Parallel Library-Centric Application Design by a Generic Scientific Simulation Environment

René Heinzl, Philipp Schwaha, Franz Stimpfl, and Siegfried Selberherr

Institute for Microelectronics, TU Wien, Gußhausstraße 27-29, Vienna, Austria

1 Introduction

The increasing complexity of physical models often brings also an increase of the complexity and amount of source code, thereby essentially increasing the importance of efficiently developing and maintaining source code. This issue can be addressed by providing modular building blocks which can be tested and refined independently of each other and seamlessly integrated into the desired applications. Hence, the concept of library-centric application design [1, 2] and the availability of a set of high performance libraries significantly eases the development of highly scalable applications.

To secure a further gain in computing performance the semiconductor industry has shifted the upcoming processor upgrades to multi-core computer systems, where the gain in processing power is obtained by using an increased number of processing cores in the CPUs. Current single desktop computer systems can already handle a large amount of scientific simulations locally. Nevertheless, industrial problems and settings often require large scale simulations which have to be performed on supercomputers, where the individual nodes of these supercomputers often do not differ in the execution speed from the desktop computers anymore. Instead they are heavily parallelized with a large amount of shared memory. Only for the final industrial setting or final example, the application has to run on a supercomputer, where the amount of CPUs is drastically increased, as is the amount of available memory. This scaling also has to be reflected in nowadays application design.

While scientific simulations have been among the first applications to embrace parallelization, still not all fields of scientific computing make use of it. Most related work therefore focuses on parallel toolkits within their frameworks [3]. Our approach is based on providing modular blocks which can be used on top of existing libraries, such as the Boost graph library (BGL [4]) or CGAL [5]. Also utmost emphasis is placed on the issues that already tested and stable code has to be parallelized without modification of existing source code. We therefore present two approaches with our Generic Scientific Simulation Environment (GSSE) [2, 6]:

- Various multi-threading libraries are used in conjunction with the topological partitioning provided by GSSE to subdivide the amount of topological objects. Several discretization schemes and the assembly times benefit greatly from this approach.
- Recent developments towards parallel STL [7, 8] techniques or libraries can easily be incorporated, which require a recompilation step, where all STL algorithms and our GSSE algorithms, which are built on top of these algorithms, are then executed parallely. These techniques are already going to be incorporated into the GCC [9].

These approaches enable the utilization of several parallelization techniques without altering the developed, tested, and calibrated applications by simple re-compilation steps.

2 The Parallel GSSE

Our approach deals with the identification and implementation of building blocks for an easy specification of various types of discretized differential equations, reducing error prone tasks such as index calculations or the evaluation of the elements of the Jacobian matrix, while at the same time not sacrificing run-time performance [6, 10].

Several tasks in scientific computing can be considerably eased by the utilization of functional programming. Unfortunately several tasks defy the nature of stateless description. All different types of storage mechanisms as well as streaming processes cannot be easily described by functions. Here, the actually stored elements are the important parts and not their functional description. So, our approach is a multi-paradigm approach, where each paradigm is used, where it performs best. The object-oriented paradigm is used, where hierarchies of data types are relevant, e.g., data type selections or additional properties of data. The functional programming paradigm is best at describing functional expressions, in our case discretized and linearized projections of the continuous function spaces. The generic programming paradigm couples the object-oriented and functional paradigm. Generic programming excels at the abstract treatment of objects, called concepts.

