Theory and Practice in System-Level Design of Application-Specific Heterogeneous Multiprocessors

Yosef Gavriel Tirat-Gefen

CENG 97-29

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213-740-4476)
August 1997

THEORY AND PRACTICE IN SYSTEM-LEVEL DESIGN OF APPLICATION-SPECIFIC HETEROGENEOUS MULTIPROCESSORS

by

Yosef Gavriel Tirat-Gefen*

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Engineering)

August 1997

Copyright 1997 Yosef Gavriel Tirat-Gefen*

*also known as Jose' Carlos de Souza Batista

Dedication

To my parents, my sisters, my Rabbi and my homeland - Israel.

Acknowledgements

First of all. I would like to thank my academic advisor, Dr. Alice C. Parker for her support and guidance during the preparation of this thesis. Her insights were very helpful during the development of key parts of the present work.

I would like to thank Dr. Peter Beerel for the many hours of discussion while giving me a ride back home as well as during our after business hours informal meetings. I would like to thank Dr. James Moore, Dr. Melvin Breuer and Dr. Victor Prasanna for being in my guidance committee.

I thank my family. my parents and my sisters for their support. love and patience.

I thank the guidance of Rabbi Harry Greenspan. my rabbi over the long years of my stay in Los Angeles. as well as the staffs of the Rabbinical Council of California (RCC). the Yeshiva of Los Angeles (YOLA), USC Hillel Center and the Jewish Federation of Los Angeles for all their help. I thank also Rabbi David Shapiro (YOLA), Rabbi Abraham Union (RCC). Rabbi Ytzack Adlerstein (YOLA) . Rabbi Naum Sauer (YOLA). Rabbi Yona Vogel (Yeshiva Machon Daniel in Jerusalem) and Mr. Dovid Abramson (YOLA).

I thank the State of Israel. the Jewish Agency. the Israeli Defense Forces (IDF) and the staff the Israeli Consulates of Los Angeles, New York and Boston as well as the Canadian Consulate General in Los Angeles. the American Embassy in Israel and the Brazilian Consulate in Los Angeles for their assistance during the last years of my doctoral research.

I thank my workmates in the Design Automation Research Group: Mr. Diogenes C. Silva. Dr. C. P. Ravikumar. Mr. Arani Sinha. Mr. Yishing Chang. Mr. Dong-Hyun Heo. Mr. Suhrid Wadekar. Mr. Sami Habib. Dr. Pravil Gupta. Dr. Chih-Tung Chen and Dr. Shiv Prakash for their assistance and friendship.

I thank Mrs. Dianne Demmetras, Mrs. Mary Zittercob. Mrs. Regina Norton, Mr. Tim Boston and other members of the staff of the Department of Electrical

Engineering-Systems as well as the staff the Office of International Students and Scholars (OISS), the Office of Residential and Greek Life and many other departments of University of Southern California for facilitating my academic life during my stay in USC.

I thank my workmates Ken Steele, Andrew Guyler, Ellison Cohen, Beck Gerlach and other members of the staff of the Silicon Systems Division (SSD) as well as Jeff Beadles of the Support Engineering Division of Mentor Graphics Corporation.

Finally, I acknowledge the financial support provided by the Brazilian Council for Research and Development (CNPQ - Conselho Nacional de Pesquisa e Desenvolvimento) under contract no. 202302/90.3, by the Advanced Research Projects Agency (ARPA) under contract no. 53-4503-9319 and monitored by the Federal Bureau of Investigation (FBI) under contract no. J-FBI-94-161, and by the National Science Foundation (NSF) under grant no. MIP9023979.

The views and conclusions contained in this thesis are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency (ARPA), National Sciences Foundation (NSF), the U.S. Government, the Government of the State of Israel, the Government of Federal Republic of Brazil, the Canadian Government or Mentor Graphics Corporation.

Contents

Dedication ii

Acknowledgements iii

List Of Figures xii

List Of Tables xiv

Abstract xv

1 Introduction 1

1.1 Motivation 1

1.2 Overview of Mathematical Models for System-Level Design 3

1.2.1 Mixed Integer Linear Programming Models for System-Level Design 4

1.2.1.1 Pure Data-Flow Graph Models 7

1.2.1.2 MILP Based Tool Sets for System Level Design . . . 7

1.2.1.3 Naming Convention 9

1.2.1.4 Time Representation 9

1.2.1.5 Timing Variables 9

1.2.1.6 System Latency 10

1.2.1.7 Binary Variables 10

1.2.1.7.1 Task Allocation 10

1.2.1.7.2 Processor Selection 10

1.2.1.8 Parallelism in a Task-Flow Graph 10

1.2.1.9 Task Scheduling 11

1.2.1.10 Modeling Data Transfers 11

1.2.1.10.1 Data Transfer Allocation 11

1.2.1.10.2 Data Transfer Scheduling 12

1.2.1.11 Relative Scheduling 12

1.2.1.12 Performance and Cost Information 12

1.2.1.13 Execution Time of a Task 13

1.2.1.14 Communication Time of a Data Transfer 13

1.2.1.15 Representing a Non-pure Data-Flow Model 13
1.2.1.16 Linearization Techniques 14
1.2.1.17 Objective Function 16
1.2.2 Modeling Uncertainty 16
1.3 Thesis Organization 17

