A Generic Scientific Simulation Environment for Multidimensional Simulation in the Area of TCAD

René Heinzl$^{\triangle}$, Michael Spevak$^{\circ}$, Philipp Schwaha$^{\triangle}$, Tibor Grasser$^{\triangle}$

$^{\triangle}$Christian Doppler Laboratory for TCAD in Microelectronics
at the Institute for Microelectronics
$^{\circ}$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 generic environment for process and device simulation and general scientific computing is presented that imposes no restrictions on dimension or discretization schemes. Equations and even complete models can be implemented easily without any dimensional constraints. The main paradigms, the generic approach, and the applicability to the area of TCAD are presented with multidimensional discretization schemes.

Keywords: scientific computing, high performance computing, process and device simulation

1. INTRODUCTION

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 and adaptation of the simulation domain [1], so-called unstructured meshes or structured grids.

Due to the diversity of the discretization schemes, in particular in three dimensions, the development of simulation software is quite challenging. Another circumstance, which complicates the software development process is that the problems are specified in different dimensions. Some problems can be reduced to one dimension, other problems can be solved in two dimensions, and due to the ongoing development of new devices, some problems have to be solved in three dimension. Particularly new three dimensional devices like high voltage devices can be simulated in three dimensions only.

To deal with all of these issues, our institute has developed different simulation environments, libraries, and applications during the last decade. The Wafer-State-Server [2] is a geometrical and topological library with geometrical algorithms, fast point location mechanisms, interpolation mechanisms and consistency checks for simplex objects. It was developed especially for three dimensions in C++. STAP [3] 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 [4] 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 [5] is a finite-element based simulator for oxidation and diffusion phenomena with integrated mesh adaptation (refinement and hierarchical coarsement) mechanism, implemented in C++. However, none of these libraries or simulators have 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.

Other research groups have put a lot of effort into the development of libraries for scientific computing or for sub-problems occurring in scientific computing. The most important representatives are:

- Blitz++ [6] implements basic n-dimensional arrays along with algebraic operations on them. This library has introduced new programming approaches, mainly expression templates. With these techniques, which are not limited to this library, run-time speed comparable to Fortran can be achieved.
- The Matrix Template Library (MTL) [7] contains data-structures for dense and sparse matrices, as well as generic low-level algorithms (BLAS-functionality) and a generic LU-factorization. All interfaces to data structures are implemented in a generic way with high overall performance.

- Boost Graph Library (BGL) [8] is a high-speed, container-based approach to graph relationships with a generic interface. Therewith graph algorithms are implemented generically.
- CGAL [9] implements generic classes and procedures for geometric computing with generic programming techniques (e.g. iterator access).
- GrAL [10] introduces the abstraction of mesh access mechanisms to meshes in a way similar to the C++ STL [11]. This library can be seen as a direct predecessor to our own approach.

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. But no library or approach can cover the complete spectrum for different discretization schemes. abstract access to the objects without any For that reasons, we extracted the main concepts from our own simulation tools and combined them with the most promising techniques proposed by other groups. The result of this work is a *generic scientific simulation environment (GSSE)* for multidimensional simulation in the area of TCAD.

The language of our choice is C++ due to a manifold of reasons. It has been shown [12] that all other languages have significant problems with generic implementations in the field of numeric algorithms. The implementation of generic programming concepts in C++ is done with parametric polymorphism [8] with features like *template specialization, partial specialization, partial ordering of function templates*. These features also allow techniques like *meta-programming* [13] and guarantee a performance behavior similar to Fortran code [6]. All of these techniques are currently only available in C++ due to its multi-paradigm approach [12].

2. CONCEPTS FOR MULTIDIMENSIONAL TREATMENT

To establish a multidimensional simulation approach, we have developed different concepts which are described in the next sections. To support dimensional independence, a generic topological traversal is of utmost performance, which means that the elements of a topological space (e.g. triangles, tetrahedra) must be accessible in a data structure neutral way. Algorithms or functions should not interact directly with elements. This concept of separating the access mechanism between data structures and algorithms is called iterator concept. We explain this concept in more detail in Section 2.3.

2.1 Vertex and Cell Concepts

A simulation domain D which is part of a space $\mathbf{R}^n$ where $0 < n < \infty$ is described by vertices (0-dimensional objects) and cells (n-dimensional objects) and the topological information about the connections. Dimensions higher than 3 are used for example as the phase space in Boltzmann equation.

The minimal topological information that has to be stored is *vertex on cell* connections. For an efficient traversion through all sub-dimensions, we store the *cell on vertex* information as well.

