Towards Model Simulations in HPC: a Compiler Perspective

Massimo Fioravanti† Marina Nikolic†
Silvano Seva†, Daniele Cattaneo†,
Stefano Cherubin*, Federico Terraneo†,
Francesco Casella†, Alberto Leva†,
Giovanni Agosta†

† *DEIB, Politecnico di Milano, piazza Leonardo Da Vinci 32, 20133 Milano, Italy* [1]
* *Codeplay Software Limited, Argyle House Level C, EH3 9DR Edinburgh, UK*

ABSTRACT

Modelica is a modeling language for performing simulations in multiple engineering domains, which has seen increasing interest in the last decades. However, existing compilers for this language are not performant enough with respect to both temporal and spatial efficiency to deliver on the promise of easy enough simulation of modern complex and/or large scale systems. Last year, we presented a new approach to high-performance system simulation, whose cornerstone is a new Modelica compiler named MARCO. The basic groundwork for the construction of this compiler is now complete, and now we are shifting our focus on the formalization of its main features, and on incrementally expanding its capabilities. We present the challenges we encountered, and how they related to the peculiarities of the Modelica language with respect to standard procedural languages such as C.

KEYWORDS: Compilers; LLVM; Modelica; Large-Scale Models; Efficient Simulation; DSL

Introduction

Modelica [Elm78] is a Domain Specific Language (DSL) tailored for the description and simulation of cyber-physical systems composed of both digital parts, described as imperative programs, and analog parts, described by systems of equations, including Differential Algebraic Equations (DAE) and Ordinary Differential Equations (ODE) [Fri14]. Modelica takes a declarative approach to the construction of the simulation, allowing the designer to specify and compose systems of equations, and then generate an executable code linking to DAE or ODE solver libraries. In a Modelica compiler, the high-level Modelica code is lowered to a flat structure in order to resolve the complexity given by component composition. The

[1]E-mail: {massimo.fioravanti, marina.nikolic}@mail.polimi.it – {silvano.seva, daniele.cattaneo, francesco.casella, alberto.leva, federico.terraneo, giovanni.agosta}@polimi.it
[2]E-mail: stefano.cherubin@codeplay.com

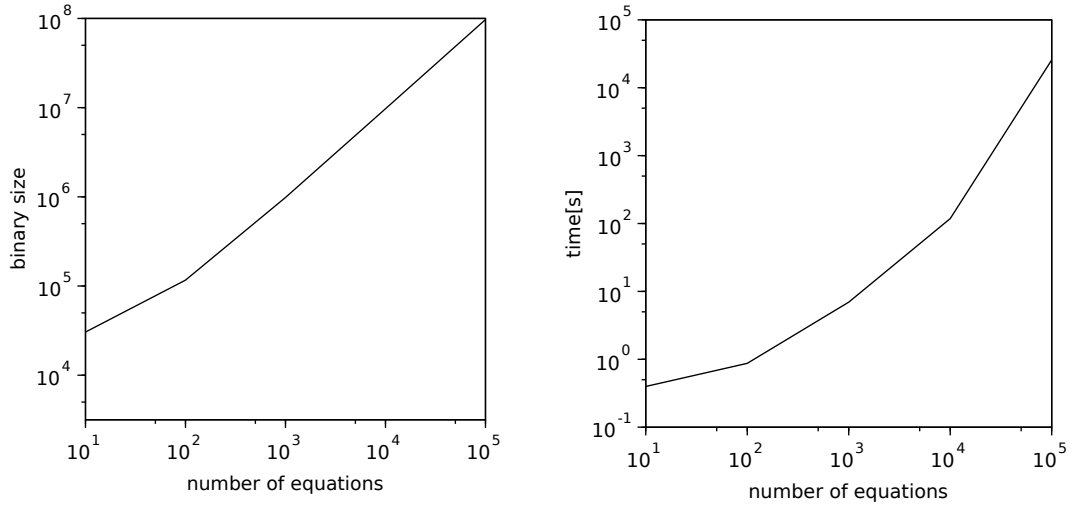

Figure 1: Code size (left) and time (right) gains that can be achieved via the LLVM-IR-based prototype Modelica compiler presented in [A+19].

lowered model, represented as an equation system, is then translated to C code and compiled using a standard C compiler. This approach is practical on one hand, because it frees the Modelica community from the need to develop a full compiler, but causes several limitations, which have been recently highlighted [A+19, Cas15, P+19]. Compilation times for large scale models may grow larger than the execution time. Additionally, in the generated C code large amount of information which was present in the Modelica source code is lost, and this lost information causes a performance degradation. It is now understood that by improving the code size and information processing of a Modelica compiler significant performance improvements are achievable. As an example, in Figure 1 we highlight the kind of improvements that can be achieved.

To address these limitations, we are working towards the construction of a new compiler for Modelica, named MARCO, based on the LLVM compiler infrastructure [LA04], which aims at progressively bridging the current gap.

