evaluation of these libraries we found that neither of them can completely cover all the needs arising in the field of TCAD. Some of them can be used in sub-problems of TCAD, for instance for device simulation, process simulation, or Monte-Carlo analysis. Other approaches can be used for highly specialized problem areas such as finite element on unstructured meshes in three dimensions only.

Based on all these approaches, we have extracted the main concepts from our own simulation tools and have combined them with the most promising techniques proposed by other groups. The re-

1-4244-0041-4/06/$20.00 ©2006 IEEE.

sult of this work is the *generic scientific simulation environment (GSSE)*. 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. As a consequence only the required mechanisms have to be included without incurring additional dependencies. This approach guarantees a great enhancement for building libraries in terms of performance. On the other hand, generic means that all data types are parametrized and can be exchanged easily, for instance the numerical data type for quantity storage.

For this reasons, GSSE can benefit from two decades of application development in the field of numerical simulation and the concepts from many different approaches.

2. Genericity in GSSE

The approach used for the development of GSSE fundamentally relies on the generic and functional programming paradigm. Generic programming, started with the development of the STL, is an especially attractive paradigm for developing libraries for high-performance computing because it emphasizes generality and efficiency simultaneously. With this approach libraries can be coupled much more tightly to compilers, allowing libraries to provide highly-reusable data types and algorithms, as specific optimization possibilities as well.

The fundamental principle of generic programming is to separate algorithms from the concrete data structure on which they operate based on the underlying problem domain concepts, allowing the algorithms and data structures to interoperate freely.

That is, from a generic point of view, algorithms do not manipulate concrete data structures directly, but instead operate on abstract interfaces defined for entire classes of data structures.

Generic programming has proven its usefulness for a class of mathematically simple, but very important concepts, namely linear sequences within the STL. Genericity can be explained easily in terms of the STL:

- Separation of algorithms and data structures: Due to the separation of algorithms and data structures complete interoperability can be achieved. A single generic function can operate on many different data structures. The code size can be reduced from $\mathrm{O}(n\ m)$ to $\mathrm{O}(n + m)$ where n is the number of algorithms and m is the number of data structures.
- Extension through function objects: The user can adapt and customize generic algorithms through the use of function objects.
- Element type parametrization: The best known way of genericity is that its containers are parametrized on the element type.

The main issue we faced was the question whether these concepts could be extended to the complex and interdisciplinary scientific computing approach and especially the numerical solution of PDEs.

In contrast to the algorithms and data structures found in the STL, scientific computing shows a lot more diversity with respect to the needed concepts. The STL is based on linear data sequences like arrays, lists, and maps. There is always a simple topological connection inside these data sequences. Scientific computing has no restrictions on topological connections or dimensions. We have to operate on vertices, edges, rectangles, triangles, cubes, or tetrahedra. Therefore we use functional programming to specify all different types of equations in a dimensionally and topologically neutral way. This programming paradigm is explained in more detail in Section 3.3.

The language of choice was and is C++ due to a manifold of reasons. It has been shown [15] that all other languages have significant problems with generic implementations in the field of numeric algorithms. Only the multi-paradigm language C++ shows excellent efficiency capabilities. The implementation of generic programming concepts in C++ is done with parametric polymorphism [10]. Modern programming techniques [8] for C++ guarantee a performance behavior similar to Fortran code.

3. Basic Concepts

Of fundamental importance for a generic scientific simulation environment like GSSE is the separation of three different concepts:

- **Topology** means the study of manifolds and their embeddings like structured or unstructured topologies.
- **Geometry** means the properties of configurations of geometric objects, for instance cylinder coordinates for a structured topology.
- **Quantity** means all kinds of attributes or properties, which can be attached to objects (topological or geometrical).

Due to the separation, different mechanism can interact easily, for instance a cylindrical geometry can be mapped onto a structured topology.

3.1. Concepts for Topology

Most of the topologies used in TCAD and in scientific computing are based on two basic types of topology:

Structured topology: A widespread approach to spatial discretization is to divide the simulation domain into a structured assembly of quadrilateral cells, with the topological information being apparent from the fact that each interior vertex has exactly the same number of neighboring cells. This kind of discretization is called *structured grid* or simply *grid*.

