A NOVEL HIGH PERFORMANCE APPROACH FOR TECHNOLOGY COMPUTER AIDED DESIGN

René HEINZL, Doctoral Degree Programme (2)
Christian Doppler Laboratory at the IUE, TU Wien
E-mail: heinzl@iue.tuwien.ac.at

Michael SPEVAK, Doctoral Degree Programme (1)
Institue for Microelectronics, TU Wien
E-mail: spevak@iue.tuwien.ac.at

Philipp SCHWAHA, Doctoral Degree Programme (2)
Christian Doppler Laboratory at the IUE, TU Wien
E-mail: heinzl@iue.tuwien.ac.at

Supervised by: Prof. Tibor Grasser

ABSTRACT

A generic scientific simulation environment is presented which offers topological, dimensional, and functional independence. Therewith complete mathematical formulas and discretization schemes like finite volumes or finite elements can be expressed directly in C++. The new approach as well as the applicability and the performance related to well established simulators are highlighted.

1 INTRODUCTION

Semiconductor devices have become an ubiquitous commodity and users have come to expect a constant increase of device performance. It is this quest for ever decreasing device dimensions and faster switching speeds that results in ever growing requirements on the employed simulation methodology. While computer performance is steadily increasing the additional complexity of these simulation models easily outgrows this gain in computational power. It is therefore of utmost importance to employ the latest techniques of software development to obtain high performance and thereby ensure adequate simulation times even for complex problems. As a consequence the development of high performance simulation software is quite challenging.

2 MOTIVATION

Different approaches [1, 2, 3] focus on topics like expression templates, high performance matrix manipulation, geometrical algorithms, and finite element discretization. The nature of dealing with partial differential equations (PDEs) with the inherent coupling of topological traversion and functional description complicates the usage of these libraries. During the last decades our institute has developed different applications to tackle these issues like SAP [4], Minimos-NT [5].

An analysis reveals that so far no approach can successfully deal with the mathematical formulation of physical phenomena within a programming language efficiently. As a consequence we have extracted the most promising concepts from all of these approaches. The result of this analysis is a *generic scientific simulation environment (GSSE)*. Concepts from the Boost MPL [6] and Boost Phoenix [7] libraries have been used to optimize the run-time performance of GSSE.

3 OUR APPROACH

To support arbitrary topological traversal mechanisms and functional description capabilities a layered concept structure has been developed (Figure 1). The lowest layer represents the concepts for topological information and quantity storage. The second layer provides the separation of traversion and quantity access which is called cursor and property map concept. The functional layer decouples the underlying concepts from the top layer and is based on the Boost Phoenix library [7]. It provides several facilities to support different discretization schemes. To complete our environment abstract interfaces for (real-time) visualization [8] and solver integration [9] as well as three dimensional unstructured meshing [10] have been developed. To demonstrate the importance of a method that is both easy and efficient a few examples are given using equations encountered in the field of TCAD.

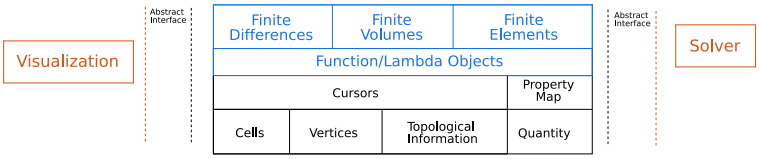

Figure 1: Conceptual view of GSSE with abstract interfaces.

3.1 LAPLACE EQUATION

The Laplace equation is used as a simple introduction. It is discretized by the finite volume method and calculated on a bounded domain Ω (Figure 2). Discretized equations can be specified without special treatment of the spatial dimension thereby optimizing flexibility without incurrying any run-time penalty. The resulting problem is given by:

$$\sum_{\text{edge}} \left(\Psi_j - \Psi_i\right) \frac{A_{ij}}{d_{ij}} = 0 \tag{1}$$

The next code snippets presents the corresponding C++ code.

```
for (vertex_cs vcs  = (*segit).vertex_begin();
               vcs != (*segit).vertex_end();  ++vcs)
{
  equation= ( gsse::sum<vertex_edge>
            [
              gsse::diff<edge_vertex>[pot_quan] * A / d
            ]  )(*vcs);
}
```

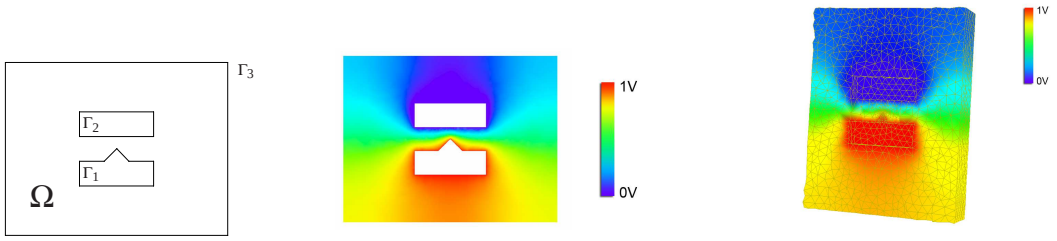

Figure 2: Left: domain Ω with boundaries. Middle and right: potential distribution obtained from a two and three dimensional simulation run.