2 Related Work **19**
2.1 MILP Models 19
2.2 Macro-Pipelined Design 20
2.2.1 Non-Preemptive Mode of Execution 20
2.2.1.1 Lower Bound on the Initiation Interval 20
2.2.1.2 Full Static Scheduling 20
2.2.1.3 Cyclic Static Scheduling 21
2.2.1.4 Graph Theory Applied to Macro-Pipelined Design . . 22
2.2.1.5 Retiming, Unfolding and Rotation 22
2.2.1.6 Functional Pipeline 22
2.2.1.7 Multidimensional Periodic Multiprocessors 22
2.2.2 Preemptive Mode of Execution 23
2.3 Soft Computing Methods - Genetic Algorithms 23
2.4 Imprecise Computation 24
2.5 Probabilistic Models and Stochastic Simulation 24
2.6 Performance Bounds Theory 26
2.7 Summary 27

3 Macro-Pipelined System Level Design **28**
3.1 Motivation 28
3.2 Definitions 29
3.2.1 Instance of Task (Data Transfer) 29
3.2.2 Full Static and Cyclic Static Allocation 29
3.2.3 Preemptive Execution 30
3.2.4 Static Scheduling 31
3.2.5 Initiation Interval, System Latency and Parallelism Degree . . 31
3.2.5.1 Finite and Infinite Horizons 32
3.2.5.1.1 Infinite Horizon 32
3.2.5.1.2 Finite Horizon 32
3.2.6 Overlapping Factor 33
3.2.7 Task-flow Graph Transformations 34
3.2.7.1 Unfolding 35
3.2.7.2 Retiming 35
3.3 A MILP Model for Macro-Pipelined Design with Finite Horizon . . . 37
3.3.1 Unrolling Factor 37
3.3.2 Synchronism between Iterations 37
3.3.3 Overlap Detection Windows 38

3.3.3.1 Iteration and Time-Invariance 38
3.3.3.2 Task Overlap Detection Window 39
3.3.3.3 Data Transfer Overlap Detection Window 39
3.3.3.4 Sufficiency of the Overlap Detection Windows 41
3.3.3.5 Independence between Overlap Detection Windows . 42
3.3.4 Binary Variables 42
3.3.4.1 Redefinition of Type α Variables 42
3.3.4.2 Redefinition of Type Φ Variables 42
3.3.4.3 Properties of $\alpha_{i\,j}[a,\ b]$ and $\Phi_{i\,j}[a,\ b]$ 43
3.3.4.3.1 $\alpha_{i\,j}[a,\ b]$ 43
3.3.4.3.2 $\Phi_{i\,j}[a,\ b]$ 45
3.3.5 Constants and Constraints of the MILP Model 47
3.3.5.1 Pruning Unnecessary Constrains 47
3.3.6 Summary of the MILP Model for Macro-Pipelined design . . . 47
3.3.6.1 Tasks Overlap Avoidance Constraints 48
3.3.6.2 Data-Transfer Overlap Avoidance Constraints 49
3.3.6.3 Additional Constraints to Ensure Numerical Convergence of the MILP Model 49
3.3.7 Software Implementation 50
3.4 Retiming and System-Level Design 50
3.4.1 Motivation 50
3.4.2 Retiming as a Task-Flow Graph Transformation 50
3.4.3 Mathematical Model 51
3.4.3.1 The Function $d : S_i \longrightarrow \mathcal{R}$ - Computation Time of a Task 51
3.4.3.2 Leiserson and Saxe's Retiming Theorem 52
3.4.3.3 An ILP Model for Simultaneous Retiming and Processor Selection 53
3.4.3.3.1 Edge Weights after Retiming 53
3.4.3.3.2 Computation Time of a Task 53
3.4.3.3.3 Cost of a Task to Processor Assignment . . 54
3.4.3.3.4 Functions D_p and W_p 54
3.4.3.3.5 Circular Walk of a TFG 54
3.4.3.3.6 Fundamental Cycles of a TFG 55
3.4.3.3.7 Linearization of the Model 56
3.4.3.3.8 Summary of the Model 59
3.4.4 Software Implementation 60
3.5 Experimental Results 60
3.5.1 Experiments with *pipelined-SOS* 60
3.5.2 Experiments with *retime* 61
3.6 Summary of the Chapter 67