2.2 Quantity Concept

Quantities mean all kinds of attributes or properties, which can be attached to objects (topological or geometrical). 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 hand, 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 hand, we have developed specializations of all of these mathematical structures to provide high performance calculations, for instance for small fixed size vectors and matrices.

2.3 Iterator/Cursor and Property Map

In scientific computing a lot of data can be associated with objects. The basic *iterator* concept [11] can only handle one associated data item due to the linear data sequence concept of the STL. This concept is not suitable for a multidimensional scientific computing environment. Therefore we have separated the traversion of the topology from the access to the properties (similar to [14]) or quantities. To support a wide variety of traversion mechanisms, there are several hierarchies of cursors (based on [10], but implemented quite differently):

- Base cursors: vertex, edge, facet, cell cursor
- Adjacency cursors: Vertex to vertex
- Incidency cursors: Cell on vertex
- Special cursors: Boundary vertex cursor

Like the iterator concept in the STL, the cursor concept of the GSSE is the glue between the data structures (meshes and grids) and the algorithms. Therewith algorithms can be specified multi-dimensionally. In GSSE, all needed types of cursors are generated at compile time for each dimension and topology to avoid run-time overhead for any dimension or any topology.

The access to quantities is given by several property maps. The geometrical coordinates and the associated data are kept in different property maps.

2.4 Abstract Interfaces

For all visualization tasks, we have developed an abstract visualization interface. For now, IBM's data ex-

plorer [15] can be used due to its multi-dimensional and multi-topological visualization capabilities. We have extended it by an additional real-time visualization module.
Next, we have developed an abstract solver interface. At the moment, we use modules from the Trilinos project [16] in conjunction with LAM/MPI [17], which provide high performance on small single CPU clusters to large SMP machines with high speed networks, even in heterogeneous environments.

3. EXAMPLES AND RESULTS

In this section the applicability of GSSE is demonstrated with resept to an example. We 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 penalties on run-time and memory consumption due to the template meta-programming technique. The key feature to achieve this level of abstraction is the cursor concept which encapsulate all topological data structure elements. Algorithms can be specified in a dimension neutral way.

3.1 Multidimensional Laplace Code

The applicability is shown with the Laplace equation within an arbitrary dimensional formulation, discretized by the finite volume method and calculated on a bounded domain Ω. Dirichlet boundary conditions $\Psi_{1,2}$ are given on Γ_1 and Γ_2.

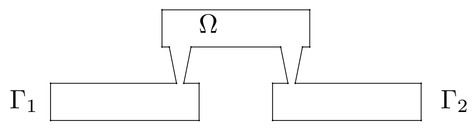

Figure 1: Domain of the given problem

The domain is tesselated by cells $\mathbf{c}_i$. In two dimensions, cells are triangles and in three dimensions tetrahedra. The problem is described by the boundary value problem:

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

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

The discretized problem is given by:

$$\sum_{\text{edge}} \left(\Psi_j - \Psi_i\right)\frac{A_{ij}}{d_{ij}} = 0 \qquad \text{in } \Omega, \qquad (3)$$

where $\left(\Psi_j - \Psi_i\right)$ stands for the difference of the potential for each vertex pair v_i, v_j on an edge (Figure 2).

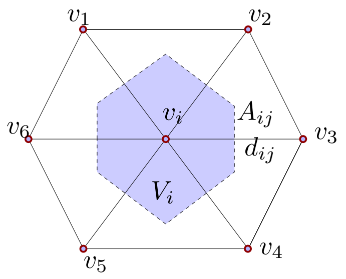

Figure 2: A two dimensional triangle patch with the corresponding dual Voronoi graph (blue)

To avoid the complex syntax from C++ code we use a transformation layer to remove the last burden of learning a complex language for the task of specifying physical problems. This meta language (Listing 1) is very near to the real syntax of our approach (Listing 2).

Listing 1: The discretization of the Laplace equation (meta language)

```
model LaplaceEquationFVM
{
 assemble
 {
  linearized_equation   equ;

  eqn=sum(V->E)[diff(E->V)[pot_quan] * A/d];

 }
}
```

Listing 2: The discretization of the Laplace equation (C++)

```
// boundary evaluation

 equ=
 (
   gsse::sum<vertex_edge>
   [
    gsse::diff<edge_vertex >[pot_quan] * A/d
   ]
 ) (vertex);

// matrix assembly ..
```

Here an equation object `equ` is specified functionally, which means, that a complete matrix line is covered by the object. The `gsse::sum` and `gsse:diff` are extensions to the Boost phoenix library to support our cursor concept. Each of these algorithms is specialized with a cursor to access different traversal mechanism (`vertex_edge` means that the traversal performed over all edges attached to a vertex, `edge_vertex` means, that the incident vertices to an edge are used).