Unstructured topology: The alternative approach is to divide the computational domain into an unstructured assembly of more or less arbitrarily formed cells. The topological information can not be deduced implicitly from the elements and has to be stored explicitly. This kind of discretization is called *unstructured mesh* or simply *mesh*.

3.2. Concepts for Quantities

In the area of scientific computing and especially in the field of TCAD the handling of a large number of different quantities is required. These quantities need to be stored on various objects (vertices, edges, facets, cells). On the one side, we have developed a completely generic quantity library which is capable of storing various mathematical structures in a dense and a sparse format: scalar values, vector values, matrices, and tensors.

On the other side, we have developed specializations of all of these mathematical structures to provide high performance calculations, for instance for small fixed size vectors and matrices.

3.3. Functional Programming

Functional programming is a programming paradigm in which functions are treated as regular values. Thus, we can have functions that take other functions as parameters which are called "higher-order" functions. A common feature of functions is that they can be polymorphic which means that the same function can be used with arguments of many types. This paradigm is not tied to a specific language.

The next code snippet shows how the values on an arbitrary mesh (or grid) can be accumulated in any dimensions. The

`vertex_vertex` iterator iterates over all vertices which are connected by an edge to the base vertex:

```
gsse::for_each( (*segit).vertex_begin(),
                (*segit).vertex_end(),
 pot_quan = gsse::accumulate<vertex_vertex>
            (1.0, arg1 * arg2)[pot_quan]);
```

4. Applicability

To demonstrate the applicability of GSSE we briefly present some examples. We want to emphasize the coupling of different concepts used in GSSE to support simple and robust software development in the area of scientific computing.
Support for several spatial dimensions is inherently included in such a way that programs can be written independently of the spatial dimension without unreasonable penalties on run-time and memory consumption. The property or quantity treatment is accessible in an abstract way through function objects and meta programming.
With this approach, a very high level of abstraction to all different kinds of data structures and functionality can be provided. Algorithms can be specified in a dimension independent and data structure neutral way.
To show the efficiency and applicability we present the Laplace equation in two and three dimensions, discretized by the finite volume method and calculated on a bounded domain Ω with the outer bound Γ_3. Two metal contacts are inside the domain Ω with boundaries Γ_1 and Γ_2. The problem is described by the following boundary value problem:

$$-\mathrm{div}(\varepsilon\ \mathrm{grad}(\Psi)) = 0 \qquad \text{in } \Omega \tag{1}$$

$$\Psi = \Psi_i \qquad \text{on } \Gamma_{1,2} \tag{2}$$

$$\Psi = 0 \qquad \text{on } \Gamma_3 \tag{3}$$

The $\Psi_{1,2}$ are given as boundary values.

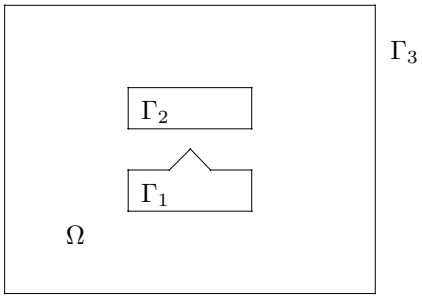

Figure 1. Domain Ω of the given problem

The domain is tesselated by cells $\mathbf{c}_i$. In two dimension, cells are triangles and in three dimensions tetrahedra. The discretized problem is given by:

$$\sum_{\text{edge}} \left(\Psi_j - \Psi_i\right) \frac{A_{ij}}{d_{ij}} = 0 \tag{4}$$

The next code snippets presents the C++ code for this discretization, where fv stands for geometrical factor $\frac{A_{ij}}{d_{ij}}$.

Listing 1: The discretization of the Laplace equation

```
for (vertex_cs
      vcs  = (*segit).vertex_begin();
      vcs != (*segit).vertex_end();
      ++vcs)
{
  // boundary evaluation

  equation=
  (
   gsse::sum<vertex_edge>
   [
     gsse::diff<edge_vertex>[pot_quan] * fv
   ]
  )(*vcs);

 // matrix assembly ..
}
```

The result of the Laplace equation is presented for two dimensions in Figure 2.