4 **System of Difference Constraints and System-Level Synthesis** **68**
4.1 Motivation . . . 68
4.2 Timing Constraint Graph G_{constr} . . . 69
4.2.1 Solving the Constraint Graph G_{constr} . . . 70
4.3 Special cases . . . 70
4.3.1 Non-Periodic Case . . . 71
4.3.1.1 The Source Node of G_{constr} . . . 71
4.3.1.2 The Relaxed Graph G^{relax}_{constr} . . . 72
4.3.1.3 An Algorithm for Solving G^{relax}_{constr} . . . 73
4.3.2 Macro-Pipelined Case . . . 74
4.3.2.1 Feasibility Interval of T_I . . . 75
4.3.2.1.1 Upper and lower Bounds for a Cyclic G_{constr} 76
4.3.2.2 The Source Node of G_{constr} . . . 77
4.3.2.3 The Relaxed Graph G^{relax}_{constr} . . . 78
4.3.2.4 An Algorithm for Solving G^{relax}_{constr} . . . 78
4.3.2.5 Tradeoff T_I versus T_{SYS} . . . 79
4.4 Applications . . . 79
4.5 Summary of the Chapter . . . 79

5 **Genetic Algorithms Applied to System-Level Design** **80**
5.1 Motivation . . . 80
5.1.1 Drawbacks of Using MILP Solvers . . . 80
5.1.2 Inadequacy of Nonlinear Programming Methods . . . 81
5.1.3 Probabilistic Optimization . . . 81
5.1.4 GAs and Pure *Branch-and-Bound* Optimization . . . 82
5.1.5 Parallelism of Genetic Algorithms . . . 82
5.2 Overview of Genetic Algorithms . . . 82
5.2.1 A Basic Template of a Genetic Algorithm for System-Level Design . . . 83
5.2.2 Chromosome Representation . . . 84
5.2.3 Mutation and Crossover Operators . . . 84
5.2.4 Genetic Algorithms and Random Search Methods . . . 85
5.2.5 Initial Solutions . . . 85
5.2.6 Measuring the Diversity of a Population . . . 85
5.2.7 Variations of the Basic GA Template for System-Level Design 87
5.2.7.1 Selection Schemes . . . 87
5.2.7.1.1 Proportionate Reproduction . . . 88
5.2.7.1.2 Tournament Selection . . . 88
5.2.7.1.3 Selection by Simulated Annealing . . . 88
5.2.7.2 Scaling of the Fitness Function . . . 89
5.2.7.3 Avoidance of Replicated Solutions . . . 90
5.2.7.4 Variable Size Population . . . 90
5.3 Summary of the Chapter . . . 91

6 **The MEGA System** **92**
6.1 Covered Computational Paradigms 92
6.2 Supported Interconnection Networks 93
6.3 Task-Flow Graph Representation 93
6.3.1 Data-Transfer Types 94
6.3.2 Dummy Tasks and Twin Tasks 94
6.3.3 Broadcast. Multicast and Dummy-Twin Data Transfers 94
6.4 Input Data 95
6.4.1 Processor Types Library 95
6.4.2 Bus Types Library 95
6.4.3 Task Flow Graph 96
6.4.3.1 Tasks 96
6.4.3.2 Data Transfers 96
6.4.3.3 Twin Tasks 97
6.4.4 Performance/Cost Constraints 97
6.4.5 Topological Constraints 98
6.4.6 Genetic Algorithm Parameters 99
6.5 Output Data 99
6.6 Main Data Structures 99
6.6.1 Processor Types Table 100
6.6.2 Processor Table 100
6.6.3 Bus Types Table 100
6.6.4 Bus Table 101
6.6.5 Task Table 101
6.6.6 Data-Transfer Table 101
6.6.7 Chromosome Representation in MEGA 102
6.6.7.1 Allocation 103
6.6.7.2 Relative Scheduling 104
6.6.7.3 Timing 104
6.7 Fitness Evaluation 105
6.7.1 Fitness Scaling by Moving Window 105
6.7.2 Normalized Fitness 106
6.8 Selection Schemes 106
6.9 Transitive Closure and Parallelism Detection 107
6.9.1 Macro-Pipelined (Finite Horizon) Case 107
6.9.1.1 Bounded Unrolled Graph 108
6.9.1.2 Twin Tasks and Iteration-Twin Data Transfers . . . 110
6.10 Fast Topologic Ordering 111
6.11 Detecting Cycles 116
6.12 ASAP-ALAP Based Scheduling Heuristics 117
6.12.1 Macro-Pipelined Finite Horizon Case 120
6.13 Detailed Timing Information and Bellman-Ford Algorithm 121

