A High Performance Generic Scientific Simulation Environment

René Heinzl[1], Michael Spevak[2], Philipp Schwaha[1], and Tibor Grasser[1]

[1] Christian Doppler Laboratory at the Institute for Microelectronics, TU Vienna
heinzl@iue.tuwien.ac.at,
[2] Institute for Microelectronics, TU Vienna, spevak@iue.tuwien.ac.at,

Abstract. A generic scientific simulation environment is presented which imposes no restriction in topological, dimensional, and functional issues. Therewith complete discretization schemes like finite volumes or finite elements can be expressed directly in C++. The new approaches as well as the applicability and the performance related to well established simulators are highlighted.

1 Introduction

In the last decades different libraries and software environments have been developed to handle various areas in the field of scientific computing. 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. 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 fully understood.

High performance computations have turned the attention especially to C++ since Blitz++ has shown that the run-time behavior is comparable to Fortran [1]. On the other hand different programming paradigms, some, such as functional programming, only available through libraries [2, 3], are well supported.

2 Motivation

Different library approaches [4–8] focus on topics like expression templates, high performance matrix treatment, and finite element discretization. An analysis reveals that so far no approach can successfully deal with the mathematical formulation of physical phenomena in a broad regime directly in C++. As a consequence we have extracted the most promising approaches from all of these approaches. The base concept of separating topology and quantity is based on the grid algorithm library (GrAL) [9].

The result of this analysis is a *generic scientific simulation environment (GSSE)* which does not impose any restriction on topological treatment or functional description with an overall high performance.

3 Our Approach

To support arbitrary topological traversal mechanisms and functional description capabilities a layered concept structure has been developed (Figure 1).

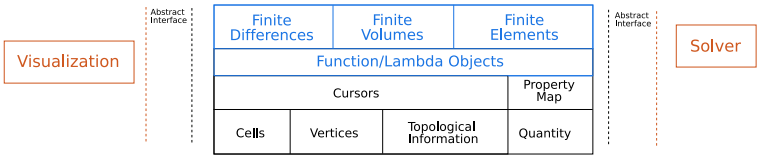

Fig. 1. Complete conceptual view of GSSE with abstract interfaces.

The bottom layer represents the concepts for topological information and quantity storage, whereas the second layer provides the separation of traversion and quantity access [10] which is called cursor and property map concept. The functional layer decouples the underlying concepts from the top layer which is based on the Boost Phoenix library [3] with several enhancements to support different discretization schemes.

A functional encapsulation is provided for the property map to create uniform access, which is called quantity accessor mechanism (e.g. the `pot_quan` usage in the following listing). To complete our environment abstract interfaces for (realtime) visualization [11] and solver integration [12] have been developed as well as three dimensional unstructured meshing issues [13].

4 Applicability

Due to space considerations we demonstrate the applicability of GSSE with a simple example only. The Laplace equation is used and discretized by the finite volume method and calculated on a bounded domain Ω (Figure 2). The problem is described by the following boundary value problem:

