# Contents


**1 Preface** 1

**2 The Weight Finder - An Advanced Profiler for Fortran Programs** 7
*by Thomas Fahringer*
2.1 Introduction . . . 7
2.2 Prerequisite . . . 10
2.3 The Weight Finder . . . 10
2.3.1 Choosing sequential program parameters . . . 11
2.3.2 Instrumentation . . . 13
2.3.3 Optimization . . . 16
2.3.4 Compile and Execute . . . 19
2.3.5 Attribute and Visualize . . . 19
2.4 Adaptation of Profile Data . . . 19
2.4.1 Program transformations . . . 20
2.4.2 Problem Size . . . 24
2.5 Conclusion and Future Work . . . 25

**3 Predicting Execution Times of Sequential Scientific Kernels** 32
*by Neil B. MacDonald*
3.1 Motivation . . . 32
3.2 Deriving time formulae for code fragments . . . 33
3.3 Obtaining a platform model . . . 34
3.4 Examples . . . 37
3.4.1 Fragment A . . . 37
3.4.2 Fragment B . . . 38
3.4.3 Fragment C . . . 39
3.4.4 Fragment D . . . 40
3.4.5 Fragment E . . . 41
3.4.6 Fragment F . . . 42
3.4.7 Summary of results . . . 43
3.5 Discussion and Further Work . . . 43

**4 Isolating the Reasons for the Performance of Parallel Machines on Numerical Programs** 45
*by Arno Formella, Silvia M. Müller, Wolfgang J. Paul, and Anke Bingert*
4.1 Introduction . . . 45
4.2 Micro Measurements . . . 46
4.2.1 Micro Measurements for a Node Processor . . . 47
4.2.2 Micro Measurements for Communication Networks . . . 52
4.3 Measurements . . . 56
4.3.1 Measurements of the Serial Kernels . . . 56

4.3.2 Measurements of the Parallel Kernels . . . . . 62
4.4 Algorithms . . . . . 67
4.4.1 CG–method . . . . . 67
4.4.2 PDE1–method . . . . . 69
4.4.3 PDE2–method . . . . . 70
4.5 Analysis of the Programs . . . . . 71
4.5.1 Serial Versions . . . . . 71
4.5.2 Parallel Versions . . . . . 73
4.6 Conclusion . . . . . 76

**5 Targeting Transputer Systems, Past and Future** **78**
*by Denis Nicole*
5.1 Introduction . . . . . 78
5.2 The T800 family . . . . . 79
5.3 The T9000 family . . . . . 81
5.4 The Chameleon family . . . . . 82

**6 Adaptor: A Compilation System for Data Parallel Fortran Programs** **84**
*by Thomas Brandes*
6.1 Introduction . . . . . 84
6.2 The Adaptor Compilation System . . . . . 85
6.2.1 Properties of Adaptor . . . . . 85
6.2.2 Overview of Adaptor . . . . . 86
6.2.3 The Input Language . . . . . 87
6.2.4 Programming Models for the Generated Programs . . . . . 87
6.2.5 Interactive Source-to-Source Transformation . . . . . 87
6.2.6 Realization of the Translation . . . . . 88
6.2.7 Distributed Array Library . . . . . 89
6.2.8 Visualization of the Run Time Behavior . . . . . 90
6.2.9 Availability . . . . . 90
6.2.10 Related Work . . . . . 90
6.3 Results of Benchmark Codes . . . . . 90
6.3.1 The Purdue Set . . . . . 91
6.3.2 Comparison of Sequential and Parallel Version . . . . . 91
6.3.3 Efficiency and Scalability . . . . . 92
6.3.4 Adaptor vs. hand-coded message passing programs . . . . . 93
6.3.5 Full vs. Loosely Synchronous Execution . . . . . 93
6.4 Results of Application Codes . . . . . 95
6.4.1 HYDFLO: a CM Fortran Code for Fluid Dynamics . . . . . 95
6.4.2 ESM: a Fortran 90 Code for Circulation . . . . . 95
6.4.3 IFS: a Fortran 77 Code for Weather Prediction . . . . . 95
6.5 Summary . . . . . 96

**7 SNAP! Prototyping a Sequential and Numerical Application Parallelizer** **99**
*by Rolf Hänisch*
7.1 Introduction . . . . . 99

7.2 Compiler . . . . . 100
7.2.1 Front-End for FORTRAN . . . . . 100
7.2.2 Dependence Analysis . . . . . 101
7.2.3 Alignment analysis . . . . . 102
7.2.4 Parallelizer . . . . . 102
7.2.5 Code generation . . . . . 103
7.3 Conclusions . . . . . 108