6.13.1 Non-Periodic Case . . . 121
6.13.2 Macro-Pipelined Finite Horizon Case . . . 122
6.14 Profile Functions . . . 123
6.15 Generating the Initial Population . . . 123
6.16 Genetic Operators . . . 126
6.16.1 Mutation . . . 126
6.16.1.1 θ and ϑ Genes . . . 126
6.16.1.2 π and ϖ Genes . . . 127
6.16.1.3 Gen_α and Gen_Φ Sets . . . 128
6.16.1.4 Probabilistic Profile in Mutation . . . 128
6.16.1.5 Feasibility after Mutation . . . 129
6.16.2 Performing Crossover . . . 129
6.16.2.1 Tasks Crossover . . . 130
6.16.2.2 Data-Transfer Crossover . . . 131
6.16.2.3 Feasibility of Offsprings . . . 132
6.16.2.4 Probabilistic Profile . . . 132
6.16.2.5 Crossing Index . . . 132
6.17 Managing Infeasible Solutions . . . 133
6.18 Managing Memory . . . 133
6.19 An Overview of the MEGA Algorithm . . . 133
6.20 Experimental Results . . . 134
6.20.1 Comparing Different Selection Schemes . . . 138
6.21 Summary of the Chapter . . . 138

7 Imprecise Computation **140**
7.1 Motivation . . . 140
7.2 Mathematical Formulation . . . 141
7.3 An MILP Formulation . . . 142
7.4 Allowing Imprecise Computation in MEGA . . . 144
7.4.1 Utilization Factors as Genes . . . 144
7.4.1.1 Trap Function . . . 145
7.4.2 Redefining Mutation in MEGA . . . 145
7.4.2.1 Probabilistic Profile . . . 145
7.4.3 Crossover . . . 146
7.4.4 Fitness Value . . . 146
7.5 Improving the Quality by Postprocessing . . . 147
7.5.1 An Overview of MEGA with Imprecise Computation . . . 148
7.6 Experimental Results . . . 149
7.7 Summary of the Chapter . . . 154

8 Probabilistic Approaches for System-Level Design 155
8.1 Motivation . . . 155
8.2 A Precise Mathematical Model for Handling Uncertainty . . . 156
8.2.1 Non-Pipelined Case . . . 157
8.2.2 Pipelined Case . . . 159
8.2.3 Difficulties of Using an Exact Formulation . . . 160
8.3 A Stochastic Simulation Based Approach . . . 161
8.3.1 Definitions . . . 161
8.3.1.1 Generator of a Random Variable . . . 161
8.3.1.2 Multidimensional Sampling . . . 162
8.3.1.3 Probabilistic Fitness Values in MEGA . . . 162
8.3.2 A Basic Template for Monte Carlo Simulation in MEGA . . . 163
8.3.3 Applying Variance Reduction Techniques . . . 163
8.3.3.1 A Stratified Sampling Heuristic . . . 164
8.3.3.2 Using Antithetic Sampling in MEGA . . . 166
8.3.3.2.1 Allowing Antithetic Sampling in MEGA . . 166
8.4 Experimental Results . . . 167
8.5 Summary . . . 170

9 Fast prediction in system-level design 174
9.1 Motivation . . . 174
9.2 Fundamentals . . . 174
9.3 A survey in performance bounds theory . . . 175
9.4 Generating a lower bound surface . . . 179
9.4.1 Non-inferior solutions . . . 179
9.4.2 Non-periodic case . . . 180
9.4.2.1 Macro-pipelined case . . . 180
9.4.3 Genetic generation of trade-off surfaces . . . 180
9.5 Experimental Results . . . 181
9.6 Summary of the chapter . . . 184

10 Conclusion and Future Research 185
10.1 Contribution . . . 185
10.2 Future Research . . . 187
10.2.1 Multidimensional Periodic Scheduling Problems . . . 187
10.2.2 Evolutionary Hardware . . . 187
10.2.3 Logic Design. Computer Arithmetic and High-Level Design . . 188
10.2.4 Performance Bounds for Application Specific Multiprocessors 188

Appendix A
Overview of multi-variable probability theory . . . 189

List Of Figures

1.1 System-level synthesis of application-specific heterogeneous multiprocessors . . . 1
1.2 Periodic task-flow graphs . . . 5
1.3 Representation of data dependencies in periodic task-flow graphs . . . 6
1.4 Comparing periodic and non-periodic task-flow graphs . . . 7
1.5 Overview of a typical MILP-based tool set for system-level design - the SOS system. . . . 8
1.6 Timing variables . . . 9

3.1 Overlapping factor, iteration latency and initiation interval . . . 33
3.2 A behavior preserving transformation . . . 34
3.3 Applying retiming to a TFG . . . 36
3.4 Task execution overlap detection window . . . 39
3.5 Data transfer overlap detection window . . . 39
3.6 Independence between α variables . . . 45
3.7 Spanning tree of a TFG and fundamental cycles . . . 57
3.8 Task-flow graph . . . 61
3.9 Gannt Chart for Designs 3, 5, 7 and 9 . . . 63
3.10 Task-flow graph before retiming . . . 65
3.11 Transformed TFG after retiming - Design 1, 2, 3, 6, 7, 8, 9 and 10 . . . 65
3.12 Transformed TFG after retiming - Designs 4 and 5 . . . 67