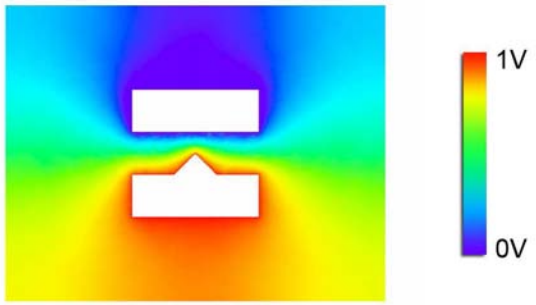

Figure 2. Two contacts with a constant potential of $\Psi_1 = 1V$ and $\Psi_2 = 0V$ and the resulting potential distribution in two dimensions.

Different error estimation techniques [2, 3] with automatic mesh adaptation steps are available within GSSE. Figure 3 shows the initial error norm with a gradient recovery method. Figure 4 shows the final error norm after three mesh adaptation steps.

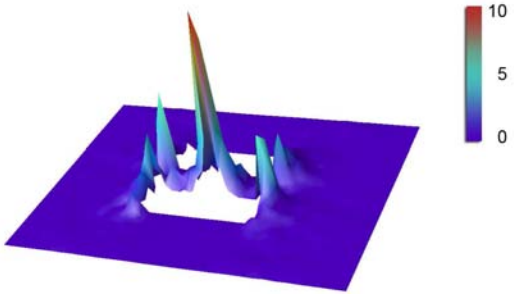

Figure 3. Resulting error norm due to the spatial discretization without mesh refinement. The error values are distributed between 10, which means a high local error norm and 1, which means a low local error norm.

The code snippet from Listing 1 can be used for three dimensional structures without modification. The simulation results for three dimensions are presented in Figure 5. An enlarged view of the critical meshed tip between the contacts can be seen in Figure 6. The corresponding isopoential areas are presented in Figure 7.

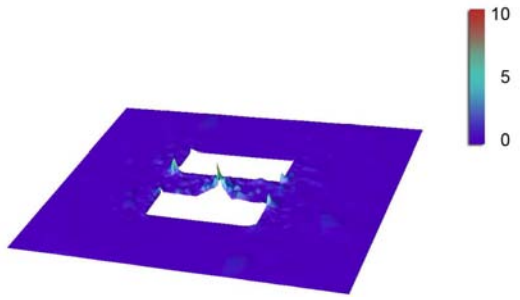

Figure 4. Reduced error norm after three mesh adaptation steps.

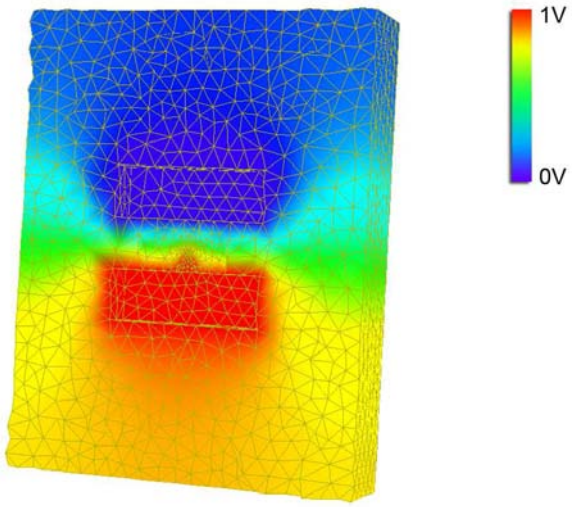

Figure 5. The same geometry as in two dimensions with a constant potential of $\Psi_1 = 1V$ and $\Psi_2 = 0V$ and the resulting potential distribution in three dimensions.

