Source Code Optimization Techniques for Data Flow Dominated Embedded Software

Dissertation
zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Universität Dortmund
am Fachbereich Informatik
von

Heiko Falk

Dortmund
2004

Tag der mündlichen Prüfung:	03. Juni 2004
Dekan / Dekanin:	Prof. Dr. Bernhard Steffen
Gutachter / Gutachterinnen:	Prof. Dr. Francky Catthoor,
	Prof. Dr. Peter Marwedel,
	Prof. Dr. Peter Padawitz

Contents

List of Figures **ix**

List of Tables **xiii**

1 Introduction **1**
1.1 Why Source Code Optimization? 3
1.1.1 Abstraction Levels of Code Optimization 4
1.1.2 Survey of the traditional Code Optimization Process . 6
1.1.3 Scopes for Code Optimization 8
1.2 Target Application Domain 10
1.3 Goals and Contributions 12
1.4 Outline of the Thesis 13
1.5 Author's Contribution to the Thesis 14

2 Existing Code Optimization Techniques **15**
2.1 Specification Optimization 16
2.2 Algorithm Selection 17
2.3 Memory Hierarchy Exploitation 18
2.4 Processor-independent Source Code Optimizations 19
2.5 Processor-specific Source Code Optimizations 20
2.6 Compiler Optimizations 22
2.6.1 Loop Optimizations for High-Performance Computing . 22
2.6.2 Code Generation for embedded Processors 23

3 Fundamental Concepts for Optimization and Evaluation **25**
3.1 Polyhedral Modeling 25
3.2 Optimization using Genetic Algorithms 29
3.3 Benchmarking Methodology 34
3.3.1 Profiling of Pipeline and Cache Performance 35
3.3.2 Compilation for Runtime and Code Size Measurement 36
3.3.3 Estimation of Energy Dissipation 38

3.4 Summary 39

4 Intermediate Representations 41
4.1 Low-level Intermediate Representations 43
4.1.1 GNU RTL 43
4.1.2 Trimaran ELCOR IR 44
4.2 Medium-level Intermediate Representations 45
4.2.1 Sun IR 45
4.2.2 IR-C / LANCE 46
4.3 High-level Intermediate Representations 47
4.3.1 SUIF 47
4.3.2 IMPACT 49
4.4 Selection of an IR for Source Code Optimization 49
4.5 Summary 52

5 Loop Nest Splitting 55
5.1 Introduction 55
5.1.1 Control Flow Overhead in Data Flow dominated Software 56
5.1.2 Control Flow Overhead caused by Data Partitioning . 57
5.1.3 Splitting of Loop Nests for Control Flow Optimization 60
5.2 Related Work 62
5.3 Analysis and Optimization Techniques for Loop Nest Splitting 65
5.3.1 Preliminaries 67
5.3.2 Condition Satisfiability 70
5.3.3 Condition Optimization 72
5.3.3.1 Chromosomal Representation 73
5.3.3.2 Fitness Function 77
5.3.3.3 Polytope Generation 83
5.3.4 Global Search Space Construction 84
5.3.5 Global Search Space Exploration 87
5.3.5.1 Chromosomal Representation 88
5.3.5.2 Fitness Function 89
5.3.6 Source Code Transformation 95
5.3.6.1 Generation of the splitting If-Statement . . . 95
5.3.6.2 Loop Nest Duplication 98
5.4 Extensions for Loops with non-constant Bounds 100
5.5 Experimental Results 106
5.5.1 Stand-alone Loop Nest Splitting 107
5.5.1.1 Pipeline and Cache Performance 109
5.5.1.2 Execution Times and Code Sizes 113
5.5.1.3 Energy Consumption 115

5.5.2 Combined Data Partitioning and Loop Nest Splitting for energy-efficient Scratchpad Utilization 117
5.5.2.1 Execution Times and Code Sizes 118
5.5.2.2 Energy Consumption 119
5.6 Summary 121

6 Advanced Code Hoisting 123
6.1 A motivating Example 123
6.2 Related Work 128
6.3 Analysis Techniques for Advanced Code Hoisting 133
6.3.1 Common Subexpression Identification 134
6.3.1.1 Collection of equivalent Expressions 135
6.3.1.2 Computation of Live Ranges of Expressions . 137
6.3.2 Determination of the outermost Loop for a CSE 145
6.3.3 Computation of Execution Frequencies using Polytope Models 145
6.4 Experimental Results 161
6.4.1 Pipeline and Cache Performance 161
6.4.2 Execution Times and Code Sizes 165
6.4.3 Energy Consumption 167
6.5 Summary 168

7 Ring Buffer Replacement 171
7.1 Motivation 172
7.2 Optimization Steps 178
7.2.1 Ring Buffer Scalarization 178
7.2.2 Loop Unrolling for Ring Buffers 179
7.3 Experimental Results 181
7.3.1 Pipeline and Cache Performance 182
7.3.2 Execution Times and Code Sizes 185
7.3.3 Energy Consumption 187
7.4 Summary 188

