The Design and Implementation of the SELF Compiler, an Optimizing Compiler for Object-Oriented Programming Languages

A Dissertation
Submitted to the Department of Computer Science
and the Committee on Graduate Studies
of Stanford University
in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

Craig Chambers

March 13, 1992

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

David Ungar (Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

John Hennessy

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Mark Linton

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

Abstract

Object-oriented programming languages promise to improve programmer productivity by supporting *abstract data types, inheritance*, and *message passing* directly within the language. Unfortunately, traditional implementations of object-oriented language features, particularly message passing, have been much slower than traditional implementations of their non-object-oriented counterparts: the fastest existing implementation of Smalltalk-80 runs at only a tenth the speed of an optimizing C implementation. The dearth of suitable implementation technology has forced most object-oriented languages to be designed as hybrids with traditional non-object-oriented languages, complicating the languages and making programs harder to extend and reuse.

This dissertation describes a collection of implementation techniques that can improve the run-time performance of object-oriented languages, in hopes of reducing the need for hybrid languages and encouraging wider spread of purely object-oriented languages. The purpose of the new techniques is to identify those messages whose receiver can only be of a single representation and eliminate the overhead of message passing by replacing the message with a normal direct procedure call; these direct procedure calls are then amenable to traditional inline-expansion. The techniques include a *type analysis* component that analyzes the procedures being compiled and extracts representation-level type information about the receivers of messages. To enable more messages to be optimized away, the techniques include a number of transformations which can increase the number of messages with a single receiver type. *Customization* transforms a single source method into several compiled versions, each version specific to a particular inheriting receiver type; customization allows all messages to **`self`** to be inlined away (or at least replaced with direct procedure calls). To avoid generating too much compiled code, the compiler is invoked at run-time, generating customized versions only for those method/receiver type pairs used by a particular program. *Splitting* transforms a single path through a source method into multiple separate fragments of compiled code, each fragment specific to a particular combination of run-time types. Messages to expressions of these discriminated types can then be optimized away in the split versions. The techniques are designed to coexist with other requirements of the language and programming environment, such as generic arithmetic, user-defined control structures, robust error-checking language primitives, source-level debugging, and automatic recompilation of out-of-date methods after a programming change.

These techniques have been implemented as part of the compiler for the SELF language, a purely object-oriented language designed as a refinement of Smalltalk-80. If only pre-existing implementation technology were used for SELF, programs in SELF would run one to two orders of magnitude slower than their counterparts written in a traditional non-object-oriented language. However, by applying the techniques described in this dissertation, the performance of the SELF system is five times better than the fastest Smalltalk-80 system, better than that of an optimizing Scheme implementation, and close to half that of an optimizing C implementation.

These techniques could be applied to other object-oriented languages to boost their performance or enable a more object-oriented programming style. They also are applicable to non-object-oriented languages incorporating generic arithmetic or other generic operations, including Lisp, Icon, and APL. Finally, they might be applicable to languages that include multiple representations or states of a single program structure, such as logic variables in Prolog and futures in Multilisp.

Acknowledgments

My heartfelt appreciation and thanks go to my advisor, David Ungar, for providing me with the great opportunity to participate in the SELF project. David always treated me as a colleague, promoted my work to others, and did his best to prepare me for independent research. He taught me much about research, writing, speaking, advising, and cartooning. I could never have had such an enjoyable and educational graduate experience without him. I will strive to be as good to my students as he was to me.

Besides serving on my thesis reading committee and providing much important feedback, Mark Linton also provided advice and direction during my first year at Stanford. His weekly research group meetings throughout my years at Stanford were interesting and educational. I appreciate Mark's strong support of my work. John Hennessy also served on my reading committee; his perceptive comments on my dissertation significantly improved its presentation, and in the process taught me a great deal about good research.

I also thank the other members of the SELF team for stimulating discussions and fun outings. Thanks to Elgin Lee for sharing an office and the birth of the SELF system with me. Bay-Wei Chang and Urs Hölzle worked and played with me as the SELF project matured. These patient people put up with my not-so-occasional coffee-induced tirades and continued to listen and discuss after the caffeine wore off. I will always remember fondly the times we've shared. More recently, Randy Smith, Ole Agesen, John Maloney, and Lars Bak have kept the SELF group a fun and stimulating collection of people.

Others at Stanford provided moral support and social diversions that helped me through. Ross Finlayson, Steve "Brat" Goldberg, Paul Calder, and John Vlissides were particularly good friends. My brother-in-common-law, Martin Rinard, made my time at Stanford interesting, to say the least.

I appreciate the support and patience of my new colleagues at the University of Washington as I finished this dissertation concurrently with my other responsibilities. By helping to make my transition from student to professor so smooth, they enabled me to finish this tome relatively quickly and painlessly. I also owe much to my early training as an undergraduate at MIT in Barbara Liskov's research group. Mark Day, Bob Scheifler, Paul Johnson, and Bill Weihl, my undergraduate thesis advisor, taught me about research and real system-building, and this experience enabled me to get started on research at Stanford right away.

Finally, I thank my family for their continued support and encouragement. I thank Bill McLaughlin, one of my real brothers-in-law, for his help in putting together the final version of this thesis and taking care of the last administrative details. To my wife, Sylvia, I owe my sincerest gratitude and give my deepest love.

This research has been generously supported by the National Science Foundation, Sun Microsystems, IBM, Apple Computer, Cray Laboratories, Tandem Computers, NCR, Texas Instruments, and Digital Equipment Corporation.