Table of Contents

Chapter 1 Introduction 1

1.1 Introduction 1

1.2 The SELF Language 1

1.3 Our Research 1

1.4 Outline 2

Chapter 2 Language Features and Implementation Problems 3

2.1 Abstract Data Types 3

2.1.1 Benefits to Programmers 3

2.1.2 Implementation Effects 4

2.2 Object-Oriented Programming 4

2.2.1 Benefits to Programmers 4

2.2.1.1 Message Passing 4

2.2.1.2 Inheritance 6

2.2.2 Implementation Effects 7

2.2.3 Traditional Compromises 7

2.3 User-Defined Control Structures 9

2.3.1 Benefits to Programmers 9

2.3.2 Implementation Effects 9

2.3.3 Traditional Compromises 9

2.4 Safe Primitives 10

2.4.1 Benefits to Programmers 10

2.4.2 Implementation Effects 10

2.4.3 Traditional Compromises 10

2.5 Generic Arithmetic 10

2.5.1 Benefits to Programmers 10

2.5.2 Implementation Effects 11

2.5.3 Traditional Compromises 11

2.6 Summary 11

Chapter 3 Previous Work 13

3.1 Smalltalk Systems 13

3.1.1 The Smalltalk-80 Language 13

3.1.2 Deutsch-Schiffman Smalltalk-80 System 13

3.1.3 Typed Smalltalk and TS Optimizing Compiler 16

3.1.4 Atkinson's Hurricane Compiler 19

3.1.5 Summary 20

3.2 Statically-Typed Object-Oriented Languages 20

3.2.1 C++ 20

3.2.2 Other Fast Dispatch Mechanisms 23

3.2.3 Trellis/Owl 25

3.2.4 Emerald 26

3.2.5 Eiffel 27

3.2.6 Summary 27

3.3 Scheme Systems 27
3.3.1 The Scheme Language 27
3.3.2 The T Language and the ORBIT Compiler 28
3.3.3 Shivers' Control Flow Analysis and Type Recovery 28
3.3.4 Summary 28
3.4 Traditional Compiler Techniques 29
3.4.1 Data Flow Analysis 29
3.4.2 Abstract Interpretation 30
3.4.3 Partial Evaluation 30
3.4.4 Common Subexpression Elimination and Static Single Assignment Form 30
3.4.5 Wegman's Node Distinction 31
3.4.6 Procedure Inlining 31
3.4.7 Register Allocation 32
3.4.8 Summary 32

Chapter 4 The SELF Language 35
4.1 Basic Object Model 35
4.1.1 Object Syntax 36
4.2 Message Evaluation 36
4.2.1 Message Syntax 37
4.3 Blocks 38
4.4 Implicit Self Sends 39
4.5 Primitives 39
4.6 An Example: Cartesian and Polar Points 40

Chapter 5 Overview of the Compiler. 43
5.1 Goals of Our Work 43
5.2 Overall Approach 44
5.2.1 Representation-Level Type Information Is Key 44
5.2.2 Interface-Level Type Declarations Do Not Help 44
5.2.3 Transforming Polymorphic into Monomorphic Code 45
5.3 Organization of the Compiler 46

Chapter 6 Overall System Architecture 47
6.1 Object Storage System 47
6.1.1 Maps 47
6.1.2 Segregation 48
6.1.3 Garbage Collection 49
6.2 The Parser 50
6.3 The Run-Time System 52
6.3.1 Stacks 52
6.3.2 Blocks 53
6.4 Summary 53

Chapter 7 Inlining 55

7.1 Message Inlining 55

7.1.1 Assignable versus Constant Slots 55

7.1.2 Heuristics for Method Inlining 56

7.1.2.1 Length Checks 57

7.1.2.2 Recursion Checks 58

7.1.3 Speeding Compile-Time Message Lookup 61

7.2 Inlining Primitives 61

7.3 Summary 62

Chapter 8 Customization 63

8.1 Customization 63

8.2 Customization and Dynamic Compilation 64

8.2.1 Impact of Dynamic Compilation 64

8.2.2 Compiled Code Cache 65

8.2.3 LRU Cache Flushing Support 65

8.3 Customization and Static Compilation 65

8.4 Customization as Partial Evaluation 66

8.5 In-Line Caching 66

8.5.1 In-Line Caching and Customization 66

8.5.2 In-Line Caching and Dynamic Inheritance 68

8.5.3 In-Line Caching and _Performs 68

8.6 Future Work 69

8.7 Summary 70

Chapter 9 Type Analysis 71

9.1 Internal Representation of Programs and Type Information 71

9.1.1 Control Flow Graph 71

9.1.2 Names 72

9.1.3 Values 73

9.1.4 Types 75

9.1.4.1 Map Types 75

9.1.4.2 Constant Types 76

9.1.4.3 Integer Subrange Types 76

9.1.4.4 The Unknown Type 76

9.1.4.5 Union Types 76

9.1.4.6 Exclude Types 77

9.2 Type Analysis 78

9.2.1 Assignment Nodes 78

9.2.2 Merge Nodes 78

9.2.3 Branch Nodes 78

9.2.4 Message Send Nodes 78

9.2.5 Primitive Operation Nodes 78

9.3 Type Casing 79

9.4 Type Prediction 80

9.5 Block Analysis 82
9.5.1 Deferred Computations 82
9.5.2 Analysis of Exposed Blocks 85
9.6 Common Subexpression Elimination 86
9.6.1 Eliminating Redundant Arithmetic Calculations 86
9.6.2 Eliminating Redundant Memory References 87
9.6.3 Future Work: Eliminating Unnecessary Object Creations and Initializations 89
9.7 Summary 91