**8 Knowledge–Based Automatic Parallelization by Pattern Recognition**
*by Christoph W. Keßler*
8.1 Introduction and Overview . . . . . 110
8.2 Preprocessing the Source Code . . . . . 112
8.3 Which Patterns are Supported? . . . . . 115
8.4 Pattern Recognition: A Detailed View . . . . . 116
8.4.1 Program Representation . . . . . 117
8.4.2 Pattern Hierarchy Graph . . . . . 118
8.4.3 The Matching Algorithm . . . . . 119
8.4.4 Standard Pattern Matching: A simple example . . . . . 120
8.4.5 Removing redundant IF statements . . . . . 121
8.4.6 Loop Rerolling . . . . . 122
8.4.7 Difference Stars . . . . . 125
8.4.8 Beyond standard matching: Identification of multigrid hierarchies 126
8.5 A Parallel Algorithm for each Pattern . . . . . 127
8.6 Alignment and Partitioning . . . . . 128
8.7 Determining Cost Functions: Estimating and Benchmarking . . . . . 130
8.8 Implementation and Future Extensions . . . . . 130
8.9 Conclusions . . . . . 132

**9 Automatic Data Layout for Distributed–Memory Machines in the D Programming Environment** **136**
*by Ulrich Kremer, John Mellor-Crummey, Ken Kennedy, and Alan Carle*
9.1 Introduction . . . . . 136
9.2 Compilation system . . . . . 137
9.3 Dynamic Data Layout: Two Examples . . . . . 138
9.4 Towards Dynamic Data Layout . . . . . 142
9.4.1 Alignment Analysis . . . . . 142
9.4.2 Distribution Analysis . . . . . 143
9.4.3 Inter-Phase Decomposition Analysis . . . . . 143
9.5 Related Work . . . . . 146
9.6 Summary and Future Work . . . . . 147

**10 Subspace Optimizations** **153**
*by Kathleen Knobe and William J. Dally*
10.1 Introduction . . . . . 153
10.1.1 Data Optimization . . . . . 154
10.1.2 Shapes . . . . . 155

10.2 Subspaces . . . 157
10.3 Subspace Changes . . . 158
10.3.1 Scalars . . . 159
10.3.2 Control Expressions . . . 160
10.3.3 Array Sections . . . 161
10.3.4 Explicit Dimensions . . . 162
10.3.5 Reductions . . . 163
10.4 Subspace Optimizations . . . 164
10.4.1 Relative Costs . . . 165
10.4.2 Subspace Minimization . . . 166
10.4.3 Subspace Minimization with other Types of Expansion . . . 169
10.4.4 Combining Multiple Expansions . . . 170
10.4.5 Expansion Strength Reduction . . . 171
10.4.6 Expansion Costs . . . 172
10.4.7 Reducing the Computation within Expansions . . . 173
10.5 Subspaces Optimization Compared to Alignment . . . 174
10.6 Summary . . . 175
10.7 Acknowledgments . . . 175

**11 Data and Process Alignment in Modula-2* 177**
*by Michael Philippsen and Markus U. Mock*
11.1 Introduction . . . 177
11.2 Modula-2* . . . 178
11.2.1 FORALL statement . . . 179
11.2.2 Allocation of array data . . . 179
11.3 Alignment in Modula-2* . . . 180
11.3.1 Data Alignment . . . 181
11.3.2 Process Alignment . . . 182
11.4 Arrangement Graphs and Conflicts . . . 183
11.4.1 Type and Structure . . . 183
11.4.2 Conflicts . . . 184
11.5 Cost Considerations . . . 187
11.6 Example . . . 188
11.7 Conclusion . . . 189

**12 Automatic Parallelization for Distributed Memory Multiprocessors 192**
*by Anne Dierstein, Roman Hayer, and Thomas Rauber*
12.1 Introduction . . . 192
12.2 Related Work . . . 193
12.3 Overview . . . 194
12.4 Parallelization Strategy . . . 195
12.5 Branch–and–Bound Algorithm . . . 199
12.5.1 Basic Approach . . . 199
12.5.2 Distribution Graph . . . 201
12.5.3 Redistribution during Program Execution . . . 204
12.6 Performance Estimator . . . 205

12.6.1 Transfer costs . . . . . 206
12.6.2 Combining the transfer costs . . . . . 209
12.6.3 Data Transfer Graph . . . . . 210
12.7 Prototype Implementation and Results . . . . . 212
12.7.1 Implementation . . . . . 212
12.7.2 Livermore Loops . . . . . 212
12.7.3 Gauss–Seidel Relaxation . . . . . 213
12.7.4 Jacobi Relaxation . . . . . 214
12.8 Conclusions and Further Research . . . . . 215
12.9 Acknowledgements . . . . . 216

**A Trademarks** **218**