Different error estimation techniques [11] with automatic mesh adaptation steps are available in GSSE and the results are shown in Figure 3. These techniques are based on an orthogonal approach for mesh refinement, which means that different error estimation and mesh adaptation modules can be easily combined and of course integrated in other software environments.

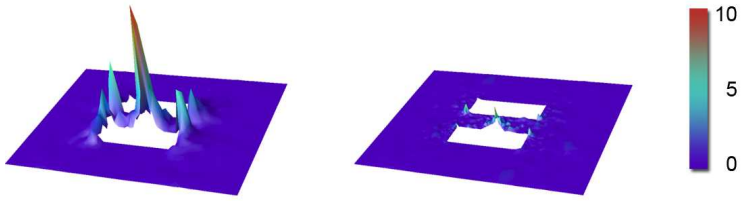

Figure 3: Left: resulting error norm before mesh refinement. Right: error norm after three mesh adaption steps. The error values are distributed between 10, which means a high local error norm and 0, which means a low local error norm.

3.2 BOLTZMANN'S EQUATION

Boltzmann's equation for electron transport in semiconductors, as given in Equation 2, is the base for many calculations for device simulations. Here f is the distribution function and $\vec{v}$ the velocity of the charge carriers, while $\vec{F}$ describes the force of an electric field on these particles.

$$\frac{\partial}{\partial t} f + \vec{v} \cdot \mathrm{grad}_r f + \vec{F} \cdot \mathrm{grad}_k f = \frac{\partial}{\partial t} f|_{collisions} \tag{2}$$

Due to the complexity of Boltzmann's equation several techniques have been developed to obtain simplified models. One of these is the drift diffusion model [12], that can be derived from Equation 2 by applying the method of moments. This results in current relations of the following form shown in Equation 3.

$$\vec{J}_\mathrm{p} = \mathrm{q}\, p\, \mu_\mathrm{p}\, \mathrm{grad}\, \Psi - \mathrm{q} D_\mathrm{p}\, \mathrm{grad}\, p \qquad \vec{J}_\mathrm{n} = \mathrm{q}\, n\, \mu_\mathrm{n}\, \mathrm{grad}\, \Psi + \mathrm{q} D_\mathrm{n}\, \mathrm{grad}\, n \tag{3}$$

$$\mathrm{div}\ (\mathrm{grad}\, \varepsilon\, \Psi) = -\rho \tag{4}$$

Such non-linear equation systems [12] need to be solved self consistently with Poisson's equation, given in Equation 4 and discretized using the Scharfetter-Gummel scheme. Using our functional formalism this non trivial problem can be implemented straightforwardly in C++. The source code is given in the following code snippet.

```
// Poisson equation
equation_pot = (sum<vertex_edge>
[
  diff<edge_vertex>[pot_quan] * area / dist
] + ( n - p + nA - nD) * ( vol * q / (eps0 * epsr)))  (vertex);
// Continuity equation for electrons
equation_n = (sum<vertex_edge>
[
  diff<edge_vertex>(-n_quan*Bern(diff<edge_vertex>[ pot_quan]/U_th),
                    -n_quan*Bern(diff<edge_vertex>[-pot_quan]/U_th))
                    * q * mu_n * U_th * area/dist
]) (vertex);
```

To demonstrate that the proposed scheme is indeed operational we provide the potential of a pn diode in Figure 4 at different stages of calculation. While the example has been provided in two dimensions, the above implementation makes no assumptions about the dimension of the problem and is suitable for any dimension.

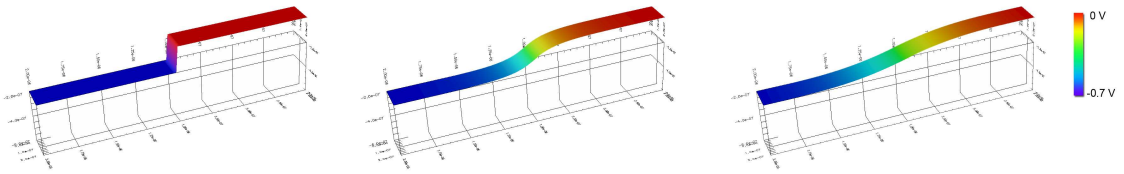

Figure 4: Potential of a pn diode during different stages of the Newton iteration. From initial (left) to the final result (right).

