Optimizing Simulink® Models

Bentley James Oakes[†], Clark Verbrugge[†], and Hans Vangheluwe[†‡]

[†]McGill University, Montreal, Canada [‡]University of Antwerp, Belgium

Problem Statement

The Simulink® modelling tool is used to diagram and study cyber-physical systems, and to generate embeddable code directly from the models. However, this code generation process means that inefficiencies in the model may be propagated to the code. Optimizations may be performed during code generation, but this may lead to an unacceptable loss of traceability in determining which parts of the model were modified or removed.

Our works focuses on defining model-to-model optimizations, where the optimized model can be loaded back into Simulink for further development or analysis. We suggest this improves traceability and can allow model specialization for different platforms. In this work, we present three optimizations to improve model simulation performance and/or the visual layout of the model.

Optimization Classification

In our work, we propose an optimization classification based upon the platform-dependence and intent of the optimization. This classification creates stronger theoretical connections to the compiler domain and the model transformation field, and assists us to discover and implement new optimizations.

Model-level

These optimizations are those that are not dependent in any way on the target architecture. As such, they focus on changing the structure of the model itself.

Examples:

- Algebraic simplification - Reduce computational effort
- Dead-block removal - Extraneous blocks should be removed
- Flattening - Remove subsystems from the model

Platform-independent

These optimizations are specific to a general class of target architecture, such as whether the target is single- or multi- core, or the target programming language of code generation.

Examples:

- Transforming floating point calculations into integer representations
- Specializing blocks into structures appropriate for a given target language.

Platform-dependent

Platform-dependent optimizations can be characterized by their dependence on a particular target architecture.

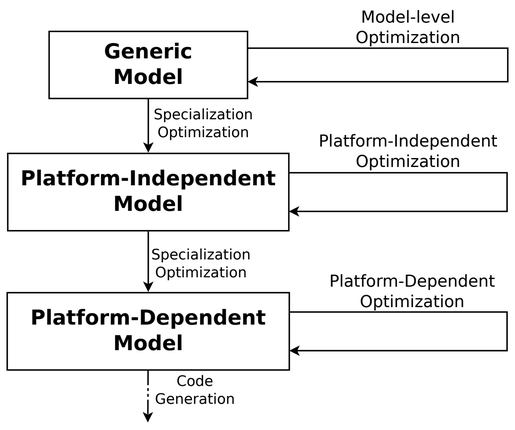
Examples:

- Re-order blocks to account for machine's caching strategy
- Schedule blocks among machine's processors

Optimization Hierarchy

These classification levels can be placed in a hierarchical relationship, with optimizations further down the hierarchy more specific to a particular machine.

A code synthesis workflow may include transformations between these levels, so that a general model is specialized further and further until code is generated from a platform-specific model.



Optimization Procedure

Analysis

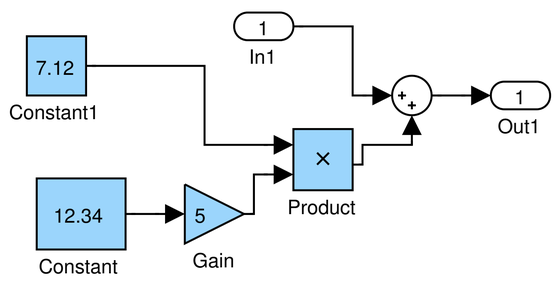
Our work defines an analysis procedure to collect dataflow information for each node in the Simulink model. This procedure is based upon that used in the compiler literature. For example, for constant folding, the information propagated is whether a block will always produce a constant value during simulation.

Transformation

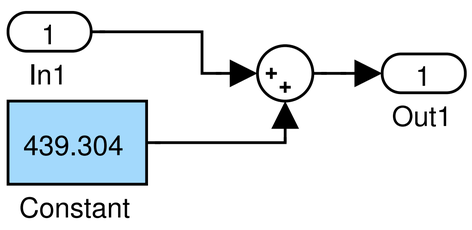
The optimization framework can then transform the model, based on the analysis results. This transformation is performed by utilizing the Himesis model format. After the transformation is complete, the optimized model is imported back into Simulink to be further developed [2].

