Code generation from an Esterel PDG

Cristian Soviani

October 30, 2002

Abstract

If a concise CFG exists for the given PDG, an optimal CFG can be efficiently generated. This is not the case for most Esterel programs. Solving the general problem optimally is NP-complete, so my project will find an non-optimal but efficient CFG.

1 Introduction

From the programmer's point of view, Esterel can be the ideal choice for writing a large class of embedded systems software. The real battle is performance. Speed and size are both crucial issues in embedded systems. If the Esterel compiler does not generate very efficient code, the programmer will have to switch back to C.

The standard Esterel V5 compiler[1] can generate both automata and netlist code. The first is very fast but code size can explode for other than small programs. The former keeps the code small but it is slow. These methods have a solid theoretical background (and they are excellent tools for program analyzing and debugging) but run-time performance requires a more pragmatic view of the problem.

In his EC compiler, Edwards [1] starts from the Berry's IC intermediary format to generate a CCFG (concurrent control flow graph) then a SCFG (sequential control flow graph) which can be easily translated into code. Also EC generates quite efficient code, there is still place for improvement.

So compiling Esterel efficiently is not at all a trivial.

My approach is to generate the CFG (controlflow graph) and thus the code from the PDG (program dependency graph). The PDG is an intermediary representation of the program, consisting from a CFG (control flow graph) and a DDG (data dependency graph). Instead of being a simple "translation" of the input program, like IC, the PDG describes the program at a higher level of abstraction, removing arbitrary links between the nodes and thus containing only mandatory flow and control dependency. Generating a CFG from a PDG is not a trivial problem but it is a promising way to get both short and fast code.

[1] www.esterel-technologies.com

2 Related Work

Simmons and Ferrante describe an efficient algorithm [2] for generating a CFG from a PDG, when a concise CFG exists (i.e. no additional predicates/ code duplication is required). Their ingenious algorithm reduces the problem to the ordering of each parent node's children (siblings) and runs in polynomial time (O(ne) n=nodes, e=edges).

The algorithm walks the CFG twice and computes for each node a "region" and a "eec". Also quickly computed, this information will if there are CFG constraints(such as external edges) in scheduling siblings. Sibling ordering is done by inspecting sibling's "eec" and data dependencies, using a set of ordering rules.

Also the algorithm stops when finding a concise CFG is not possible, it clearly points out where guard variables / code duplication are needed to solve the general problem.

Steensgaard extends Ferrante's work to handle irreductible programs which contain multiple entry loops [3]. This is done by introducing the notions of loop entry and close nodes. His work still considers only

PDGs for which a concise CFG exist.

Edwards attacks the problem from a different angle in his EC Esterel compiler [1]. He notices that the general problem is equivalent to scheduling the nodes in a topological order according to the PDG graph. He also shows the general problem is NP-complete (by proving it can be used to solve a known NP-complete problem).

Technically, the key is generating a SCFG from the CCFG. Also intuitive, it is rather difficult. It can intuitively seen as "thread interleaving". EC statically slices the threads and introduces additional variables to store the thread "state" at cut points. The variables are later used to "resume" the thread execution.

EC does not search for the "impossible to find" optimal solution but uses a depth first algorithm. Edwards' work is sustained by the experimental results which show a slightly increase in code size for a large class of real world Esterel inputs (even a malicious party can find an input to blow up the output size).

Fortunately, Esterel CFGs do not contain loops because of Esterel's timing concept. The program's CFG will run each tick. State variables maintain the "machine state" between runs. Each tick, the program will compute current outputs depending of the current inputs and the state variables.

My project will use Edwards' slicing technique, as well as Ferrante's sibling scheduling algorithm.

References

[1] Stephen A. Edwards. An Esterel compiler for large control-dominated systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(2):169–183, February 2002.

[2] Barbara Simons and Jeanne Ferrante. An efficient algorithm for constructing a control flow graph for parallel code. Technical Report TR–03.465, IBM, Santa Teresa Laboratory, San Jose, California, February 1993.

[3] Bjarne Steensgaard. Sequentializing program dependence graphs for irreducible programs. Technical Report MSR-TR-93-14, Microsoft, October 1993.