Figure 3 presents the result of a two-dimensional simulation with a sparse mesh, and Figure 4 the same simulation with a dense mesh. In Figure 5 and 6 the results within three dimensions can be seen.

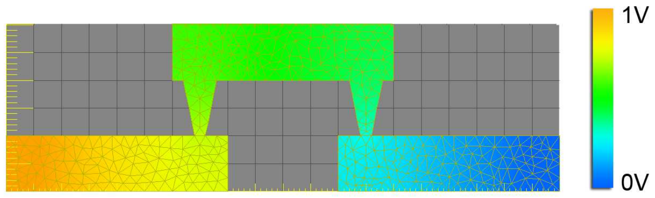

Figure 3: Potential distribution for a two dimensional structure with a sparse mesh.

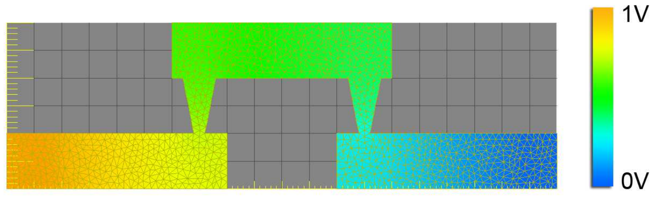

Figure 4: Potential distribution for a two dimensional structure with a dense mesh.

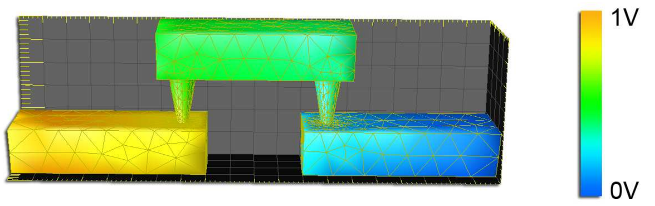

Figure 5: Potential distribution for a corresponding three dimensional structure with a sparse mesh.

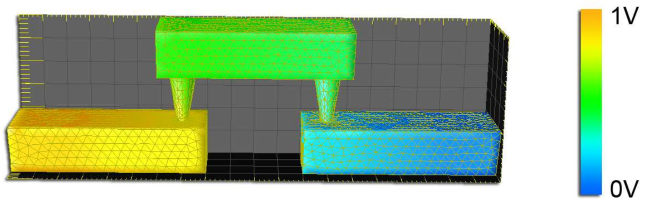

Figure 6: Potential distribution for a corresponding three dimensional structure with a dense mesh.

4. CONCLUSION

We have developed a generic scientific simulation environment with the separation of topology, quantity, and geometry. With our cursor concept, problems can be specified in a dimensional neutral way. Finally we have presented the applicability of our approach with an example from interconnect simulation.

6. REFERENCES

[1] R. Heinzl and T. Grasser, in *Proc. SISPAD* (Kobe, Japan, 2005), pp. 211–214.

[2] A. Hössinger, R. Minixhofer, and S. Selberherr, in *Proc. SISPAD* (Munich, Germany, 2004), pp. 129–132.

[3] R. Sabelka and S. Selberherr, in *Proc. Intl. Interconnect Technology Conference* (Burlingame, California, 1998), pp. 250–252.

[4] 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.

[5] H. Ceric, Dissertation, Technische Universität Wien, 2004.

[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 Workshops* (1998), pp. 466–467.

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

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

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

[11] M. H. Austern, *Generic Programming and the STL: Using and Extending the C++ Standard Template Library* (Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998).

[12] R. Garcia *et al.*, in *Proc. of the 18th Annual ACM SIGPLAN* (ACM Press, New York, NY, USA, 2003), pp. 115–134.

[13] D. Abrahams and A. Gurtovoy, *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond (C++ in Depth Series)* (Addison-Wesley Professional, 2004).

[14] D. Abrahams, J. Siek, and T. Witt, Technical Report No. N1477 03-0060, ISO/IEC JTC 1, Information Technology, Subcommittee SC 22, Programming Language C++ (unpublished).

[15] *IBM visualization Data Explorer*, 3rd ed., IBM Corporation, Yorktown Heights, NY, USA, 1993.

[16] M. Heroux *et al.*, Technical Report No. SAND2003-2927, Sandia National Laboratories (unpublished).

[17] J. M. Squyres and A. Lumsdaine, in *Proceedings, 10th European PVM/MPI Users' Group Meeting*, No. 2840 in *Lecture Notes in Computer Science* (Springer-Verlag, Venice, Italy, 2003), pp. 379–387.