An Architecture for Combinator Graph Reduction

Philip J. Koopman Jr.

To my parents

Contents

List of Tables xi

List of Illustrations xiii

Preface xv

1. Introduction 1
1.1. OVERVIEW OF THE PROBLEM AREA 1
1.2. ORGANIZATION OF THIS BOOK 3

2. Background 5
2.1. PROBLEM DEFINITION 5
2.1.1. Lazy Functional Programming 5
2.1.2. Closure Reduction and Graph Reduction 6
2.1.3. Performance Inefficiencies 8
2.2. PREVIOUS RESEARCH 9
2.2.1. Miranda 9
2.2.2. Hyperlazy Evaluation 10
2.2.3. The G-Machine 10
2.2.4. TIM 11
2.2.5. NORMA 12
2.2.6. The Combinatorgraph Reducer 12
2.2.7. Analysis and Summary 13
2.3. APPROACH OF THIS RESEARCH 14

3. Development of the TIGRE Method 15
3.1. THE CONVENTIONAL GRAPH REDUCTION METHOD . 15
3.2. FAST INTERPRETIVE EXECUTION OF GRAPHS 17
3.3. DIRECT EXECUTION OF GRAPHS 19

4. Implementation of the TIGRE Machine 25
4.1. THE TIGRE ABSTRACT MACHINE 25
4.1.1. Hardware Definition 25

4.1.2. TIGRE Assembly Language 26
4.1.3. A TIGRE Compiler 30
4.2. MAPPING OF TIGRE ONTO VARIOUS EXECUTION MODELS 31
4.2.1. Mapping of TIGRE Onto the C Execution Model 31
4.2.2. Mapping of TIGRE Assembly Language Onto a VAX . . 33
4.2.3. Mapping of TIGRE Assembly Language Onto a MIPS R2000 35
4.2.4. Translation to Other Architectures 36
4.3. TIGRE ASSEMBLER DEFINITIONS OF COMBINATORS 37
4.3.1. Non-Strict Combinators 37
4.3.1.1. 1-Projection Combinators 37
4.3.1.2. Simple Graph Rewriting Combinators 38
4.3.2. Strict Combinators 38
4.3.2.1. Totally Strict Combinators 39
4.3.2.2. Partially Strict Combinators 40
4.3.3. List Manipulation Combinators 41
4.3.4. Supercombinators 43
4.4. SOFTWARE SUPPORT 45
4.4.1. Garbage Collection 45
4.4.2. Other Software Support 47

5. TIGRE Performance 49
5.1. TIGRE PERFORMANCE ON VARIOUS PLATFORMS . . . 49
5.1.1. TIGRE Performance for the Turner Set 51
5.1.2. TIGRE Performance for Supercombinator Compilation 53
5.2. COMPARISONS WITH OTHER METHODS 54
5.2.1. Miranda 54
5.2.2. Hyperlazy Evaluation 55
5.2.3. The G-Machine 55
5.2.4. TIM 56
5.2.5. NORMA 56
5.3. TIGRE VERSUS OTHER LANGUAGES 57
5.3.1. Non-Lazy Language: T Version 3.0 57
5.3.2. Imperative Language: MIPS R2000 C Compiler 58
5.4. ANALYSIS OF PERFORMANCE 59

6. Architectural Metrics 63
6.1. CACHE BEHAVIOR 63
6.1.1. Exhaustive Search of the Cache Design Space 63
6.1.2. Parametric Analysis 69

6.1.2.1. Write Allocation 70
6.1.2.2. Cache Size 72
6.1.2.3. Block Size 73
6.1.2.4. Associativity 75
6.1.2.5. Replacement Policy 76
6.1.2.6. Write-Through Policy 76
6.1.3. A Desirable Cache Strategy 77
6.2. PERFORMANCE OF REAL HARDWARE 78
6.2.1. Simulation Results for a DECstation 3100 78
6.2.2. Comparison with Actual Measurements 82
6.3. DYNAMIC PROGRAM BEHAVIOR 84
6.3.1. Heap Memory Use 84
6.3.2. Stack Memory Use 86

7. The Potential of Special-Purpose Hardware 89
7.1. DECSTATION 3100 AS A BASELINE 89
7.2. IMPROVEMENTS IN CACHE MANAGEMENT 90
7.2.1. Copy-Back Cache 90
7.2.2. Increased Block Size 90
7.2.3. Prefetch on Read Misses 91
7.3. IMPROVEMENTS IN CPU ARCHITECTURE 92
7.3.1. Stack Unwinding Support 92
7.3.2. Stack Access Support 93
7.3.3. Doubleword Store 94
7.4. PERFORMANCE IMPROVEMENT POSSIBILITIES 94

8. Conclusions 97
8.1. CONTRIBUTIONS OF THIS RESEARCH 97
8.2. AREAS FOR FURTHER RESEARCH 98

Appendix A. A Tutorial on Combinator Graph Reduction . 101
A.1. FUNCTIONAL PROGRAMS 101
A.2. MAPPING FUNCTIONAL PROGRAMS TO LAMBDA CALCULUS 102
A.3. MAPPING LAMBDA CALCULUS TO SK-COMBINATORS 103
A.4. MAPPING SK-COMBINATOR EXPRESSIONS ONTO A GRAPH 105
A.5. THE TURNER SET OF COMBINATORS 117
A.6. SUPERCOMBINATORS 120
A.7. INHERENT PARALLELISM IN COMBINATOR GRAPHS 121

Appendix B. Selected TIGRE Program Listings 123
B.1. REDUCE.H 123
B.2. KERNEL.C 126
B.3. TIGRE.S 129
B.4. MIPS.S 136
B.5. HEAP.H 144
B.6. HEAP.C 145

