# Optimizing Simulink® Diagrams

Bentley James Oakes

McGill University, Montreal, Canada

## Problem Statement

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

Our works focuses on defining model-to-model optimizations. This means that the optimized model can be loaded back into Simulink for further development or analysis, improving traceability and allowing model specialization for different platforms. An analysis framework has been created, based on dataflow analysis from the compiler optimization domain. This allows fast and accurate definition of new optimizations. As well, an initial optimization classification was developed to aid in the discovery of new optimizations.

## Optimization Classification

In our work, we propose a classification for causal-block model optimization, based upon the platform-dependence and intent of the optimization. We believe that defining classifications for optimizations will create stronger theoretical connections to the compiler domain, as well as to transformations from the model transformation field.

### Model-level

The model-level optimizations are those that are not dependent in any way on the target architecture, and focus on changing the structure itself of the model.

Examples include, dead-block removal where blocks that are not needed for calculation in the system should be removed, or algebraic simplification of models to reduce computational effort.

### Platform-independent

In this second level of optimization generality, 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.

Example optimizations include transforming floating point calculations into integer representations, and specializing blocks into structures appropriate for a given target language.

### Platform-dependent

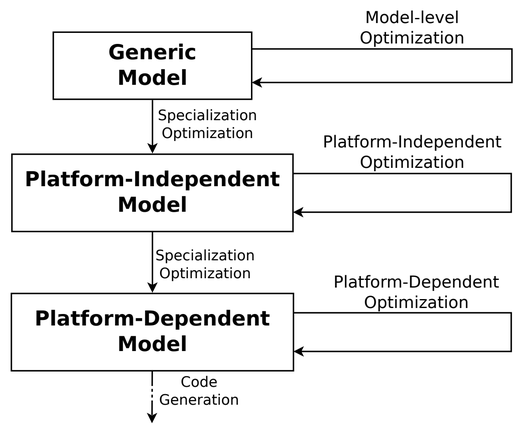
Platform-dependent optimizations can be characterized by their dependence on a particular target architecture.

Examples include rearranging the model in order to take advantage of a particular machine's caching strategy or hardware layout.

### 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

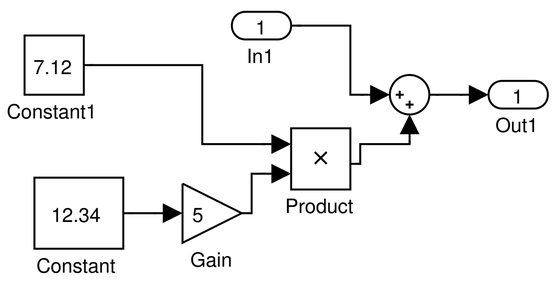
Our work defines an analysis procedure to collect dataflow information for each node in the Simulink model. This procedure is based upon that used in the compiler literature. For example, for constant folding, the information propagated is whether a block will always produce a constant value during simulation.

### Transformation

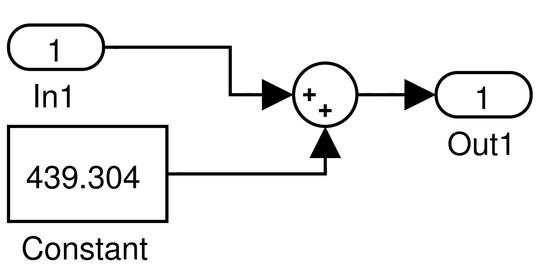
The optimization framework can then transform the model, based on the analysis results. This transformation is performed by utilizing the Himesis model format. After the transformation is complete, the optimized model is imported back into Simulink. This provides a second model to be further tested or modified.

## Example Optimizations

### Constant Folding



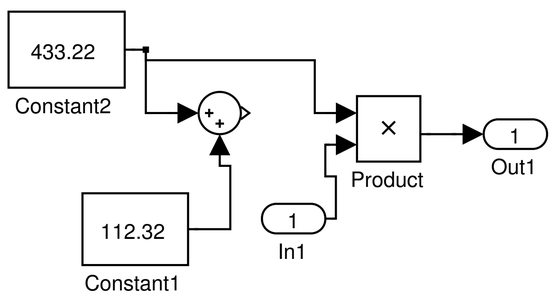

Model before



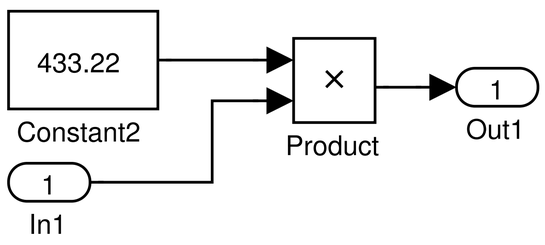

Model after

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. This reduces the computation effort required by this model.