Example Model-level Optimizations

Constant Folding



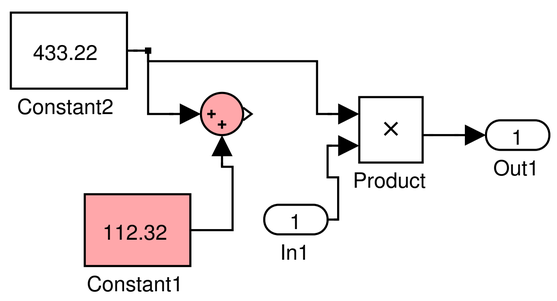
Model before



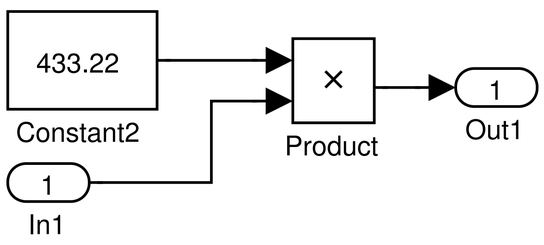
Model after

The constant folding optimization determines which blocks will always produce a constant value. For example, the four blocks on the left in the original model can be replaced by one constant block, reducing the computational effort required to simulate this model.

Dead-Block Removal



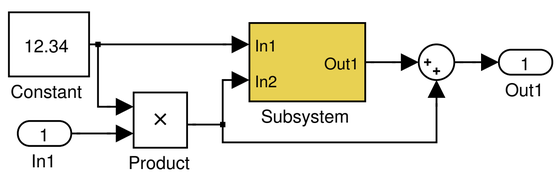
Model before



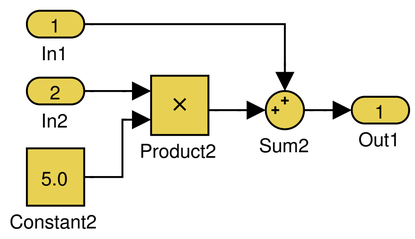
Model after

The dead-block removal optimization determines which blocks produce output that is not involved in further computation. For example, the output of the addition block on the left in the original model is not connected to another block. Thus the addition block and one constant block can be removed from the model.

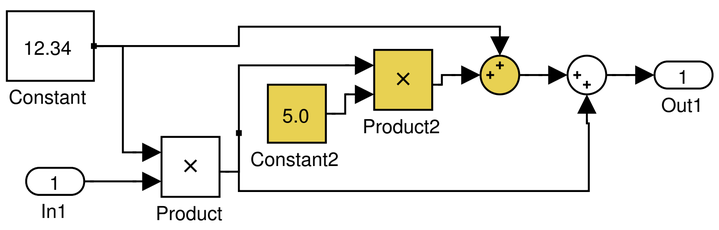
Flattening



Model before



Inner subsystem before



Flattened model after

The flattening optimization "lifts" blocks out of an inner subsystem of a model. Similarly to function inlining, this may allow other optimizations to simplify the model even further. As well, a flattening transformation may replace the flattening step performed during code generation, allowing the code generator to be less complex.

Experiments

Simulation timings before and after model optimization

Optimization	Before Opt. (sec.) Avg.	Before Opt. (sec.) Std. Dev.	After Opt. (sec.) Avg.	After Opt. (sec.) Std. Dev.
Constant Folding				
Model 1	**19.78**	.05	**16.95**	.21
Model 2	**18.35**	.07	**15.89**	.13
Model 3	**23.53**	.09	**20.77**	.09
Model 4	**18.01**	.09	**17.22**	.23
Dead-Block Removal				
Model 1	16.79	.27	16.91	.25
Flattening				
Model 1	18.87	.22	18.75	.19
Model 2	21.77	.16	21.75	.38
Model 3	19.54	.12	19.94	.06

Timings for each step in our framework

Experiment Step	Avg. Time (sec.)	Std. Dev.
Connect to Simulink	11.71	.24
Import from Simulink	4.94	.04
Create model in Python	.08	.01
Analysis	.02	.00
Transformation	.01	.00
Export to Simulink	.01	.01

Conclusions & Future Work

The optimizations shown here demonstrate how the performance of a model simulation can be increased (as for the constant folding optimization), and how the visual layout of a model can be improved (as with the dead-block removal and flattening optimizations). Our framework allows these optimizations to be specified and implemented easily, and for model optimization to be on the order of seconds.

Future work will focus on specifying more optimizations at all levels of the optimization hierarchy. As well, experiments will be performed on larger and more complex models. Finally, we aim to formally verify the model transformations used.

Acknowledgments

We would like to thank Joachim Denil (NECSIS), Bart Pussig (University of Antwerp), and Pieter Mosterman (The Mathworks) for their support.

Bibliography

[1] Bentley James Oakes, *Optimizing Simulink Models*, Report for COMP 621 - Program Analysis and Transformations, https://github.com/BentleyJOakes/BDOT

[2] Joachim Denil and Pieter J. Mosterman and Hans Vangheluwe, *Rule-Based Model Transformation For, and In Simulink*, Theory of Modeling and Simulation 2014 (to appear)

http://msdl.cs.mcgill.ca/ bentley.oakes@mail.mcgill.ca