Towards a Free Open Source Process and Device Simulation Framework

J. Weinbub*, K. Rupp†*, L. Filipovic*, A. Makarov*, S. Selberherr*
*Institute for Microelectronics, TU Wien, Gußhausstraße 27-29, 1040 Wien, Austria
†Institute for Analysis and Scientific Computing, TU Wien, Wiedner Hauptstraße 8-10, 1040 Wien, Austria
E-mail: weinbub@iue.tuwien.ac.at

***Abstract*—We present an approach for implementing open source simulation tools in the field of semiconductor device and process simulation based on our execution framework ViennaX. We apply a modular concept, where functionality is separated into plugins, which in turn can be combined to form full-fledged simulation tools by utilizing ViennaX's task graph approach. Due to the applied plugin concept, a high degree of flexibility is introduced, as components can be easily exchanged. Simulation results are shown, depicting the applicability of our approach for different tools.**

I. Introduction

The field of semiconductor process and device simulation offers a plethora of publicly available simulation tools [1], [2]. However, only a fraction of the tools is available under a free open source license, also referred to as free software [3]. This impedes the progress of research in academia, as researchers are unable to access the code base of previously implemented software and extend it. In such a case, simulation tools must eventually by re-implemented. Clearly, this introduces additional development overhead which has a negative impact on the actual net research time. Therefore, the field of semiconductor process and device simulation in academia can greatly benefit from free open source software packages. These facts have already been established by various software developments, such as the Archimedes project [3]. Other fields, such as computational fluid dynamics, show that this concept works by providing multi-purpose simulation frameworks, similar to the COOLFluiD framework [4].

Another important aspect is to introduce a higher level of reusability in the simulation tools. A conventional example is the implementation of a deterministic simulator. Typically, such a tool is composed of core parts, e.g., an initial guess module, a Finite Element assembler, a linear solver, etc. A common task, is to exchange specific modules in order to investigate alternative approaches provided by different tools. For example, a different linear solver package might yield improved convergence behavior. This introduces the dire need for orthogonality in the simulator's code base, as switching of a module must not affect other components by, for example, introducing an altered interface which no longer fits to the remaining parts.

This work introduces an approach for setting up flexible simulations in the field of semiconductor device and process simulation. We utilize our plugin execution framework ViennaX [5] to not only reuse already available functionality but also to decouple simulations as much as possible (Figure 1). Our approach follows the task graph concept, since such functionality is seen as tasks represented as vertices of a graph [6]. Task dependencies are modeled as edges in the graph, ultimately defining a task graph which is used to drive the overall execution of the task flow. Due to the modular concept of ViennaX, the actual simulations are ultimately described with a set of configuration data. This data not only contains the required plugins, but also additional plugin-specific parameters, which can be accessed by the respective plugins. The framework is coded in the C++ programming language and relies on already available functionality, such as the Boost libraries [7]. Our focus is on supporting the dynamics within an academic environment, primarily imposed by an inherent flow of scientific personnel. This refers to the typical scenario of researchers' temporary employment within academic institutions. Therefore, we focus on extendibility, maintainability, flexibility, and, in addition, high-performance. Where the first three properties are fundamental to reduce the required development time of simulation tools, the latter aims for minimizing the actual execution time of the simulation runs. In the following sections the core aspects of our approach for future simulation flows in the field of semiconductor device and process simulation are discussed.

