State of the Art on: High Performance Modelica Compilers

MARINA NIKOLIC, marina.nikolic@mail.polimi.it

1. Introduction to the research topic

Our research topic belongs to the area of Programming Language Theory. Programming Language Theory is the branch of Theoretical Computer Science that covers the design, implementation and analysis of programming languages. The topic of the research involves both the sub-fields Compiler Construction and Modelling Languages.

We are focusing on the Modelica language, that is a modelling language used for the description and simulation on physical systems. Hence the literature on this topic can be found in both Computer Science and Control and System Engineering journals.

Among the most relevant journals related to the topic, there are:

- ACM Transactions on Computer Systems, ISSN 07342071, h-index 63.
- ACM Transactions on Programming Languages and Systems, ISSN 01640925, h-index 63.
- ACM Transactions on Architecture and Code Optimization, ISSN 15443973(online) and 15443566(printed), h-index 32.
- Software - Practice and Experience, published by John Wiley & Sons Inc., ISSN 1097024X (online) and 00380644 (printed), h-index 63.
- SoftwareX, published by Elsevier BV, ISSN 23527110, h-index 10. Despite the low h-index, this journal in in the first quartile of journals concerning Computer Science Application or Software.
- Advances in Engineering Software, published by Elsevier Ltd., ISSN 09659978, h-index 66.
- Simulation Modeling and Practice, by Elsevier BV, ISSN 1569190X, h-index 58.
- Simulation, published by SAGE Publications, ISSN 00375497, h-index 45.
- IEEE Transaction on Industrial Informatics, ISSN 15513203, h-index 100. This journal is in the first quartile for both Computer Science Applications and Control and System Engineering.

Programming Language research works is often presented at conferences, such as the International Symposium on Code Generation and Optimization (CGO), and the International Conference on Compiler Construction (CC).

The most relevant compiler infrastructure used in research is LLVM. There are two conferences organised by the LLVM community: the LLVM Developers’ Meetings, held twice a year (once in the USA and once in Europe).

As for modelling and simulation, it is worth mentioning three venues: the EOOLT workshop, which addresses the current state-of-the-art of equation-based object-oriented modelling languages; the Modelica Conference (the last one was held in March 2019 and was the 13th of the series, where has been presented a paper about a benchmark suite to test the performances of Modelica compilers compared to hand-written code); the IEEE International Conference on Industrial Informatics (INDIN).

---

1www.modelica.org
2Association for Computing Machinery
3Institute of Electrical and Electronics Engineers
1.1. Preliminaries

Modelica is a domain specific language used for simulation of physical systems. It is a descriptive language, i.e. it does not provide instructions to be executed in order to obtain a result. Instead, it is used to describe certain characteristics of a system (through equations). It is the compiler’s the task to evaluate the model and to create the code that runs the system simulation.

The simulation code is supposed to run and to produce the results once and for all. The change of some data (such as input or some parameters) leads to the need to re-evaluate the model and to create new code for the simulation. Thus, the re-usability of a compiled model is really low. Due to this aspect, the compile time becomes as important as the simulation time. This means that the evaluation of the performances of the compilers already present on the market or under development is indeed a research topic itself.

The tools on the market simulate the models described with Modelica language. The output of such simulations is usually a multi-dimensional array (where one of the dimension is time).

There are both commercial and open source tools. Among the commercial, the most used are AMESim from the company Imagine SA, Dymola from Dynasim AB, CyModelica from CyDesign Labs, Wolfram SystemModeler from Wolfram MathCore AB, SimulationX from ESI ITI GmbH, MapleSim from Maplesoft, CATIA Systems from Dassault Systemes. Open source tools available are OpenModelica and JModelica.

1.2. Research topic

Research on this topic focuses on the compilation of physical system models.

The processing of a Modelica file has several step.

- **Resolution** of the system of DAE\(^4\). There are several solvers on the market that implement algorithms for different types of system. Some examples of solvers are DASSL and IDA. The model is firstly evaluated and then transformed into an intermediary representation that can be processed further.