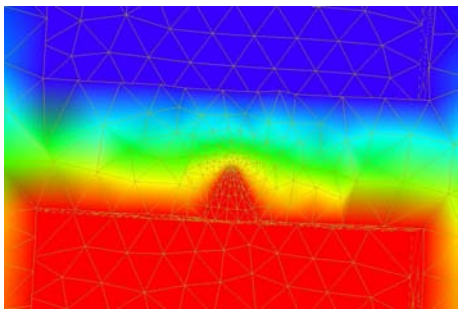

Figure 6. Enlarged view of the critical meshed tip structure between the contacts.

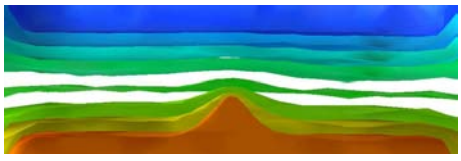

Figure 7. Isopotential surfaces between the contacts of the resulting potential distribution.

5. Conclusion

We have presented the GSSE with generic and functional programming paradigms and the applicability in a dimensional and topological neutral way. Therewith code elements and models can be implemented for different dimensions and topologies at once. As a consequence the generic concepts can be extended to the complex and interdisciplinary scientific computing approach and especially to the numerical solution of PDEs.

6.References

[1] R. Heinzl and T. Grasser, "Generalized Comprehensive Approach for Robust Three-Dimensional Mesh Generation for TCAD," in *Proc. SISPAD*, (Kobe, Japan), pp. 211–214, Sept. 2005.

[2] R. Heinzl, M. Spevak, P. Schwaha, and T. Grasser, "A Novel Technique for Coupling Three-Dimensional Mesh Adaptation with an A Posteriori Error Estimator," in *2005 PhD Research in Microelectronics and Electronics (PRIME 2005)*, (Lausanne, Switzerland), pp. 175–178, IEEE, July 25-28 2005.

[3] P. Schwaha, R. Heinzl, M. Spevak, and T. Grasser, "Coupling Three-Dimensional Mesh Adaptation with an A Posteriori Error Estimator," in *Proc. SISPAD*, (Kobe, Japan), Sept. 2005.

[4] A. Hössinger, R. Minixhofer, and S. Selberherr, "Full Three-Dimensional Analysis of a Non-Volatile Memory Cell," in *Proc. SISPAD*, (Munich, Germany), pp. 129–132, Sept. 2004.

[5] R. Sabelka and S. Selberherr, "A Finite Element Simulator for Three-Dimensional Analysis of Interconnect Structures," vol. 32, no. 2, pp. 163–171, 2001.

[6] *MINIMOS-NT 2.1 User's Guide*. Institut für Mikroelektronik, Technische Universität Wien, Austria, 2004. http://www.iue.tuwien.ac.at/software/minimos-nt.

[7] H. Ceric, *Numerical Techniques in Interconnect and Process Simulation*. Dissertation, Technische Universität Wien, 2004.

[8] T. L. Veldhuizen, "C++ Templates as Partial Evaluation," in *Proc. of PEPM'99.*, pp. 13–18, University of Aarhus, Dept. of Computer Science, Jan. 1999.

[9] J. G. Siek and A. Lumsdaine, "The Matrix Template Library: A Unifying Framework for Numerical Linear Algebra," in *ECOOP Workshops*, pp. 466–467, 1998.

[10] J. Siek, L.-Q. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, 2002.

[11] A. Fabri, "CGAL- The Computational Geometry Algorithm Library,", 2001. `citeseer.ist.psu.edu/fabri01cgal.html`.

[12] G. Berti, "GrAL - The Grid Algorithms Library," in *ICCS '02: Proceedings of the International Conference on Computational Science-Part III*, (London, UK), pp. 745–754, Springer-Verlag, 2002.

[13] C. S. Rafferty and R. K. Smith, "Solving Partial Differential Equations with the Prophet Simulator," Bell Laboratories, Lucent Technologies, 1996.

[14] `deal.II` *Differential Equations Analysis Library, Technical Reference*. `http://www.dealii.org`.

[15] R. Garcia, J. Jarvi, A. Lumsdaine, J. Siek, and J. Willcock in *Proc. of the 18th Annual ACM SIGPLAN*, (New York, NY, USA), pp. 115–134, ACM Press, 2003.