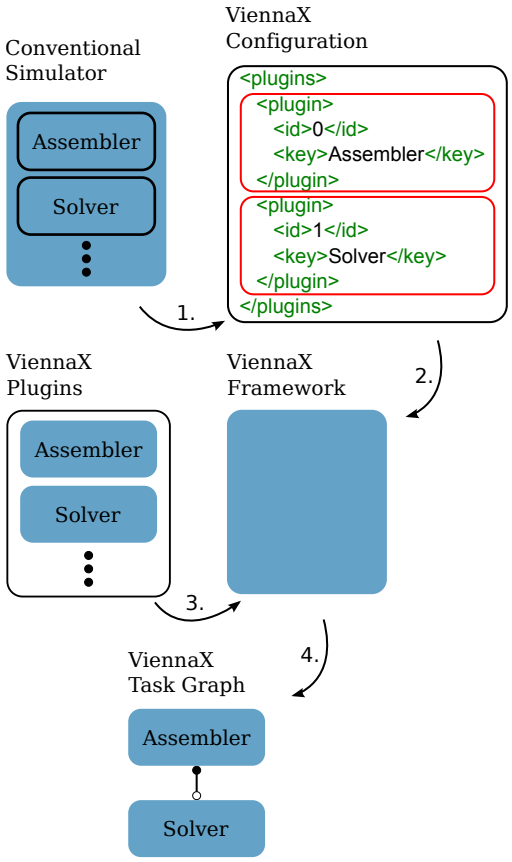

Fig. 1: The proposed decoupling of simulation tools is depicted by utilizing ViennaX. Step 1: A conventional simulation tool is described by a configuration file. Step 2: The configuration file is loaded. Step 3: The required plugins are loaded, according to the configuration data. Step 4: The task graph is generated based on the dependencies, and ultimately executed. The final simulation flow follows the initial one, however, the modularity, hence the flexibility, is increased substantially, as plugins can be easily exchanged.

II. Related Work

The Intel Threading Building Blocks (Intel TBB) library [8] is an Open Source library licensed under the GPLv2. The library provides mechanisms to express parallelism based on a shared-memory approach to C++ implementations. One of the core features is the so-called flow graph. A flow graph can be used to send messages, representing arbitrary data, between components. Our approach is similar to the one of Intel's TBB library. The primary difference, though, is the fact, that the TBB library is based solely on a shared-memory approach. Therefore, it does not scale beyond one computing node.

The COOLFluiD project [4], [9] enables multiphysics simulations based on a component framework. The framework is primarily designed for problems in the field of computational fluid dynamics (CFD). The core is a flexible plugin system, coupled with a data communication layer based on so-called data sockets. Each plugin can set up data sockets which are in turn used to generate a dependence hierarchy. This dependence information is used to drive the overall execution. The source code is available under the LGPLv3 license. The significance of the COOLFluiD framework with respect to our approach is twofold. First, we adopted the plugin system, enabling us to conveniently reuse already available functionality. Second, the communication layer based on data sockets is the basis for our implementation. However, our approach differs significantly. COOLFluiD performs an automatic partitioning and distribution of the data structures via the Message Passing Interface (MPI), whereas we follow the approach, that distribution should be performed on the user's intent within plugins, not automatically. In our opinion, COOLFluiD is entirely focused on the applications within the regime of CFD simulations. Our approach, however, enables a more general way to model processes, like the Intel TBB library, which ultimately supports utilization in a much broader field of scientific computing.

III. Our Approach

We focus on the decoupling of simulation flows according to their inherent functional blocks, as depicted in Figure 1. This modularization is implemented by utilizing our ViennaX framework, which provides a plugin system, configuration mechanisms, and execution schedulers. A serial and a parallel MPI-based scheduler are available. While the serial scheduler solely utilizes a single CPU core for executing tasks, the parallel MPI-based scheduler distributes plugins to different available cores, as long as the plugins can be executed concurrently (Figure 2). The plugin system is powered by a so-called self-registering technique [10]. This plugin approach introduces a high level of reusability by wrapping already available functionality into components with a specific, unified interface. The plugins can contain core parts of simulations, such as a linear solver implementation, but also full-fledged simulators in their own right. This approach is highly flexible; for example, simulation tools may be combined to form multiphysics simulation flows, but they may also be decomposed into smaller components, enabling specific exchanges of functionality by switching the respective plugins.

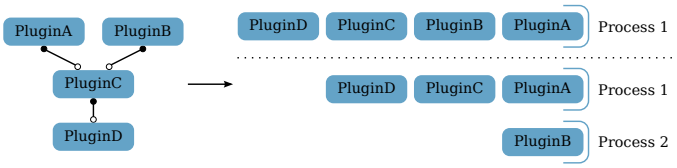

Fig. 2: The plugin execution is handled by the scheduler. If the plugins are parallelizable and there are free processes, the MPI-based scheduler distributes the task executions. Dots refer to outgoing data dependencies, and circles to incoming.

