A Parallel Generic Scientific Simulation Environment

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

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

Abstract. Upcoming parallelization of CPU cores as well as the availability of large networks with high-speed interconnects require a change in the used programming paradigms. The utilization of many-core processors places new difficulties in the area of application design in the field of scientific computing. Multi-dimensional topological traversal mechanism are presented as well as inherent parallel assembly mechanisms for large equations systems.

1 Introduction

The current evolutionary shift of how the computational power of systems increases requires the adoption of new programming methodologies and libraries. Only by applying the concepts of parallelism in everyday programming can the full power of the new processors be unlocked. While scientific simulations have been among the first applications to embrace parallelization, still not all fields of scientific computing make use of it, as it is perceived as an additional and often overly complex task. Already tested and stable code has to be parallelized without modification of existing code. We therefore present two approaches with the generic scientific simulation environment [1] which makes use of several parallelization techniques possible without altering the existing algorithms. The parallel STL [2] and multi-threading techniques are utilized. Orthogonally by using the inherently parallel functional programming paradigm for algorithm specification, while combining it with generic and object-oriented programming paradigms. The time needed to calculate the distribution of quantities of interest in a given simulation domain is mostly allocated to the assembly of an equation system and the subsequent solution of the matrix representing said equation system. How this time requirement is allocated depends on the complexity of the equations being assembled as well as on their numerical condition. Most related work 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 to unify these interfaces as well as a replacement for the traversal schemes of the C++ STL.

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. The increasing complexity of underlying physical models often brings an increase of the complexity and amount of source code. Therefore, the efficiency of developing and maintaining source code is an increasingly important issue. It 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.

2 The Parallel GSSE

As has already been demonstrated [4, 5, 1], the goals of high runtime-performance and genericity do not have to be contradictory. Our approach deals with the identification and implementation of building blocks for an easy specification of all different 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 runtime performance. The most basic building block for the transition of continuous function spaces to discrete spaces is the operation of topological traversal. Several tasks in scientific computing can be greatly eased by the utilization of functional programming, but some tasks defy the nature of stateless description, e.g. loading a file. All different types of storage mechanisms as well as streaming processes cannot be easily described by functions. Instead our approach is based on 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 paradigm performs best to describe functional expressions, in our case discretized and linearized projections of the continuous function spaces.
- Finally, the generic programming paradigm couples the object-oriented and functional paradigm by, e.g., the parametric polymorphism of C++. In a more general way, generic programming excels at the abstract treatment of objects, called concepts.

To present our approach, the following source snippet presents a simple STL algorithm, of which a parallel version is already available as part of the parallel STL.

```
std::for_each(container.begin(), container.end(), functor);
```

The GSSE offers the same mechanism but with a full functional environment based on the Boost::Phoenix library. Where the STL offers support for various simple data-structures, the GSSE offers all different types of mesh and grid data-structures of all dimensions. The functional specification built on top of the topological interface is then dimensionally and topologically indepent. An example is given next, where geometrical points are selected by a coordinate functor:

```
traverse<segment>()
[
  traverse<vertex>()  [  if_(coord[0] > 5.3) [  quan = 1  ]   ]
](domain);
```

2.1 Assembly

The various discretization schemes differ with respect to which quantities they compute and in which fashion. The crucial difference concerning assembly, however, is the distinction of the required traversal operation. Actual access to quantities and insertion into the matrix itself remains identical. The assembly of the Jacobian matrix is the final crucial part, where the functional specification of the discretized equations is combined with the required traversal operations. The clean separation of these two steps guaranteed by our approach also provides a clear interface for parallelization. The traversal of the discretized simulation domain is responsible for the global assembly of the Jacobian matrix, whereas the assembly module takes care of mapping the local operations of each topological object to the global Jacobian matrix.

```
for (vertex_iterator vit = (iter_part).vertex_begin(threadID);
                    vit != (iter_part).vertex_end(threadID); ++vit)
      // assembly
```

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

3 Examples

3.1 Mesh Generation and Adaptation

An example of the parallel topological traversal mechanism is presented by 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 [6]. This guarantees a volume mesh generation approach, where each segment is meshed concurrently.