### Dead-Block Removal



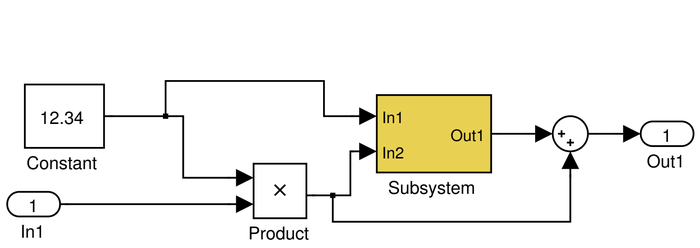

Model before



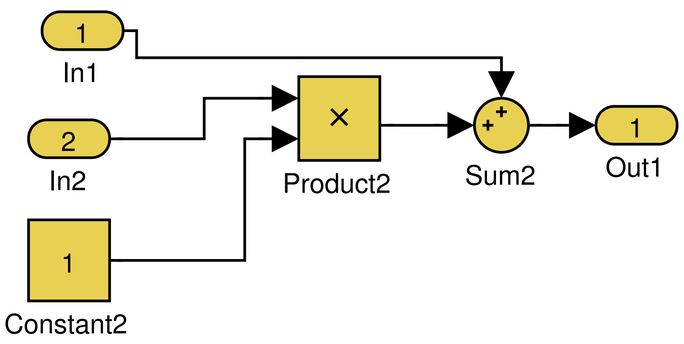

Model after

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. This reduces the computation effort required by this model.

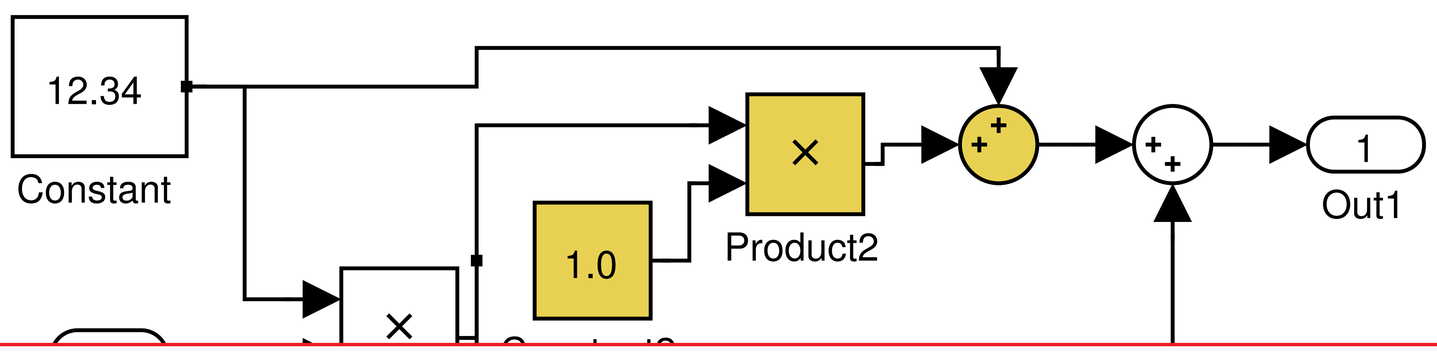
### Flattening




Blah blah blah




Blah blah blah




## Experiments

| 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 |

*Representative timings for performing an optimization*

| Optimization | Original Model (sec.) | | Transformed Model (sec.) | |
|---|---|---|---|---|
| | Avg. | Std. Dev. | Avg. | 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 |

*Simulation timings with and without optimizations*

## Conclusions & Future Work

The DSLTrans principles can be extended to building other domain specific translation languages. We are currently thinking about building domain specific translation languages by composing components such that the resulting language will always describe *finite* and *confluent* transformations. If that is the case, transformation model checkers such as the one for DSLTrans can also be (semi-)automatically built.

## Bibliography

[1] Bruno Barroca, Levi Lucio, Vasco Amaral, Roberto Félix and Vasco Sousa, *DSLTrans: A Turing Incomplete Transformation Language*, Proceedings of the SLE 2010 conference, Springer 2010, pp. 296-305.

[2] Levi Lucio, Bruno Barroca, Vasco Amaral, *A Technique for Automatic Validation of Model Transformations*, Proceedings of the MoDELS 2010 Conference, Springer, pp. 136-150.