- **Transformation** of the equations (which are descriptive and do not impose an ordering among themselves) into a sequence of instructions that will produce the simulation (i.e. the code of an algorithm).

- **Simulation** of the model and retrieval, analysis and plotting of the results.

The main challenge for the compiler is the transition from descriptive to prescriptive behaviour, that does not happen with other languages.

According to the current Modelica specification, before the manipulation of the equations, the model has to be flattened\(^5\). This operation involves the scalarization of arrays and the lowering of hierarchy-based structures. The main consequence of this operation is the loss of the object-oriented structure of the model.

Improving the performances of Modelica compilers and simulation engines would impact on other research areas that depend on this modelling language. Modelica is nowadays used by many automotive companies\(^5\) to design energy efficient vehicles, and by power plant providers\(^6\). Modelica is designed to be domain neutral. It can be used in many other fields, such as fluid and thermo-fluid systems, mechanical systems (e.g. mechatronics and multi-body systems) and is particularly suited for system-level simulation (i.e. simulation of large cyber-physical systems, sometimes not fully specified).

We identified a set of issues that are currently blocking the adoption of these techniques to large scale physical models. These issues lie in the complexity of the current methodologies to simulate the models. We recall that time-to-solution is composed by the sum of compilation time and simulation time.

Another possible bottleneck is space complexity: the simulation of large models with a few kinds of component replicated causes to really large binary files, which are difficult even just to be stored\(^8\). Improvements in the compilation techniques would allow to simulate even larger systems than those that are supported now.

---

\(^{4}\)Differential and Algebraic Equations  
\(^{5}\)Automotive industries that use Modelica are Audi, BMW, Daimler, Ford, Toyota, VW and others, according to www.modelica.org  
\(^{6}\)ABB, EDF, Siemens and other power plant providers.
2. Main related works

2.1. Classification of the main related works

A first classification can be applied with respect to the nature of the projects. The literature on Modelica can either focus on particular use case, on the algorithmic part, on the development of tools for simulations, or on the evaluation of such tools.

Another dimension that can be used to classify the works is their coverage of the language. From this point of view, a rough distinction is between production grade tools and partial projects.

2.2. Brief description of the main related works

2.2.1 Production grade tools

There are several commercial tools on the market but, since they have proprietary licenses, their code and algorithms are not disclosed. The most relevant tool is Dymola, as it represents the evolution of the first Modelica definition and implementation.

OpenModelica\(^7\) is the most famous and significant open source work related to Modelica simulation. It is maintained by the OSMC\(^8\) and aims at providing long-term support to modeling and simulation environments. The research works that the OSMC is carrying out are in language design, control system design, symbolic and numerical algorithms and others. One research activity that is concerned about compilers is multi-core parallel code generation [11]. Even though OpenModelica is not the most performing tool on the market, it represents the most relevant open source effort in the Modelica community so far.

2.2.2 Partial works

Julia\(^9\) is an object-oriented programming language designed for high performance. Modia\(^10\) is an extension of the Julia language for modeling and simulation of physical systems and it is presented in [5] and [6]. Julia is compiled via LLVM, which allows to generate efficient native code. However, it uses just-in-time compilation techniques which are not always good for performances. In fact, the time to load a third-party package can become significant.

The main purpose of Modia is to offer a playground for testing new features and fast prototyping. The models are described by equations, just like in Modelica, and it implements an object-oriented approach. The current release still covers only partially the Modelica language, including some important missing features to be implemented, like arrays of components. Another limit of Modia with respect to other Modelica tools is the limited support of solver for the system of (differential or algebraic) equations, which cannot be decided by the programmer. Modelica tools typically offer a selection of solver libraries and techniques, while Modia uses only the Sundials IDA solver.

The Julia language has, however, an API that permits to connect Julia scripts to OpenModelica (OMJulia [10]). OMJulia uses the existing features of OpenModelica, so it presents the same performances issues as OpenModelica.