Chapter 10 Splitting 93

10.1 A Motivating Example 93
10.2 Reluctant Splitting 103
10.2.1 Splittable versus Unsplittable Union Types 104
10.2.2 Splitting Multiple Union Types Simultaneously 105
10.2.2.1 Paths 106
10.2.2.2 Splitting Merge Nodes 108
10.2.2.3 Splitting Branch Nodes 109
10.2.2.4 Implementation of Paths 110
10.2.3 Heuristics for Reluctant Splitting 112
10.2.3.1 Costs 112
10.2.3.2 Weights 113
10.2.3.3 Cost and Weight Thresholds 114
10.2.4 Future Work 115
10.2.5 Related Work 115
10.3 Eager Splitting 116
10.4 Divided Splitting 122
10.5 Lazy Compilation of Uncommon Branches 123
10.6 Summary 126

Chapter 11 Type Analysis and Splitting of Loops .127

11.1 Pessimistic Type Analysis 127
11.2 Traditional Iterative Data Flow Analysis 128
11.3 Iterative Type Analysis 128
11.3.1 Compatibility 131
11.3.2 Initial Loop Head Type Information 132
11.3.3 Iterating the Analysis 134
11.4 Iterative Type Analysis and Splitting 136
11.5 Summary 140

Chapter 12 Back-End of the Compiler 141

12.1 Global Register Allocation 141
12.1.1 Assigning Variables to Names 142
12.1.2 Live Variable Analysis 142
12.1.3 Register Selection 143
12.1.4 Inserting Register Moves 144
12.1.5 Future Work 144
12.2 Common Subexpression Elimination of Constants 145

12.3 Eliminating Unneeded Computations 145
12.4 Filling Delay Slots 145
12.5 Code Generation 146
12.6 Portability Issues 146
12.7 Summary 147

Chapter 13 Programming Environment Support 149

13.1 Support for Source-Level Debugging 150
13.1.1 Compiler-Generated Debugging Information 150
13.1.2 Virtual/Physical Program Counter Translation 151
13.1.3 Current Debugging Primitives 152
13.1.4 Interactions between Debugging and Optimization 152
13.2 Support for Programming Changes 153
13.2.1 Ways of Supporting Programming Changes 153
13.2.2 Dependency Links 154
13.2.3 Invalidation 155
13.3 Summary 157

Chapter 14 Performance Evaluation 159

14.1 Methodology 159
14.1.1 The Benchmarks 159
14.1.2 The Hardware 160
14.1.3 Charting Technique 160
14.2 Performance versus Other Languages 161
14.3 Relative Effectiveness of the Techniques 165
14.4 Some Remaining Sources of Overhead 167
14.5 Summary 169

Chapter 15 Conclusions. 171

15.1 Results of this Work 171
15.2 Applicability of the Techniques 171
15.3 Future Work 172
15.4 Conclusion 172

Appendix A Object Formats 173

A.1 Tag Formats 173
A.2 Object Layout 174

Appendix B Detailed Performance Evaluation 177

B.1 Detailed Description of the Benchmarks 177
B.2 Measurement Procedures 178
B.3 Relative Effectiveness of the Techniques 179
B.3.1 Inlining 179
B.3.2 Caching Compile-Time Message Lookups 180
B.3.3 Customization 180

B.3.4 In-Line Caching 181
B.3.5 Type Analysis 182
B.3.6 Value-Based Type Analysis 182
B.3.7 Integer Subrange Analysis 183
B.3.8 Type Prediction 184
B.3.9 Block Analysis 185
B.3.9.1 Deferring Block Creations 185
B.3.9.2 Exposed Block Analysis 186
B.3.10 Common Subexpression Elimination 186
B.3.11 Register Allocation 187
B.3.12 Eliminating Unneeded Computations 189
B.3.13 Filling Delay Slots 190
B.3.14 Summary of Effectiveness of the Techniques 190
B.4 Splitting Strategies 191
B.4.1 Reluctant Splitting Strategies 191
B.4.2 Lazy Compilation of Uncommon Branches 193
B.4.3 Vector Type Prediction 193
B.4.4 Eager Splitting Strategies 196
B.4.5 Divided Splitting 198
B.4.6 Summary of Splitting Strategies 198
B.5 Some Remaining Sources of Overhead 201
B.5.1 Type Tests 201
B.5.2 Overflow Checking 202
B.5.3 Array Bounds Checking 203
B.5.4 Block Zapping 203
B.5.5 Primitive Failure Checking 205
B.5.6 Debugger-Visible Names 206
B.5.7 Interrupt Checking and Stack Overflow Checking 207
B.5.8 LRU Compiled Method Reclamation Support 208
B.5.9 Summary of Remaining Sources of Overhead 208
B.6 Additional Space Costs 209

Appendix C Raw Benchmark Data. 211
C.1 recur 212
C.2 sumTo 212
C.3 sumFromTo 213
C.4 fastSumTo 213
C.5 nestedLoop 214
C.6 atAllPut 214
C.7 sumAll 215
C.8 incrementAll 215
C.9 tak 216
C.10 takl 216
C.11 sieve 217
C.12 perm 217
C.13 towers 218
C.14 queens 218
C.15 intmm 219
C.16 quick 219
C.17 bubble 220
C.18 tree 220
C.19 oo-perm 221

C.20 oo-towers 221
C.21 oo-queens 222
C.22 oo-intmm 222
C.23 oo-quick 223
C.24 oo-bubble 223
C.25 oo-tree 224
C.26 puzzle 224
C.27 richards 225
C.28 parser 225
C.29 primMaker 226
C.30 pathCache 226

Bibliography 227