2.1 Topological Traversal and Functional Description

The C++ STL offers great mechanisms for sequential containers and the corresponding algorithms, but for more complex data structures a common way of accessing data or iteration is not available at the moment. Different developments, such as the BGL or CGAL, offer their own mechanisms, derived from the STL. The GSSE offers a common topological approach, the generic topology library (GTL [2]), where all data structures are mapped to a cell complex and the corresponding mechanisms are derived from algebraic topology. The generic functor library (GFL [2]) built on top of the topological interface is then dimensionally and topologically independent. An example of these two libraries is given next, where a finite volume discretization of a generic Poisson equation is discretized:

```
equ = (sum<edge>()
[ sum<vertex>()  [ equ_pot ] * area / dist ] + rho * vol
)  (*v_it);
```

The functional body can be arbitrarily extended by other traversal operations, calculations, or assignments. This example is inherently fully parallelizable due to the functional specification, where `v_it` represents a vertex iterator, an object from the traversal space. The parallel version of the GSSE introduces additional mechanisms on top of the already existing libraries in a non-intrusive way by partitioning the this traversal space:

```
for (vertex_iterator v_it = (iter_part).vertex_begin(threadID);
                     v_it != (iter_part).vertex_end(threadID); ++v_it)
```

The assembly algorithms remain unchanged for parallelization, only the topological traversal is partitioned automatically, e.g., an environment variable changes the number of parallel execution tasks.

2.2 Parallel STL

Various approaches extend the C++ STL by parallel execution paths. The GSSE offers the same concept but in a more general way which separates the discrete topological space (elements of a data structure) and the access to quantities. The following example illustrates the calculation for all edge lengths of a more complex container structure.

```
traverse<edge>()
  [ dist = norm ( sum<vertex>() [  coord  ] ) )(container);
```

The implementations of these traversal mechanisms use the C++ STL algorithms internally and are thereby automatically parallelized by utilizing one of the parallel STL approaches. A linear speed-up corresponding to the number of cores can be accomplished by just a recompilation step and adjusting a run-time environment variable.

3 Examples

To present the application of the parallelization techniques we choose the area of Technology Computer Aided Design (TCAD), which serves as the semiconductor industry's branch of scientific computing.

3.1 Discretization Schemes

The following source snippet presents the parallel GSSE approach for an implicit surface evolution, finite-difference (levelset) simulation. Here the topological vertex traversal space is partitioned automatically by the number of available CPUs or threads.

```
for (vertex_iterator vit = (iter_part).vertex_begin(threadID);
                     vit != (iter_part).vertex_end(threadID); ++vit)
     updatePoint(domain, v_it, cfl_timestep, speed_function);
```

The result of the deposition simulation is not only calculated concurrently, but also meshed parallely [11]. Due to the partitioned traversal space and the independence of the levelset function, a linear performance gain related to the number of available cores can be achieved. Finite-volume schemes can also be traversed in parallel by a partitioned traversal space due to the line-wise entries of the stencil matrix [2, 12].

3.2 Mesh Generation and Adaptation

Another example for the utilization of a parallel topological traversal is given by the task of Delaunay mesh generation and adaptation. Here we present a parallel combined Delaunay and advancing front mesh generation and adaptation approach. The complete hull is pre-processed separately to comply with the Delaunay property [11]. This guarantees a volume mesh generation approach, where each segment can be meshed concurrently. The following snippet of code shows a central part of the mesh generation application:

```
  gsse::for_each(container.segment_begin(),
                 container.segment_end(),
                 generate_mesh(thread_id++));
```

A GSSE container is used as an interface for segments which are fed to a functional meshing routine.

Figure 1 depicts a three-dimensional device structure (MOSFET), which can be calculated on a workstation computer, whereas the full device is then simulated on four AMD 4xDual-Core Opterons 8222 SE with 2x32 GByte and 2x16GByte RAM.