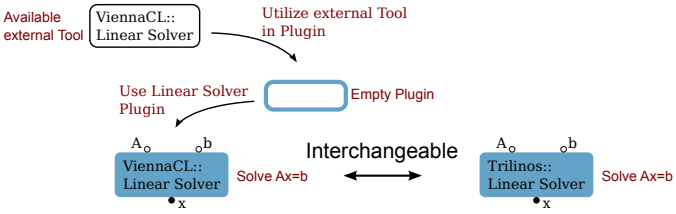

Fig. 3: A plugin can be used to wrap available functionality. Due to the abstraction mechanism provided by the socket input/output dependencies, plugins can be exchanged by other plugins.

Figure 3 depicts the set up and exchange of a plugin. If the process of interchanging plugins is compared to the one of conventional simulation tools, it is clear that the conventional approach would require actual coding, and as such requires in-depth knowledge of the implementation at hand. For obvious reasons, this fact impedes the implementation of changing functionality. With our plugin-based approach, the exchange can be realized conveniently, by adjusting the input configuration data accordingly.

A core part of a task graph execution framework is the communication layer. As already mentioned, we adopted the data socket approach of the COOLFluiD project [4]. Essentially, each plugin can define input and output data sockets, called sink and source, respectively. These data sockets can be used to send data to and from plugins. Sockets are defined before the execution of the overall task graph, and can be based on parameters provided by the input configuration data. Figure 4 depicts our approach for defining a plugin's data sockets. Arbitrary data types can be associated with each data socket, which are additionally identified with a unique identification string. Connecting the data sockets, which relates to generating the underlying task graph, is carried out automatically by the framework. The unique identification string as well as the associated data type of each data socket is used to automatically connect the corresponding counterparts.

To ascertain the validity of the input data, units of physical quantities can be attached to the identification string of the sockets. As the socket connection algorithm checks against this information, sockets with data of different units cannot be connected. This is an important aspect, as it automatically catches one of the fundamental error sources of scientific computing.

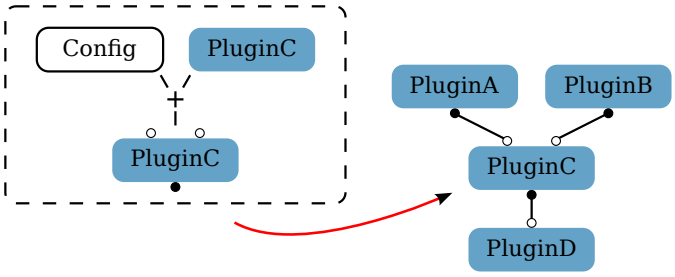

Fig. 4: The data sockets of a plugin are created based on the configuration during run-time. The sockets can be used to exchange data with other plugins.

A typical simulation task is implementing optimization processes. ViennaX enables this type of simulations by supporting loops in the task graphs, as depicted in Figure 5. This loop mechanism can be used to, for example, implement automatic mesh generation based on convergence to an optimal set of parameters driven by a specific metric, such as the quality of the generated mesh elements.

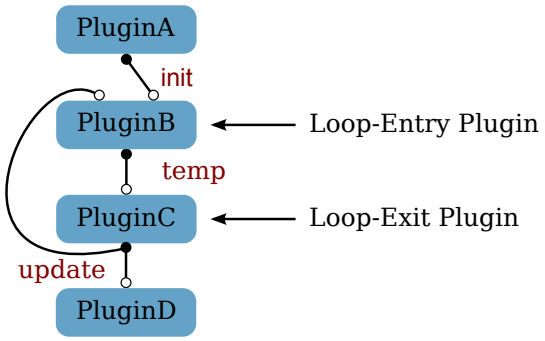

Fig. 5: An archetypal loop execution is depicted. PluginB and PluginC handle the loop logic. The loop is only exited, if the internal logic of the loop-exit plugin, PluginC in this case, decides so.

IV. Results

Some simulation results of three of our in-house developed simulation tools are shown, all of which have been executed as plugins in our framework.

Figure 6 depicts the evolution of a surface over time computed using our Level Set simulation tool [11]. Figure 7 shows the electron distribution in a symmetrically sliced, active FinFET device, evaluated using our spherical harmonics expansions (SHE) based Boltzmann equation (BE) solver ViennaSHE [12], [13]. Figure 8 outlines the magnetization vector field of a switching process for a penta-layer MTJ STT-RAM computed using our micromagnetic simulation tool [14].