8 Summary and Conclusions 191
8.1 Summary and Contribution to Research 191
8.2 Future Work 194

A Experimental Comparison of SUIF and IR-C / LANCE 199

B Benchmarking Data for Loop Nest Splitting 201
B.1 Values of performance-monitoring Counters 201
B.1.1 Intel Pentium III 201
B.1.2 Sun UltraSPARC III 203
B.1.3 MIPS R10000 205
B.2 Execution Times and Code Sizes 206
B.3 Energy Consumption of an ARM7TDMI Core 209
B.4 Combined Data Partitioning and Loop Nest Splitting 210
B.4.1 Execution Times and Code Sizes 210
B.4.2 Energy Consumption 210

C Benchmarking Data for Advanced Code Hoisting 213
C.1 Values of performance-monitoring Counters 213
C.1.1 Intel Pentium III 213
C.1.2 Sun UltraSPARC III 214
C.1.3 MIPS R10000 215
C.2 Execution Times and Code Sizes 216
C.3 Energy Consumption of an ARM7TDMI Core 218

D Benchmarking Data for Ring Buffer Replacement 219
D.1 Values of performance-monitoring Counters 219
D.1.1 Intel Pentium III 219
D.1.2 Sun UltraSPARC III 220
D.1.3 MIPS R10000 220
D.2 Execution Times and Code Sizes 221
D.3 Energy Consumption of an ARM7TDMI Core 222

References 223

Index 235

List of Figures

1.1 Examples of typical Embedded Systems 2
1.2 Abstraction Levels and achievable Improvements of Code Optimizations 5
1.3 Structure of Optimizing Compilers 7
1.4 Implementation of a saturating Addition 8

2.1 Abstraction Levels of Code Optimizations 15
2.2 Overview of the DTSE Methodology 18
2.3 Replacement of complex Code by a TI C6x Intrinsic 21

3.1 A convex Polytope 26
3.2 A non-convex Set of Points 26
3.3 Schematic Overview of Genetic Algorithms 30
3.4 Genetic Algorithm Encoding 30
3.5 Crossover 31
3.6 Mutation 31
3.7 Overview of the Workflow employed in the *encc* Compiler . . . 38

4.1 Experimental Comparison of Runtimes based on SUIF and IR-C / LANCE 51

5.1 A typical Loop Nest 57
5.2 A typical Code Fragment before and after Data Partitioning . 59
5.3 Loop Nest after Splitting 61
5.4 Conventional Loop Splitting 62
5.5 Conventional Loop Unswitching 63
5.6 Design Flow of Loop Nest Splitting 66
5.7 Conditions as Polytopes 71
5.8 Structure of the Fitness Function for Global Search Space Exploration 92
5.9 A typical Loop Nest with variable Loop Bounds 101
5.10 Structure of the Fitness Function for Condition Optimization . 103

5.11 A Loop Nest with variable Loop Bounds after Splitting 107
5.12 Relative Pipeline and Cache Behavior for Intel Pentium III . . 109
5.13 Relative Pipeline and Cache Behavior for Sun UltraSPARC III 111
5.14 Relative Pipeline and Cache Behavior for MIPS R10000 113
5.15 Relative Runtimes after Loop Nest Splitting 114
5.16 Relative Code Sizes after Loop Nest Splitting 115
5.17 Relative Energy Consumption after Loop Nest Splitting 116
5.18 Relative Runtimes after Data Partitioning and Loop Nest Splitting 118
5.19 Relative Code Sizes after Data Partitioning and Loop Nest Splitting 118
5.20 Relative Energy Consumption after Data Partitioning 120
5.21 Relative Energy Consumption after Data Partitioning and Loop Nest Splitting 121

6.1 A Code Sequence of GSM Codebook Search 125
6.2 The GSM Codebook Search after Code Hoisting without Control Flow Analysis 126
6.3 The GSM Codebook Search after Advanced Code Hoisting . . 127
6.4 Local Common Subexpression Elimination 129
6.5 Global Common Subexpression Elimination 130
6.6 Loop-invariant Code Motion 131
6.7 Parts of DTSE causing Control Flow and Addressing Overhead 132
6.8 A Fragment of the CAVITY Benchmark before and after DTSE 133
6.9 Algorithm for Advanced Code Hoisting 134
6.10 Algorithm to identify Common Subexpressions 135
6.11 The SUIF Abstract Syntax Tree 136
6.12 Bottom-up Syntax Tree Traversal 139
6.13 Algorithm to determine the Parent of $expr_{i,c}$ and $expr_{i,c+1}$ . . 140
6.14 Basic Algorithm for Syntax Tree Traversal 141
6.15 Algorithm to compute Live Ranges 144
6.16 Algorithm to determine the outermost Loop for a CSE 146
6.17 General Structure of Control Flow surrounding an Expression $expr$ 148
6.18 Relative Pipeline and Cache Behavior for Intel Pentium III . . 162
6.19 Relative Pipeline and Cache Behavior for Sun UltraSPARC III 163
6.20 Relative Pipeline and Cache Behavior for MIPS R10000 164
6.21 Relative Runtimes after Advanced Code Hoisting 165
6.22 Runtime Comparison of Advanced Code Hoisting and Common Subexpression Elimination 166
6.23 Relative Code Sizes after Advanced Code Hoisting 167

