Adding Static Single Information Form to LLVM

Andre Luiz Camargos Tavares
andrelct@dcc.ufmg.br

27 de março de 2009

1 Objective

The objective of this work is to extend the intermediate representation of LLVM with capabilities to handle programs in Static Single Information form [1]. Once the intermediate representation of LLVM has been properly extended, we will use the new features to provide LLVM with a variable bitwidth analysis [9] and the ABCD algorithm for array bounds checking elimination [2].

The conversion for SSI form is an idea that some people find useless. I would like to have opinions on this matter to decide if I will convert to SSI, or only provide LLVM with a variable bitwidth analysis and the ABCD algorithm for array bounds checking elimination.

2 Motivation

The LLVM compiler uses an intermediate representation strongly based on the Static Single Assignment (SSA) form [5], which has the property that each variable is used at most once. The Static Single Information (SSI) form [1] is an extension of SSA form which has the additional property that each definition of a variable will be used only along one path of the control flow graph of the program.

The SSI representation is a central part of many different analysis and optimizations that are used in the back end of compilers. It is used, for instance, to implement bounds checking elimination of arrays [2] and other memory regions [8]. It is also used to implement bitwidth analysis, which maps each variable in the source code to an approximation of its size, in bits [9]. Recently it was showed that the interference graphs of programs in SSI form are *interval graphs*, what simplifies register allocation [3].

3 Methodology

We plan to extend the intermediate representation of LLVM with a new instruction, that we call σ node [1]. Additionally, we will add to LLVM one pass that converts a program from SSA form to SSI form, using the algorithm described in [1]. In this way, the new representation will be used only when necessary, and programs compiled using the original LLVM IR will not need to be recompiled.

The σ-nodes, or σ-functions, are the dual of ϕ-functions. Whereas the latter has the functionality of a variable multiplexer, the former is analogous to a demultiplexer, that performs a parallel assignment depending on the execution path taken. Consider, for instance, the assignment below:

$$[(v_{11}, \ldots, v_{n1}) : L_1, \ldots (v_{1m}, \ldots, v_{nm}) : L_m] = \sigma(v_1, \ldots, v_n)$$

[1] This instruction is called π-node by Bodik *et al* [2], and *switch operator* in [6].

which represents m σ-nodes such as $(v_{i1} : L_1, \ldots, v_{im} : L_m) \leftarrow \sigma(v_i)$. This instruction has the effect of assigning to each variable $v_{ij} : L_j$ the value in v_i if control flows into block L_j. Notice that variables alive in different branches of a basic block are given different names by the π-function that ends that basic block.

To validate our additions to LLVM, we plan to implement two new passes for this compiler. The first pass will be the bitwidth analysis described by Stephenson *et al.* [9], which maps variables to a lower bound of their size in bits. This analysis is necessary to hardware synthesis, a domain on which LLVM has already been used [4]. We hope that our analysis will be able to help improve experiments like those performed by Cong *et al.* [4], who mapped variables to random widths, instead of using approximated lower bounds. The second pass that we plan to implement will be the ABCD algorithm for eliminating array bounds-checking [2]. We plan to use this new pass to verify if induction variables are being used inside correct boundaries.

4 Timeline

This is a three months project. The first month will be dedicated to extending the LLVM intermediate representation with the new σ instruction, and a pass to map SSA form into SSI form. The second month will be dedicated to the implementation of the variable bitwidth analysis, whereas the third month will be dedicated to the implementation of the ABCD algorithm.

5 Biography

I am currently a first year master student at the Federal University of Minas Gerais, under orientation of professor Roberto Bigonha, and co-orientation of Dr Fernando Pereira and professor Mariza Bigonha. Before joining the master program, I spent two years doing undergraduate research in two different lines areas: OSGi, with one published journal at the SEN 2008 [12]; Aspect Oriented Programs, with three published papers, SAC 2008 [13], WASP 2006 [10] and REIC 2007 [11].

I believe I will be able to successfully work in this project for three reasons. First, my co-adviser, Fernando Pereira, has a deep knowledge on LLVM. He participated as a Google Summer of Coder in 2006, developing a chordal based register allocator for LLVM [7]. Second, I will implement a register allocator based on coloring of chordal graphs, similar to [7], this semester, which will enrich my knowledge on LLVM. Third, I am really excited about working with LLVM.

Referências

[1] Scott Ananian. The static single information form. Master's thesis, MIT, September 1999.

[2] Rastislav Bodik, Rajiv Gupta, and Vivek Sarkar. ABCD: eliminating array bounds checks on demand. In *Conference on Programming Language Design and Implementation*, pages 321–333, 2000.

[3] Philip Brisk and Majid Sarrafzadeh. Interference graphs for procedures in static single information form are interval graphs. In *10th international workshop on Software and compilers for embedded systems*, pages 101–110. ACM Press, 2007.

[4] Jason Cong, Yiping Fan, Guoling Han, Yizhou Lin, Junjuan Xu, Zhiru Zhang, and Xu Cheng. Bitwidth-aware scheduling and binding in high-level synthesis. *Design Au-*

tomation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific, 2:856–861, 18-21 Jan. 2005.

[5] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *TOPLAS*, 13(4):451–490, 1991.

[6] Richard Johnson and Keshav Pingali. Dependence-based program analysis. In *Conference on Programming Language Design and Implementation*, pages 78–89, 1993.

[7] Fernando Magno Quintao Pereira and Jens Palsberg. Register allocation via coloring of chordal graphs. In *APLAS*, pages 315–329. Springer, 2005.

[8] Radu Rugina and Martin Rinard. Symbolic bounds analysis of pointers, array indices, and accessed memory regions. *SIGPLAN Not.*, 35(5):182–195, 2000.

[9] Mark Stephenson, Jonathan Babb, and Saman Amarasinghe. Bidwidth analysis with application to silicon compilation. In *PLDI '00: Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation*, pages 108–120, New York, NY, USA, 2000. ACM.

[10] Andre Luiz Camargos Tavares and Marco Tulio de Oliveira Valente. Execução remota de aplicações para computadores móveis usando aspectos. In *III Workshop Brasileiro de Desenvolvimento Orientado a Aspectos (WASP)*, pages 21–29, 2006.

[11] Andre Luiz Camargos Tavares and Marco Tulio de Oliveira Valente. Aspectos para execução remota de aplicações j2me. In *Revista Eletrônica de Iniciação Científica*, pages 1–10, 2007.

[12] Andre Luiz Camargos Tavares and Marco Tulio de Oliveira Valente. A gentle introduction to OSGi. *SIGSOFT Softw. Eng. Notes*, 33(5):1–5, 2008.

[13] Andre Luiz Camargos Tavares and Marco Tulio de Oliveira Valente. A remote display system for java-based mobile applications. In *SAC*, pages 1918–1922, 2008.