Previous work from our research group [1] shows an idea to rewrite from scratch a Modelica compiler with performances as the main goal. The new compiler would use the LLVM [9] compiler framework and transform the Modelica code into an intermediate representation that retains the structural information from the original code, allowing the generation of machine code that is optimised for the target hardware architecture (instead of generating an intermediate C code as OpenModelica does, allowing faster compilation). The main goal of the research is to preserve arrays, loops and, in general, the object-oriented structure of the original model. To this end, the idea is to use an object for the function required by the solver to run the simulation. This function is

\(^7\)www.openmodelica.org
\(^8\)Open Source Modelica Consortium
\(^9\)www.julialang.org
\(^10\)www.github.com/ModiaSim/Modia.jl
called residuals of DAE, and it has to be generated only once and then called using different inputs and outputs each time it is needed.

The results of previous work \[3\] show that the compilation times of the tools grows exponentially with the number of equations, while the hand-written code has nearly constant compile time. As for the time-to-solution, it is shown that the gain of the C++ code grows exponentially. Finally, the space complexity of the model is shown to be constant for the C++ code and linear for the tools. The results of the prototype code are due to the absence of the flattening phase with results in a model which is smaller in size (as it maintains its hierarchical structure).

Another interesting issue \[3\] is the approach used to map equations into C functions. The result is a large code size and high overhead due to function calls. Together with the loop unrolling used in the existing compilers, this causes the presence of unnecessarily high code duplication.

2.2.3 Benchmark suites

Most Modelica benchmark suites are not designed to tackle the problem of how performances scale with the size of the model.

A first example of benchmark for large models is Modelimark \[7\]. The performances are tested using Python scripting, to see how different tools scale with the size of the models. The compilers tested are OpenModelica, JModelica and Dymola. The results of the simulation show that, usually, Dymola performs better than the open source tools, but the time complexity is almost the same \[11\].

A scalable test suite has been implemented as a library \[4\] to test the solver performance (though, it can be used to evaluate compiler performance as well). It offers scalable tests whose size can be set as a parameter. It includes a set of physically meaningful models that represent some classes of real-life problems, and it is available online \[12\].

Another plan to create a benchmark suite is described in \[2\]. The suite is named HiPerMod and aims at creating a suite where the performances of present and future tools can be compared with hand-written C++ code. The C++ code is optimised by hand, thus can be considered an ideal target for compiler performances. Each benchmark will be independent from the others and the optimised code will be helpful for the design of high-performance compilers in the future. The full suite will cover different model structures and different solvers, in order to be used as a reference for future development of Modelica tools. Then, there are the results of test run on a simple model. It seems that the C++ code tends to be one order of magnitude faster.

2.3. Discussion

The benchmarks evaluate both time and space complexity, since the size of a file to be compiled affects the compile time as well. Large files, indeed, need large capabilities to be stored and thus make it impossible to compile models that are too large in size. Reducing the space needed to store models is surely one direction to improve performances.

The main reason for performance degradation is flattening, which results in a model that has lost its object-oriented structure. These models tend to need a large amount of space to be stored and processed. Since a lot of large models are in fact composed of arrays of smaller homogeneous components, flattening causes the loss of any SIMD optimisation that may be done thanks to the repetition of components with basically the same behaviour. The high space complexity, then, causes the compilation time to grow with the size of the model up to a point where the process fails.

It is common belief within the Modelica community that any substantial improvement to be done to Modelica compilers has to take into account flattening. Recent ideas \[11\] \[8\] consider to avoid flattening or, at least, to postpone it as much as possible in order to use the structural information for new optimisations.

Besides making files smaller in size, preserving the structure of models may lead to new and potentially better optimisations. For example, the use of equations involving arrays is quite frequent when simulating large and

\[11\] Polynomial in the number of equations
\[12\] github.com/casella/ScalableTestSuite
very large models, but it needs new data structures to be fully exploited. Interesting works about how to extend the existing algorithms to work with arrays have been presented recently [12].

REFERENCES