4.1 Timing Constraint graph . . . 69

6.1 Unfolded graph G_α . . . 112
6.2 Synchronous unfolded graph G_β . . . 113
6.3 Myopic synchronous unfolded graph G_γ . . . 114
6.4 Bounded unrolled graph G_δ . . . 115
6.5 Task flow graphs . . . 135
6.6 Gannt Chart for Design I . . . 137
6.7 Best-solution fitness convergence for different selection schemes . . . 139
6.8 Worst-solution fitness convergence for different selection schemes . . . 139

7.1 Modeling non-preemptive imprecise computation . . . 141

7.2 Task flow graphs . . . 149
7.3 Effect of the trap length z (TFG A) . . . 150
7.4 Effect of the trap length z (TFG B) . . . 152
7.5 Trade-off points for the task flow graphs . . . 152

8.1 Task flow graphs . . . 167
8.2 System latency (T_{SYS}) of Design VII . . . 169
8.3 Effect of variance reduction techniques (Design VII) . . . 171
8.4 Gannt chart for Design VII . . . 172

9.1 Task flow graphs . . . 182
9.2 Buses types - cost and performance information . . . 183
9.3 TFG A - Cost and Latency trade-off curve . . . 183
9.4 TFG B - Cost and Latency trade-off curve . . . 184

List Of Tables

3.1 Processor types - cost and performance information 61
3.2 Solutions - Cost and Performance Trade-off 62
3.3 Processor types - cost and performance information 64
3.4 Solutions - Cost and Performance Trade-off 66

6.1 Processor costs and task execution times - Task Flow Graph A 135
6.2 Processor cost and task execution times - Task Flow Graph B 135
6.3 Solutions - Cost and Latency trade-off 136

7.1 Processor types - cost and performance information for TFG A 150
7.2 Processor types - cost and performance information for TFG B 151
7.3 Buses types - cost and performance information 151
7.4 Selected Designs 153

8.1 Processor types - cost and performance information for TFG A 167
8.2 Processor types - cost and performance information for TFG B 168
8.3 Buses types - cost and performance information 168
8.4 Selected Designs - Probabilistic Trade offs for Latency (T_{SYS}) and Cost 170

9.1 Processor costs and task execution times - Task Flow Graph A 182
9.2 Processor cost and task execution times - Task Flow Graph B 182

Abstract

Application-specific multiprocessors are becoming an important implementation style for integrated systems. This dissertation introduces a formulation for optimal system-level design of application-specific heterogeneous multiprocessors (ASHMs) executing in a macro-pipelined (periodic) fashion. A mixed-integer linear programming (MILP) model allowing simultaneous task mapping, processor selection and retiming when designing heterogeneous multiprocessors is also described.

Another main contribution of this dissertation is a set of new heuristic methods using genetic algorithms and particular properties of the MILP formulations for the ASHM design problem, which is implemented in a software system called MEGA. These methods are able to achieve optimal/near-optimal designs with a lower computational cost than by exact optimization of MILP models.

A new genetic algorithm formulation for ASHM design allowing *imprecise computation*, another possible approach for system-level design, allowing a *fine grain* trade-off between performance and cost, is presented and incorporated into the tools in the MEGA system.

The system-level design of application-specific multiprocessors relies heavily on the availability of cost/performance parameters supplied by the designer or by tools for computer aided design such as chip-level performance/cost predictors. These parameters can have some associated uncertainty, e.g. they can be expressed as random variables. In this thesis, mathematical formulations for optimization in presence of probabilistic cost/performance parameters as well as soft computing methods to solve them, i.e. genetic algorithms, are proposed and implemented in the framework of the MEGA tool set.

A brief survey on analytic performance bound prediction along with a set of fast methods for *performance and cost trade-off curve* prediction for periodic and non-periodic cases based on algorithms developed for the MEGA tool set are presented.

Chapter 1

Introduction

1.1 Motivation

This thesis presents a set of algorithms and heuristics for solving problems in system-level design, i.e., the design of multiprocessors having a heterogeneous mix of off-the-shelf and custom designed processors, where the inter-processor communication costs are not assumed to be negligible and the performance and cost figures of each processor, as well as of the interconnection network, are parameters supplied by the designer and/or processor (chip) design tools. These parameters are assumed to be either deterministic numbers or random variables. Emphasis is given to the design of application-specific multiprocessors in the sense that the multiprocessor is optimized for the execution of just one application, but could be used to execute other applications whose tasks can be executed by one or more of the processors.