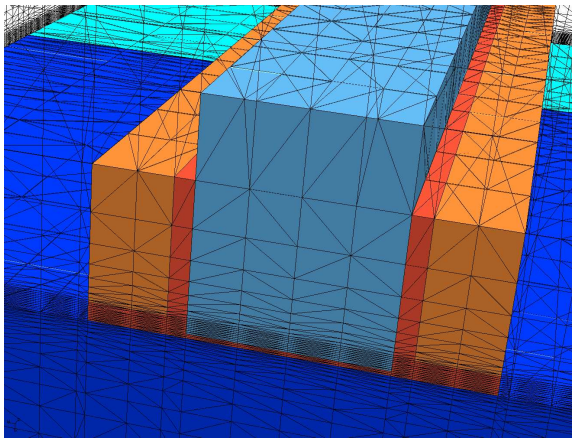

Fig. 1: A three-dimensional device structure for a MOSFET with an additional externally supplied point cloud. The important part is the regularity of the elements in the channel region (thin layer of mesh elements in the center). The different aspect ratios are also an additional complication for the mesh generation algorithm.

Example	Sequential mesh	Dual-core	Quad-core	Num. points	Num. segments
MOSFET (industrial)	172 s	101s	101 s	1.7e6	7
MOSFET	70 s	42s	21 s	3.6e5	7

Table 1: Comparisons of the mesh generation and included mesh adaptation times (in seconds) on AMD's X2 6000 and AMD X4 Phenom 9600 (quad-core).

4 Conclusion

Library-centric design does not only ease the development of applications significantly by providing building blocks centralized in a generic environment, the GSSE. Also the evolution of single-processing applications into parallel applications suitable for multi-core processors is significantly supported by parallel components, thereby simplifying development, scalability, stabilization, further support, and parallelization. Only applying the concepts of parallelism in everyday programming can unlock the full power of the new multi-core processors.

5 Acknowledgment

This work has been supported by the Austrian Science Fund FWF, project P19532-N13, and the Intel Corporation.

References

1. Heinzl, R., Schwaha, P., Selberherr, S.: Modern Concepts for High-Perfomance Scientific Computing. In: Proc. of the 2nd ICSOFT 2007, Barcelona, Spain (July 2007) 100–107
2. Heinzl, R.: Concepts for Scientific Computing. Dissertation, Technische Universität Wien, Austria (2007)
3. Kagstrom, B., Elmroth, E., Dongarra, J., (ed.), J.W.: Applied Parallel Computing. State of the Art in Scientific Computing. Berlin / Heidelberg (2007)
4. Siek, J., Lee, L.Q., Lumsdaine, A.: The Boost Graph Library: User Guide and Reference Manual. Addison-Wesley (2002)
5. Fabri, A.: CGAL - The Computational Geometry Algorithm Library. In: Proc. of the 10th Intl. Meshing Roundtable, CA, USA (2001) 137–142
6. Heinzl, R., Schwaha, P., Selberherr, S.: A High Performance Generic Scientific Simulation Environment. In B. Kaagström et al., ed.: Lecture Notes in Computer Science. Volume 4699/2007. Springer, Berlin (2007) 781–790
7. Putze, F., Sanders, P., Singler, J.: MCSTL: The Multi-Core Standard Template Library. In: Proc. Symposium on Principles and Practice of Parallel Programming, New York, NY, USA, ACM (2007) 144–145
8. Singler, J., Sanders, P., Putze, F.: The Multi-Core Standard Template Library. In: Lecture Notes in Computer Science. Volume 4641/2007. Springer, Berlin (2007) 682–694
9. Singler, J., Kosnik, B.: The libstdc++ Parallel Mode: Software Engineering Considerations. In: Proc. of IWMSE 2008. (2008)
10. Heinzl, R., Spevak, M., Schwaha, P., Selberherr, S.: A Generic Topology Library. In: Proc. of the Object-Oriented Programming Systems, Languages, and Applications Conf., Portland, OR, USA (October 2006) 85–93
11. Stimpfl, F., Heinzl, R., Schwaha, P., Selberherr, S.: High Performance Parallel Delaunay Mesh Generation and Adaptation. In: Proc. of the PARA Conf., Trondheim, Norway (May 2008)
12. Spevak, M.: On the Specification and the Assembly of Discretized Differential Equations. Dissertation, Technische Universität Wien, Austria (2008)