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

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

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

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] * fv
              ]  )(*vcs);
}
```

The potential distribution in the domain Ω for two and three dimensions can be observed in Figure 2. Different error estimation techniques [14] with automatic mesh adaptation steps are available in GSSE which are shown in Figure 3.

5 Performance

To achieve overall high performance we use template meta-programming to derive all cursor data types at compile-time. The run-time behaviour is analysed with a simple vector addition $\boldsymbol{A_f} = \boldsymbol{A_1} + \boldsymbol{A_2} + \boldsymbol{A_3}$, which is used in the finite element assembly with three operations (two additions and one assignment) in Figure 4 in relation to Fortran 77 code. The y-axes depict million operations per seconds. The legend text `ET` means a simple expression template mechanism [15], `FP` stands for our functional approach, `Blitz++` for the corresponding library, and `naive C++` means a simple STL vector class with temporary objects.

For an overall comparison of our approach we compare the system matrix assembly of a finite element discretization for the Laplace equation in assembled equations per seconds in Figure 5. The inhouse developed *smart analysis programs* (SAP) [16] are used as a reference. This package is designed specifically for highly accurate analysis of multi-layer VLSI interconnect structures with an overall high performance.

All benchmarks were performed on a Pentium4 2.8 GHz, 2 GB RAM with GCC 4.0.2.

Notes and Comments. To show the applicability of our approach the Laplace, diffusion, and drift-diffusion equations with the finite volume and the Laplace and diffusion equations with finite element discretization schemes in arbitrary dimensions and on structured as well as on unstructured meshes are implemented. As can be seen the run-time speed of C++ code does not contradict to high expressiveness.

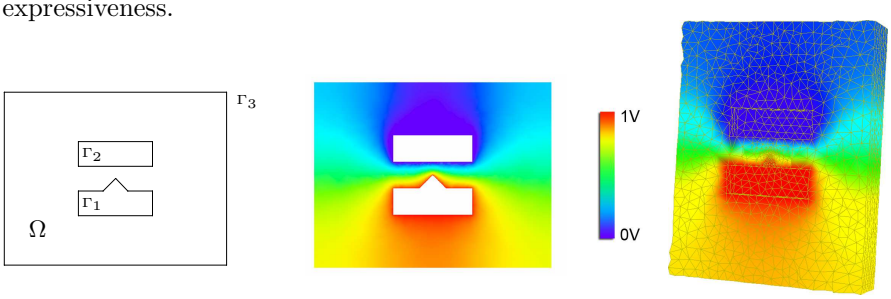

Fig. 2. Two contacts with a constant potential of $\Psi_1 = 1V$ and $\Psi_2 = 0V$ and the resulting potential distribution in two and three dimensions.

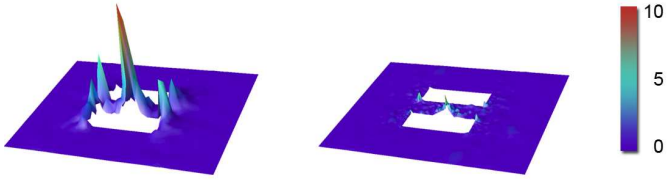

Fig. 3. Left: resulting error norm without mesh refinement. Right: error norm after three mesh adaption steps. The error values are distributed between 10, which means a high local error norm and 0, which means a low local error norm.

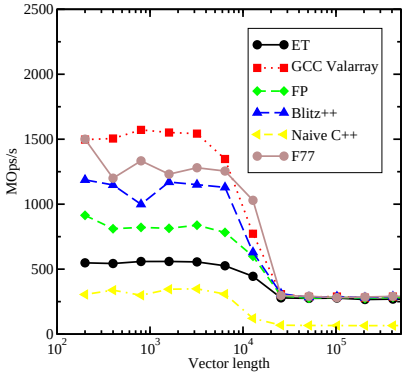

Fig. 4. Matrix addition benchmark with different approaches in C++

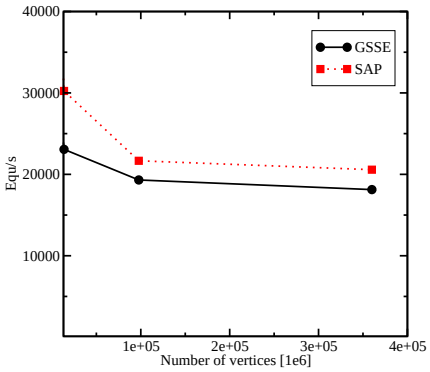

Fig. 5. Finite element assembly benchmark for SAP and GSSE

References

1. T. Veldhuizen, C++ Report **7**, 36 (1995), reprinted in C++ Gems, ed. Stanley Lippman.
2. B. McNamara and Y. Smaragdakis, **36**, 25 (2001).
3. `Phoenix`, `http://spirit.sourceforge.net/`.
4. T. L. Veldhuizen, in *Proc. of PEPM'99.* (University of Aarhus, Dept. of Computer Science, 1999), pp. 13–18.
5. J. G. Siek and A. Lumsdaine, in *ECOOP Workshops* (1998), pp. 466–467.
6. J. Siek, L.-Q. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual* (Addison-Wesley, 2002).
7. A. Fabri, CGAL- The Computational Geometry Algorithm Library, 2001, `citeseer.ist.psu.edu/fabri01cgal.html`.
8. W. Bangerth, R. Hartmann, and G. Kanschat, `deal.II` *Differential Equations Analysis Library, Technical Reference*, `http://www.dealii.org`.
9. G. Berti, in *ICCS '02: Proceedings of the International Conference on Computational Science-Part III* (Springer-Verlag, London, UK, 2002), pp. 745–754.
10. 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).
11. *IBM Visualization Data Explorer*, 3rd ed., IBM Corporation, Yorktown Heights, NY, USA, 1993.
12. M. A. Heroux *et al.*, ACM Transactions on Mathematical Software , for TOMS special issue on the ACTS Collection.
13. R. Heinzl and T. Grasser, in *Proc. SISPAD* (Kobe, Japan, 2005), pp. 211–214.
14. R. Heinzl, M. Spevak, P. Schwaha, and T. Grasser, in *2005 PhD Research in Microelectronics and Electronics (PRIME 2005)* (IEEE, Lausanne, Switzerland, 2005), pp. 175–178.
15. D. Abrahams and A. Gurtovoy, *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond (C++ in Depth Series)* (Addison-Wesley Professional, 2004).
16. R. Sabelka and S. Selberherr, in *Proc. Intl. Interconnect Technology Conference* (Burlingame, California, 1998), pp. 250–252.