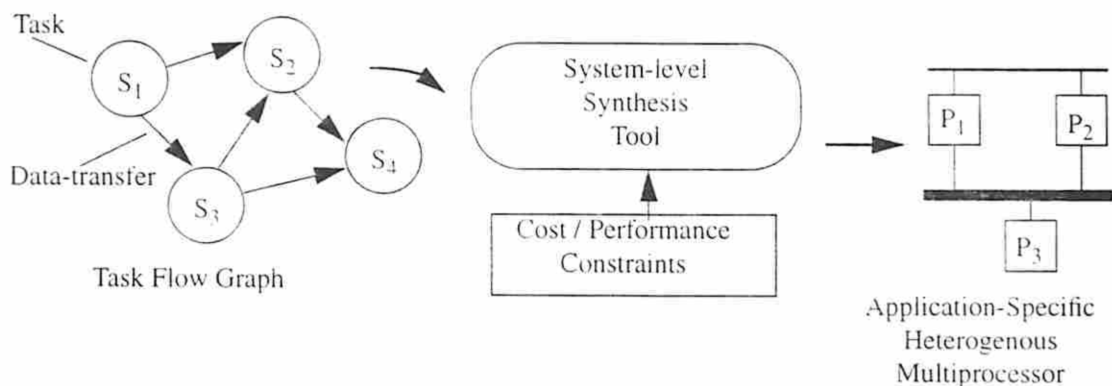


Figure 1.1: System-level synthesis of application-specific heterogeneous multiprocessors

There is a demand for application-specific heterogeneous multiprocessors in such areas as multimedia, high definition television (HDTV) and high-performance digital signal processing, which makes tools for these types of multiprocessors important for systems integration. These kinds of applications can be specified at the system level in terms of task-flow graphs (TFGs) such as the example graph in Figure 1.1. The tasks in the TFG are subsystem specifications of different complexities which possess particular properties. Heterogeneity is a common feature of such a system-level description. In general, to achieve highest performance each task is best mapped to a different kind of processor. However cost constraints force the sharing of processors and communication links by different tasks.

The use of high-level chip synthesis tools as well as other powerful CAD tools [14] [46] allows the easy development of processors of different costs and speeds for each task or subset of tasks of a task flow. The availability of libraries of custom and off-the-shelf components increases the number of available choices. The possibility of using new packaging technologies such multi-chip modules (MCM), and the increased density of integrated circuit technology (e.g. sub-micron) as well advances in printed circuit board manufacturing allow both the mapping of high performance computing applications, such as the ones mentioned above, to custom multiprocessors and also the synthesis of such multiprocessors at a reasonable cost.

Optimality (or near-optimality) in terms of performance and cost is usually desired in application-specific implementations, but the numbers of combinations of processors, interconnection networks and possible task schedules and allocations are far too great to be handled by a human designer in the current short-duration design cycle, where different cost/performance trade-offs should be examined in parallel by the designer.

There is a need for system-level synthesis which are based on theory such as performance bounds, mathematical models for synthesis of heterogeneous multiprocessors, and formal techniques for optimal synthesis along with computationally-inexpensive heuristics. Performance bounds allow the evaluation of the size of the search space and the expected quality of a solution (design) as well as support the avoidance of searching infeasible implementations. Performance bounds are particularly important for bound-based search methods such as branch and bound and A* [108] . Mathematical models are the backbone for a good synthesis tool. The type of

model will determine the expected computational cost, quality of the designs to be found by the tool and the level of detail to be handled. Synthesis tools guaranteeing optimal designs are necessarily based on formal models, allowing proof of optimality of the synthesis algorithm. Computationally-inexpensive heuristics are needed when optimal methods become prohibitive in terms of computational cost.

A system-level design tool is highly dependent on cost/performance parameters supplied by the designer or by other tools such as chip performance/cost predictors [67] [68] [69]. Often these parameters are just estimates, having some degree of uncertainty associated with them. A state-of-the-art computer tool for system-level design should also be able to handle uncertainty. The deterministic case where the parameters are assumed to not have associated uncertainty is by definition a simplification to avoid the complex realistic situation where some parameters are better described as random variables.

The *imprecise computation* model assumes that each task is composed of a *mandatory* subtask followed by an *optional* one. A task is said to have a *precise* result if it is allowed to compute till its end. The term *imprecise computation* means that some tasks may not have their optional subparts executed till completion due to constraints such as maximum implementation cost and hard deadlines. The *imprecise execution* of the optional parts will decrease the *quality* of the output data being produced by the multiprocessor. However an implementation with lower *output data quality* might have a lower implementation cost. The *imprecise computation* model allows a fine-grain trade-off between *quality* and cost that might be helpful to the system designer in some applications.