Roadmap

Our development efforts for MARCO are following a roadmap consisting of three tasks. In the first task, we aim to develop a new backend for compiling the Modelica language to LLVM-IR. The other two tasks are planned simultaneously to the first, and involve the development of front-end related features. Thus, the second task involves the development of a new Modelica parser and frontend, and the third task the modularization of the algorithms developed for structural analysis.

From the point of view of a compiler engineer, all the aforementioned components could be classified as part of the frontend. However, this frontend is much more complex than the frontend of a standard programming language such as C or C++, because Modelica is a declarative language. For this reason, depending on the form of the equations in the model there are several possible solving methods. All of these methods are valid, but only a few are fast when executed by a computer.

Figure 2: The pipeline stages used by a typical Modelica compiler frontend [CK06].

MARCO aims to dramatically shift the state of the art in Modelica compilers by exploiting more complex transformations than existing implementations. These new transformation aim to preserve data locality and reduce code size by exploiting information already present in the source code, which is usually discarded.

For example an user can represent a relationship between the items in two different vectors. This can be expressed as a single equation subject to a *for-all* clause to describe the iteration across all elements of the vector. This equation is called a *vector equation*. However, a traditional Modelica compiler transforms every vector to scalar variables, and the single vector equation is lowered to as many scalar equations as there are elements in the vector. MARCO, instead, preserves the concept of vector in its internal representation, allowing more efficient allocation in memory of the elements of the vector, and enabling the usage of loops in the generated code to improve code size.

We have now reached a mid-point stage where an initial build of MARCO can compile simplified flattened Modelica models. Therefore, we now consider MARCO to have reached an *alpha* stage of development. Additionally, we have developed the beginnings of a modular Modelica frontend, which can be easily extended to support additional higher-level features of the language.

Technical Challenges

As a consequence of our continued development of the MARCO compiler, we uncovered several challenges that were not considered before. Indeed, the requirements for the algorithms employed in each stage of the compiler pipeline – which is shown in Figure 2 – are much stricter than what is currently found in the state of the art. Therefore, we have developed new algorithms for use in these stages, which must be formalized and studied to prove their correctness and computational complexity.

The first algorithm we consider, *matching*, determines which equation is used to evaluate each variable. In other words, this step transforms each equation in an expression statement that can be executed. The matching algorithm used by MARCO ensures that variables generated by expansion of the same vector are all evaluated by equations with the same structure.

The second algorithm is the *Strongly Connected Component Search*. This algorithm searches circular dependencies between equations, and if any are found, it defers solving to an external library. Often, vectors in the model result in well-known patterns in the graph of dependencies of the equations. Therefore, in MARCO we can exploit this information – which traditionally was not available to this stage.

Finally, we must adapt the *scheduling* algorithm, which decides the order in which the equations are evaluated and the values are assigned to the variables. MARCO ensure that, during scheduling, all variables expanded from the same vector are evaluated sequentially.

Future Developments

Our team is commited to further improve MARCO by completing the development of the new front-end, and by introducing an interface for external equation solving libraries. Such interface will be proposed as an open standard.

Additionally, to assess the effectiveness of MARCO, we will employ a new Modelica benchmark suite, HIPERMOD [ACC+19], which is currently under development. It contains scalable models specifically crafted to stress the compiler, and especially its capability to address large scale models created by the composition of relatively few basic components. This is a a common case in large scale Modelica models, which typically represent systems such as power grids or thermal properties of buildings.

Long-term goals include the investigation of parallel computing paradigms as well as other dynamic compilation techniques [C+20].

References

[A+19] Giovanni Agosta et al. Towards a high-performance modelica compiler. In *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*, number 157. Linköping University Electronic Press, 2019.

[ACC+19] Giovanni Agosta, Francesco Casella, Stefano Cherubin, Alberto Leva, and Federico Terraneo. Towards a benchmark suite for high-performance modelica compilers. In *International Workshop on Equation-Based Object-Oriented Modeling Language and Tools (EOOLT)*, pages 1–4, 2019.

[C+20] Stefano Cherubin et al. Dynamic precision autotuning with TAFFO. *ACM Transaction on Architecture and Code Optimization*, 17(2), may 2020.

[Cas15] Francesco Casella. Simulation of large-scale models in modelica: State of the art and future perspectives. In *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*, volume 0, pages 459–468. Linköping University Electronic Press, Linköpings universitet, 2015.

[CK06] François E Cellier and Ernesto Kofman. *Continuous system simulation*. Springer Science & Business Media, 2006.

[Elm78] Hilding Elmqvist. *A Structured Model Language for Large Continuous Systems*. PhD thesis, 1978.

[Fri14] Peter Fritzson. *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons, 2014.

[LA04] Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pages 75–86. IEEE, 2004.

[P+19] Adrian Pop et al. A new openmodelica compiler high performance frontend. In *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4-6, 2019*, number 157, page 10. Linköping University Electronic Press, Linköpings universitet, 2019.