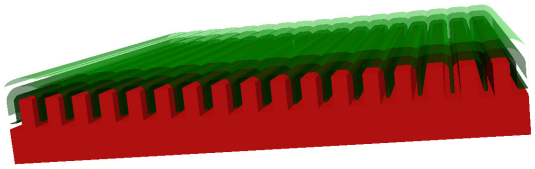

Fig. 6: A result of our Level Set simulation plugin is shown. The green surfaces denote the evolution of the surface position over time starting from an initial surface (red).

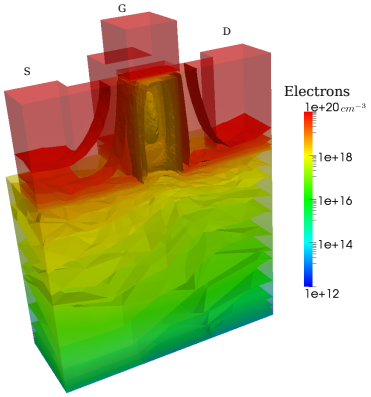

Fig. 7: An electron distribution based on a sliced, active FinFET device is shown, which has been computed using our SHE-based BE solver.

V. Summary

We have presented our approach for decoupling simulation flows in the field of semiconductor device and process simulation. The core components of ViennaX have been introduced in detail. Our approach for an open source simulation framework has proven to be efficient in integrating heterogenous simulation packages with minimal effort.

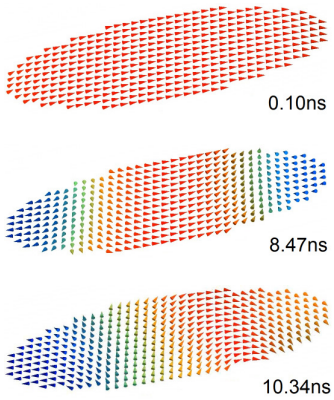

Fig. 8: The magnetization of a penta-layer STT-RAM at different time steps is shown, computed using our micromagnetic simulation tool. The colors indicate the x-component of the magnetization.

Acknowledgment

This work has been supported by the European Research Council through the grant #247056 MOSILSPIN. Karl Rupp acknowledges support by the Austrian Science Fund (FWF), grant P23598.

References

[1] "nanoHUB." [Online]. Available: http://nanohub.org/
[2] "tiberCAD." [Online]. Available: http://www.tibercad.org/
[3] "Archimedes." [Online]. Available: http://www.gnu.org/software/archimedes/
[4] "COOLFluiD." [Online]. Available: http://coolfluid.github.com/
[5] "ViennaX." [Online]. Available: http://viennax.sourceforge.net/
[6] A. Miller, "The Task Graph Pattern," in *Workshop on Parallel Programming Patterns (ParaPLoP)*, 2010, pp. 8:1–8:7.
[7] "Boost." [Online]. Available: http://www.boost.org/
[8] "The Intel Threading Building Blocks." [Online]. Available: http://threadingbuildingblocks.org/
[9] T. Quintino, "A Component Environment for High-Performance Scientific Computing," Ph.D. thesis, Katholieke Universiteit Leuven, 2008.
[10] D. Kharrat and S. Quadri, "Self-Registering Plug-ins: An Architecture for Extensible Software," in *Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2005, pp. 1324 –1327.
[11] L. Filipovic, O. Ertl, and S. Selberherr, "Parallelization Strategy for Hierarchical Run Length Encoded Data Structures," in *IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN)*, 2011, pp. 131–138.
[12] K. Rupp, T. Grasser, and A. Jüngel:, "On the Feasibility of Spherical Harmonics Expansions of the Boltzmann Transport Equation for Three-Dimensional Device Geometries," in *International Electron Devices Meeting (IEDM)*, 2011.
[13] "ViennaSHE." [Online]. Available: http://viennashe.sourceforge.net/
[14] A. Makarov, V. Sverdlov, D. Osintsev, and S. Selberherr:, "Reduction of Switching Time in Pentalayer Magnetic Tunnel Junctions with a Composite-Free Layer," *Physica Status Solidi - Rapid Research Letters*, vol. 5, no. 12, 2011.