6.24 Relative Energy Consumption after Advanced Code Hoisting . 168

7.1 Structure of a Ring Buffer 171
7.2 Realization of a Ring Buffer using an Array 171
7.3 Original Structure of the CAVITY Application 172
7.4 Structure of the CAVITY Application after Data Reuse Transformation 173
7.5 Access Analysis of Line Buffers during In-Place Mapping . . . 174
7.6 Scalar Replacement of Array Elements 175
7.7 Principal operating Mode of Ring Buffer Replacement 176
7.8 Ring Buffer Scalarization 178
7.9 Loop Unrolling for Ring Buffers 180
7.10 Relative Pipeline and Cache Behavior for Intel Pentium III . . 182
7.11 Relative Pipeline and Cache Behavior for Sun UltraSPARC III 184
7.12 Relative Pipeline and Cache Behavior for MIPS R10000 185
7.13 Relative Runtimes after Ring Buffer Replacement 186
7.14 Relative Code Sizes after Ring Buffer Replacement 187
7.15 Relative Energy Consumption after Ring Buffer Replacement . 188

8.1 Relative Runtimes after entire Optimization Sequence 193
8.2 Possible Speed / Size Trade-Offs for Loop Nest Splitting 195

B.1 Structure of the Intel Pentium III Architecture 202
B.2 Structure of the Sun UltraSPARC III Architecture 204

List of Tables

5.1 Characteristic Properties of Benchmarks 108

A.1 Runtimes of original Benchmark Versions 199
A.2 Runtimes of SUIF Benchmark Versions 200
A.3 Runtimes of IR-C Benchmark Versions 200

B.1 PMC Values for original Benchmark Versions 203
B.2 PMC Values for Benchmarks after Loop Nest Splitting 203
B.3 PIC Values for original Benchmark Versions 204
B.4 PIC Values for Benchmarks after Loop Nest Splitting 205
B.5 PC Values for original Benchmark Versions 205
B.6 PC Values for Benchmarks after Loop Nest Splitting 206
B.7 Compilers and Optimization Levels for Runtime and Code Size Measurements 206
B.8 Runtimes of original Benchmark Versions 207
B.9 Runtimes of Benchmarks after Loop Nest Splitting 207
B.10 Code Sizes of original Benchmark Versions 208
B.11 Code Sizes of Benchmarks after Loop Nest Splitting 208
B.12 Energy Consumption of original Benchmark Versions 209
B.13 Energy Consumption of Benchmarks after Loop Nest Splitting 209
B.14 Runtimes of Benchmarks after Data Partitioning and Loop Nest Splitting 210
B.15 Code Sizes of Benchmarks after Data Partitioning and Loop Nest Splitting 210
B.16 Energy Consumption of original Benchmark Versions 211
B.17 Energy Consumption of Benchmarks after Data Partitioning . 211
B.18 Energy Consumption of Benchmarks after Loop Nest Splitting 211

C.1 PMC Values for original Benchmark Versions 214
C.2 PMC Values for Benchmarks after Advanced Code Hoisting . 214
C.3 PIC Values for original Benchmark Versions 215

C.4 PIC Values for Benchmarks after Advanced Code Hoisting . . 215
C.5 PC Values for original Benchmark Versions 215
C.6 PC Values for Benchmarks after Advanced Code Hoisting . . . 216
C.7 Runtimes of original Benchmark Versions 216
C.8 Runtimes of Benchmarks after Advanced Code Hoisting 217
C.9 Code Sizes of original Benchmark Versions 217
C.10 Code Sizes of Benchmarks after Advanced Code Hoisting . . . 217
C.11 Energy Consumption of original Benchmark Versions 218
C.12 Energy Consumption of Benchmarks after Advanced Code Hoisting 218

D.1 PMC Values of Ring Buffer Replacement for CAVITY 220
D.2 PIC Values of Ring Buffer Replacement for CAVITY 220
D.3 PC Values of Ring Buffer Replacement for CAVITY 221
D.4 Runtimes of Ring Buffer Replacement for CAVITY 221
D.5 Code Sizes of Ring Buffer Replacement for CAVITY 222
D.6 Energy Consumption of Ring Buffer Replacement for CAVITY 222