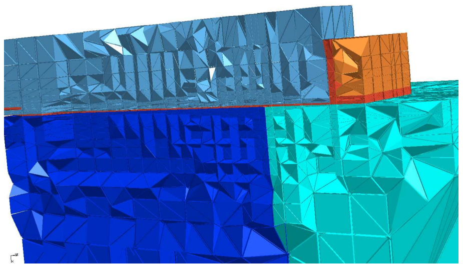

Fig. 1: A three-dimensional device structure for a MOSFET with an additional externally supplied point cloud. The important part is the regularity of the elements in the channel region (red).

Example	Sequential mesh	Dual-core	Quad-core	Num. points	Num. segments
MOSFET	69	34	12	3.6e5	7
Levelset	15	5	4	1.9e4	3

Table 1: Comparisons of the mesh generation and included mesh adaptation times (in seconds) on AMD's X2 5600.

3.2 Device Simulation

As another application we present an example of TCAD's device simulation applications. To present the parallel GSSE approach and also the functional specification mechanism the following code snippet demonstrates the actual C++ code for the electron temperature `n_te` for hydro-dynamic device simulation application. Equation 1 show the energy flux equation for electrons which is solved self-consistently with Poisson's equation and the current relations.

$$\mathrm{div}\left(-\tfrac{5}{2}\tfrac{k_B{}^2}{\mathrm{q}}\mathrm{grad}(nT_\mathrm{n}^2)+\mathrm{q}\;\mathrm{grad}\,\varphi\; n\; T_\mathrm{n}\right)=-\mathrm{grad}\,\varphi\cdot\mathbf{J}_\mathrm{n}-\frac{3}{2}\,k_B\,n\frac{T_\mathrm{n}-T_\mathrm{Lattice}}{\tau_n} \quad (1)$$

The following source snippet reflects the functional specification for a finite volume discretization scheme in actual C++ code:

```
(sum<edge>()
[ let(_x = Bern(edge_log<vertex>(equ_T_n)) / equ_T_n *-q/k_B *
      sum<vertex>() [ equ_pot ] + sum<vertex>() [ equ_T_n ]  )
   [
      equ_T_n / Bern(edge_log<vertex>(equ_T_n))  *
      sum<vertex>() [  equ_n *  equ_T_n * Bern( _x )  ] *
      5/2 * k_B * k_B / q  * n_mob_s  * area / dist
   ]
]  - sum<edge>()
[  sum<vertex>() [ equ_pot ] / dist * Jn
]  * vol  + 3/2 * k_B * equ_n * (equ_T_n - T_lattice )/tau_n*vol
)  (vx);
```

A benchmark for a simple drift-diffusion and hydro-dynamic simulation for a two-dimensional pn-diode with different compiler (gcc 4.2) optimizaton level and different numbers of concurrent threads is presented next.

Example	Sequential	Dual-core	Quad-core	Num. elements
DD, O1	34	12	8	1e4
DD, O3	12	10	8	1e4
HD, O1	44	21	9	1e4
HD, O3	22	12	8	1e4

Table 2: Comparisons of the simulation times for drift-diffusion and hydro-dynamic simulation of a pn-diode (seconds) with different optimization level (gcc 4.2) on AMD X2 5600 CPU's.

4 Conclusion

By using the concept of library-centric application design in the area of parallel environments, it can be shown that the whole simulation process can be easily separated into small building blocks. The appropriate realization of each of this blocks guarantees not only an impressive performance, but also eases development, stabilization, further support, and parallelization.

References

1. Heinzl, R., Schwaha, P., Selberherr, S.: A High Performance Generic Scientific Simulation Environment. In et al., B.K., ed.: Lecture Notes in Computer Science. Volume 4699/2007. Springer, Berlin / Heidelberg (2007) 781–790
2. Singler, J., Sanders, P., Putze, F.: The Multi-Core Standard Template Library. In: Lecture Notes in Computer Science. Volume 4641/2007. Springer, Berlin / Heidelberg (2007) 682–694
3. Kagstrom, B., Elmroth, E., Dongarra, J., (ed.), J.W.: Applied Parallel Computing. State of the Art in Scientific Computing. Berlin / Heidelberg (2007)
4. 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
5. Heinzl, R., Spevak, M., Schwaha, P., Grasser, T.: A High Performance Generic Scientific Simulation Environment. In: Proc. of the PARA Conf., Umea, Sweden (June 2006) 61
6. Stimpfl, F., Heinzl, R., Schwaha, P., Selberherr, S.: A Multi-Mode Mesh Generation Approach for Scientific Computing. In: ESM 2007, St. Julians, Malta (October 2007) 506–513