1.2 Overview of Mathematical Models for System-Level Design

Most of the mathematical models used in system-level design model the application to be implemented as a *task-flow graph* (TFG), where the *nodes* (vertices) correspond to computational tasks and the *arcs* (edges) represent data being exchanged between tasks, i.e. each *data transfer* corresponds to an *data dependency* between two tasks.

The *mode of execution* of an application can be characterized as *non-periodic* or *periodic* (*macro-pipelined*).

In the *macro-pipelined* mode of execution, the application can be modeled as a loop with an infinite number of iterations (Figure 1.2). The tasks in the task-flow graph are executed once in each iteration. *Inter-iteration* data dependencies (Figure 1.3) are modeled by giving weights (delays) for each arc/edge of the TFG. For a data-dependency such as $S_s[i] \longleftarrow S_r[i-k]$, where $S_s[i]$ denotes the instance of task S_s executed in *iteration* i, the weight of edge e corresponding to this data-dependency is given by the *difference of subscripts* between S_s and S_r.

$$w(e) = \; difference \;\; of \;\; subscripts \;\; = k \tag{1.1}$$

Therefore all edges corresponding to intra-iteration data dependencies have weight equal to *zero*, i.e. the difference of subscripts is equal to zero.

The *non-periodic* mode of execution can be understood as a particular case of the periodic case, where an iteration is only started after the previous one has finished. There is no overlap between iterations.

Due to the need to preserve *causality*, a TFG corresponding to an application to be executed in a *non-periodic mode* is always a directed acyclic graph (DAG) as seen in Figure 1.4.

On the contrary, a *periodic* TFG may have cycles (Figure 1.4). Every cycle, however, has the sum of its edge weights greater than zero in order to preserve causality.

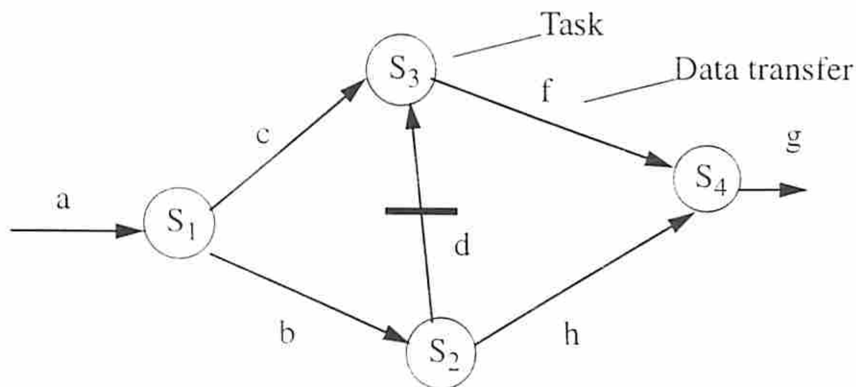
$$\sum_{\forall e \in cycle} w(e) > 0 \tag{1.2}$$

1.2.1 Mixed Integer Linear Programming Models for System-Level Design

The task-flow graph representation allows the development of straight-forward mixed integer linear programming models, which have in common a set of properties and

```
for  i = 0 to ∞  {
   S1(in a[i]; out b[i], c[i]);
   S2(in b[i]; out d[i], h[i]);
   S3(in c[i], d[i-1]; out f[i]);
   S4(in h[i], f[i]; out g[i]);
}
```

(a) Loop representation



Arc (edge) weights

$w(a) = w(c) = w(b) = w(f) = w(g) = w(h) = 0$

$w(d) = 1$

(b) Task Flow Graph Representation

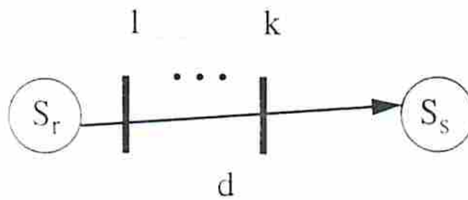
Figure 1.2: Periodic task-flow graphs

Loop representation

```
for  i = 0 to ∞  {
  ...
  Sr(in ... ; out d[i], ... );
  ...
  Ss(in ...., d[i-k]; out ...);
  ...
}
```

