Contents

Acknowledgements **vi**

1 Introduction **1**

1.1 Parallel computers . . . 1

1.1.1 Shared-memory multiprocessors . . . 3

1.1.2 Distributed-memory multicomputers . . . 5

1.2 Parallel programming . . . 6

1.2.1 Shared data; the evolution . . . 6

1.2.2 Message passing; the revolution . . . 7

1.2.3 Distributed shared memory; best of both? . . . 7

1.3 Functional programming . . . 9

1.4 The EIT Reduction Machine project . . . 11

1.5 WYBERT . . . 13

1.6 Outline . . . 14

2 Functional programming and its implementation **17**

2.1 Reduction . . . 19

2.2 Values and expressions . . . 20

2.2.1 Lists . . . 22

2.3 Higher order functions . . . 23

2.4 Lazy evaluation . . . 25

2.4.1 Infinite datastructures . . . 27

2.4.2 Stream programming . . . 29

2.5 Graph reduction . . . 31

2.5.1 Strictness analysis . . . 35

2.6 Summary . . . 37

3 Parallel implementations of lazy functional languages 39
3.1 Parallel graph reduction 41
3.1.1 Generating parallelism 42
3.1.2 Global address-space support 44
3.1.3 Storage management 48
3.1.4 Task scheduling 52
3.1.5 Controlling parallelism 55
3.2 Survey 59
3.2.1 $<\nu,G>$ 60
3.2.2 AMPGR 61
3.2.3 GAML 62
3.2.4 Flagship 63
3.2.5 PAM 64
3.2.6 HDG 65
3.2.7 PABC 66
3.2.8 GRIP 67
3.2.9 HyperM 68
3.3 Comparison 70
3.3.1 Performance 74
3.4 Conclusions 75

4 WYBERT: graph reduction on shared memory 77
4.1 Design considerations 78
4.1.1 Divide-and-conquer parallelism 79
4.1.2 The FRATS reduction strategy 81
4.1.3 Task scheduling 82
4.1.4 Local garbage collection 82
4.1.5 Evaluation method 84
4.2 FRATS: A parallel reduction strategy 86
4.2.1 Termination through transformation 88
4.2.2 Curried functions 90
4.2.3 Transformation methodology 93
4.2.4 Performance consequences 94
4.3 Top-of-Stack scheduling 97
4.3.1 Worst case behaviour 99
4.3.2 Performance from Simulation Studies 101
4.4 Memory management for parallel tasks 106
4.4.1 Local copying garbage collection 108

4.4.2 Evaluation 115
4.5 Discussion 120

5 The FAST/FCG compiler 123
5.1 The front end 125
5.2 The assembler 128
5.2.1 Implementation 130
5.3 Graph representation 131
5.4 The code generator 135
5.4.1 Tail call optimisation 138
5.4.2 Compile-time stack simulation 139
5.4.3 Inlining of primitive functions 142
5.4.4 Parameters passed in registers 143
5.4.5 Life-time analysis 145
5.5 Performance 147
5.6 Conclusions 151

6 Experimental results 153
6.1 FRATS reduction strategy 155
6.1.1 Spark-and-wait implementation 155
6.1.2 Sandwich versus spark-and-wait 156
6.2 Scheduling 158
6.3 Storage management 160
6.4 Garbage collection 163
6.5 Execution profiling 165
6.5.1 Queens 166
6.5.2 Wang 168
6.5.3 Wave 170
6.5.4 Performance characteristics 173

7 Conclusions 177

Bibliography 183

Index 197