4 PERFORMANCE

Due to the use of meta programming [6] and its evaluation at compile time the calculation associated with the specified equations can be highly optimized by the compiler and thereby ensures excellent run-time performance. The run-time behavior is analyzed with a simple vector addition $\mathbf{A}_f = \mathbf{A}_1 + \mathbf{A}_2 + \mathbf{A}_3$ with three operations in Figure 5 in relation to Fortran 77 code. The first implementation is based on the standard `valarray` datatype as implemented by the GNU GCC. Secondly, we utilize the Blitz++ [1] library, which introduced high performance calculation directly in C++. Lastly, a naive C++ implementation is used that creates two temporary objects, one for addition and one for assignment. For vector lengths smaller than 10^4, cache hits reveal the full computation power of the CPU, longer vectors show the limits imposed by memory bandwidth. Different computer architectures are compared (AMD64 and Apple G5). Investigations of parallelization attempts on multi-processor machines (G5) show that the inner loop of the finite element assembly cannot be parallelized easily. On the one hand, the update mechanisms of the element matrices can demand access to the same part of memory simultaneously, which could be avoided by a different assembly scheme, e.g. node-based assembly. On the other hand, the inner loops are compiled very efficiently and only bounded by memory bandwidth.

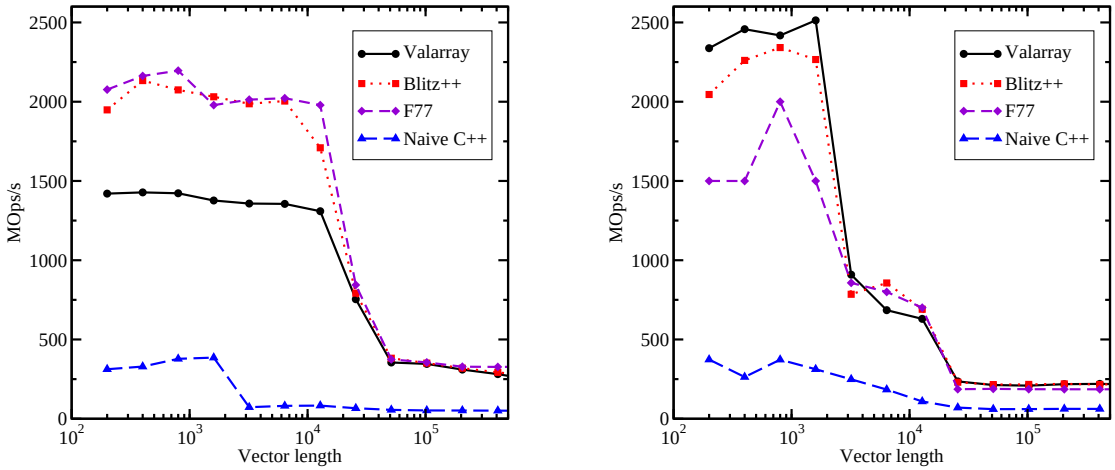

Figure 5: Left (G5) and right (AMD64): different programming approaches on modern architectures compared to hand-optimized Fortran 77 performance.

5 CONCLUSION

In summary, a generic environment was presented. To show the applicability of our approach the Laplace and drift-diffusion equations were implemented using the finite volume discretization scheme. Therewith we can show, that highly expressive code in C++ on different computer architectures does not show any abstraction penalty and thus run-time comparable to Fortran can be achieved by carefully designing the individual components.

REFERENCES

[1] Veldhuizen T. L., in Proc. of PEPM'99., pp. 13–18.

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

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

[4] Sabelka R. and Selberherr S. , Microelectronics Journal 32 (2001),2: 163-171.

[5] MINIMOS-NT 2.1 User's Guide,http://www.iue.tuwien.ac.at/software/minimos-nt.

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

[7] `Phoenix`, `http://spirit.sourceforge.net/`.

[8] IBM Visualization Data Explorer, 3rd ed., IBM Corporation, Yorktown Heights, NY, USA, 1993.

[9] Heroux M. A. et al., ACM Transactions on Mathematical Software , for TOMS special issue on the ACTS Collection.

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

[11] Heinzl R., Spevak M., Schwaha P., and Grasser T., in PRIME 2005 (IEEE, Lausanne, Switzerland, 2005), pp. 175–178.

[12] Selberherr S., Analysis and Simulation of Semiconductor Devices (Springer, Wien–New York, 1984).