Efficient IR for the OpenModelica Compiler

Contact: Martin Sjölund (martin.sjolund@liu.se), tel: 0707-567358
Peter Fritzson (peter.fritzson@liu.se, tel: 0708-281484
PELAB – Programming Environment Lab, Institutionen för Datavetenskap
www.openmodelica.org

PELAB, together with the Open Source Modelica Consortium (an international open source effort supported by 48 organizations, see www.openmodelica.org) develops OpenModelica, an open-source Modelica-based modeling and simulation environment intended for industrial and academic usage. OpenModelica, includes the OpenModelica Compiler (OMC) of the Modelica language, an equation-based and functional language aimed at modeling of cyber-physical systems. like cars, aircraft, power plants, wind power, etc.

An important goal for the OpenModelica compiler is to generate efficient code. However, one of the problems with the current Modelica/MetaModelica code generator is that it generates C-code directly from a high-level representation of functions and expressions. The generates code is far from optimal especially in the following cases:

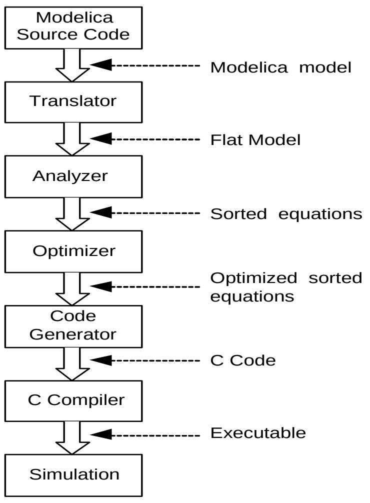
- Pattern matching (linear search instead of a state-machine)
- Register allocation / temporaries not fully minimized
- Array operations do not re-use memory

It is not possible to perform these operations on the current high-level representation of the code, which is why a better intermediate representation (IR) would be necessary. This representation needs to be able to handle common operations like allocating memory explicitly.

The master thesis project should develop a new IR for OpenModelica for more efficient code generation. While introducing a more efficient IR, it would also be advisable to make certain operations more efficient. This includes some optimizations such as dead code / use of uninitialized variables that are currently performed elsewhere in the compiler. One of the other gains would be to make interpretation of functions more efficient. Currently, even small programs like the following takes 13s to interpret:

```
function f
  input Real rs[:];  output Real sum;
algorithm
  for r in rs loop
    sum := sum+r;
  end for;
end f;
```

```
model M
  Real r=f(1:2000000);
end M;
```



The reason for the bad performance is that it performs a lookup of variable r in each iteration of the loop in order to figure out what the value is, and another to assign this value (the performance is a lot worse if there are more local variables in the function). The operation should just be a lookup in an array with a known index (and temporaries should do the same). If for-loops and other control structures should be kept in order to preserve the structure of the C-code is debatable. Fully lowering the IR makes it easier to map to for example LLVM IR, while making the generated code read more like assembly code.

It would be interesting to map this IR into LLVM bitcode in memory in order to investigate generating object code directly from OMC, although this could be left as future work.

OpenModelica deals with very large inputs at times, so bad scaling of functions is not acceptable. It can be assumed that there is a finite (but large) number of variables and statements in a function, but some inputs of functions should be assumed to be infinite.

The work is suitable for one or two students, and can be divided according to sub-task. Most of the development is in a functional language called MetaModelica, including support for pattern matching and AST transformations. It has Modelica-style syntax but functionality similar to functional languages like F# and OCAML. A short introductory course is available. Some run-time system development will be in C. The whole OpenModelica compiler is written in about 200 000 lines of MetaModelica. The compiler is bootstrapped, i.e., it compiles itself. The student should be interested in advanced programming and compiler construction. The following courses are especially useful to know:

1. Compiler Construction
2. Functional programming (knowledge of any functional programming language is an advantage)
3. Data structures and algorithms
4. Design and Analysis of Algorithms is not necessary, but knowledge of time complexity of algorithms helps a lot since performance is of importance for this thesis.