References 149

Index 153

List of Tables

Table 5-1. TIGRE performance on a variety of platforms 50
Table 5-2. Benchmark listings 52
Table 5-3. TIGRE speedups using supercombinator compilation . . 53
Table 5-4. Performance of TIGRE versus Miranda 54
Table 5-5. Performance of TIGRE versus Hyperlazy evaluation . . 55
Table 5-6. Performance of TIGRE versus TIM 56
Table 5-7. Performance of TIGRE versus NORMA 56
Table 5-8. TIGRE performance compared to T3.0 57
Table 5-9. TIGRE performance compared to C 58
Table 5-10. C program listings for comparison with TIGRE 60

Table 6-1. Cache performance simulation results for TIGRE on a MIPS R2000 65
Table 6-2. Baseline for parametric analysis 69
Table 6-3. TIGRE performance with varying cache write allocation strategy 70
Table 6-4. TIGRE performance with varying cache associativity . . 76
Table 6-5. TIGRE performance with varying cache replacement policies 76
Table 6-6. TIGRE performance with varying cache write-through strategy 77
Table 6-7. Baseline for DECstation 3100 analysis 79
Table 6-8. Performance with varying cache write allocation strategy 79
Table 6-9. Cache performance with varying cache associativity . . 80
Table 6-10. Cache performance with varying cache write-through strategy 82
Table 6-11. TIGRE use of heap memory 84
Table 6-12. TIGRE use of stack memory for fib 86

Table 7-1. Summary of TIGRE DECstation 3100 performance characteristics 90
Table 7-2. Summary of possible performance improvements 95

Table A-1. Non-strict members of the Turner combinator set . . . 117
Table A-2. Turner Set optimizations 119

List of Illustrations

Figure 2-1. Evolution of lazy functional program implementation techniques 13

Figure 3-1. Basic structure of a node 15
Figure 3-2. Example for expression ((+ 11) 22) 16
Figure 3-3. Example using indirection nodes for constants 17
Figure 3-4. Example using **LIT** nodes instead of indirection nodes for constants 18
Figure 3-5. Example with tag fields removed 19
Figure 3-6. An example TIGRE program graph, emphasizing the left spine 20
Figure 3-7. A TIGRE program graph with only subroutine call pointers 21
Figure 3-8. VAX assembly language implementation of a TIGRE expression 22

Figure 4-1. A block diagram of the TIGRE abstract machine 26
Figure 4-2. The **S'** combinator 29
Figure 4-3. Mapping of the TIGRE abstract machine onto C 31
Figure 4-4. Mapping of the TIGRE abstract machine onto a VAX 8800 34
Figure 4-5. Mapping of the TIGRE abstract machine onto a MIPS R2000 35
Figure 4-6. The **IF** combinator 41
Figure 4-7. The **P** combinator 42
Figure 4-8. The **U** combinator 43
Figure 4-9. The **$FIB** supercombinator 44

Figure 6-1. TIGRE performance with varying cache size 72
Figure 6-2. TIGRE performance with varying cache block size . . . 74
Figure 6-3. Cache performance with varying cache size 80
Figure 6-4. Performance with varying cache block size 81

Figure A-1. The function and argument structure of a node . . . 106
Figure A-2. A function argument pair 106
Figure A-3. A shared subtree 107
Figure A-4. Graph to add 11 and 22 107
Figure A-5. Operation of the **I** combinator 107
Figure A-6. Operation of the **K** combinator 108
Figure A-7. Operation of the **S** combinator 109
Figure A-8. An addition example 110
Figure A-9. Doubling function 110
Figure A-10. Doubling function applied to argument 111
Figure A-11. Reduction step 1 112
Figure A-12. Reduction step 2 112
Figure A-13. Reduction step 3 113
Figure A-14. Reduction step 4 113
Figure A-15. Reduction step 5 114
Figure A-16. Reduction step 6 114
Figure A-17. Reduction step 7 115
Figure A-18. Reduction step 8 115
Figure A-19. Reduction step 9 115
Figure A-20. Reduction step 10 116
Figure A-21. Reduction step 11 116
Figure A-22. Operation of the **B** combinator 118
Figure A-23. Operation of the **C** combinator 119

Preface

This book is based on my Ph.D. thesis for the Electrical and Computer Engineering Department at Carnegie Mellon University. It is the result of a computer engineer's journey into the realm of Computer Science theory and programming language implementation. The point of view taken is that of an engineer, and focuses on how to solve a problem (in this case, fast combinator reduction) efficiently. The book is split into two major areas. The first area is the development of the TIGRE graph reducer, along with performance measurements on a variety of machines. The second area is an architectural analysis of TIGRE's behavior.

This research would not have been possible without the support of two faculty members to cover the two areas of the research. Dan Siewiorek has helped me mature as an architect, provided guidance for the engineering half of the thesis, and was supportive when I decided to pursue an unusual (for an engineer) research direction. Peter Lee introduced me to combinator reduction, and provided encouragement, software support, and expert editing assistance. The other members of my thesis committee, Rob Rutenbar and Tom Hand, also helped guide the course of the research. John Dorband at NASA/Goddard gave me the funding support and freedom I needed to perform the research (which was funded by NASA/Goddard under contract NAG-5-1046).

During the quest for my degree, many people have helped in ways large and small. Some of the main contributors are: my wife, Mary, for her support during the stressful times, and tolerance of late nights/early mornings; Glen Haydon, who inspired my interest in threaded architectural techniques and provided helpful insight into the Ph.D. process; and Dom Carlino, who has provided sage advice and encouragement.