data dependency

Task Flow Graph representation



arc weight: **w(d) = k**

Figure 1.3: Representation of data dependencies in periodic task-flow graphs

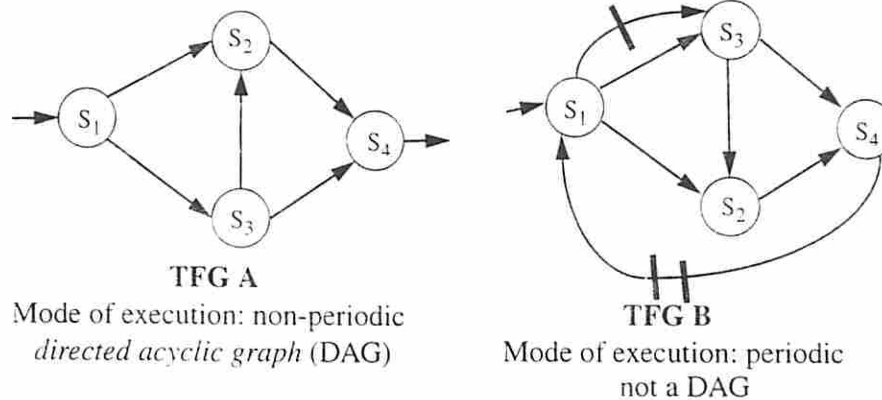


Figure 1.4: Comparing periodic and non-periodic task-flow graphs

variables which are outlined here. For sake of brevity, only the *non-periodic* non-preemptive case is discussed in this section [1.1]. The system-level design of *macro-pipelined* (periodic) application-specific multiprocessors is addressed in Chapter 3.

1.2.1.1 Pure Data-Flow Graph Models

Task-flow graphs can be understood as an generalization of *data-flow graphs* [45]. Data-flow graphs assume that a node will start computation only after all inputs (tokens) are available. The outputs of the node will be available only after the node computation is finished.

Task-flow graphs are less constrained in order to better represent real hardware/software implementations. A node (task) may be allowed to start computation without all inputs being available. Part of the output data may be available before the node computation finishes.

1.2.1.2 MILP Based Tool Sets for System Level Design

A typical tool set for system-level design of application specific multiprocessors using an MILP formulation is outlined in Figure 1.5. which outlines the SOS system [99] [22].

[1.1] Much of what is described in this section was introduced in the works of Prakash and Parker [99]. Hafer and Parker [51]. and Chu et al. [16]

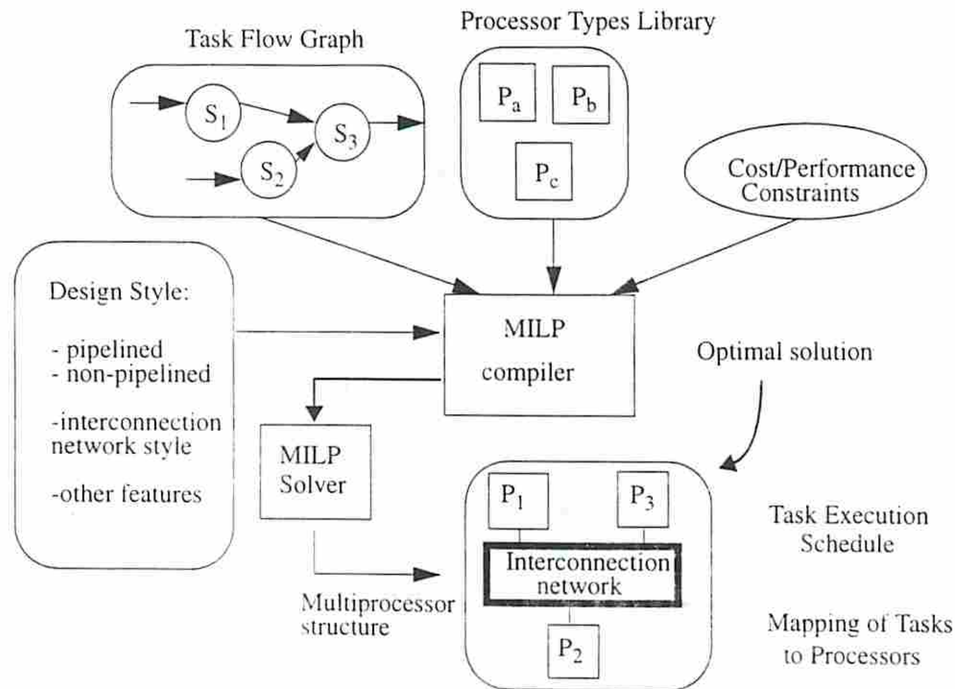


Figure 1.5: Overview of a typical MILP-based tool set for system-level design - the SOS system.

A dedicated compiler for MILP models accepts as inputs

i. the task-flow graph to be implemented in an application-specific multiprocessor,

ii. the *processor library* containing information about a set of custom and off-the-shelf processors that are available to be used in the multiprocessor to be implemented.

iii. cost and performance constraints for the design, and

iv. additional information such as mode of computation (periodic/non-periodic) and interconnection network topology, allowing the designer to control the architecture of the multiprocessor.

The MILP model compiler generates an MILP model of the problem to be solved by an MILP solver such as LINDO, [77] LAMPS [73] or BANZAI [50]. The solution found by the MILP solver will correspond to an optimal design, where the allocation and schedule of task and data transfers is completely specified.

1.2.1.3 Naming Convention

Different MILP models may adopt different naming conventions. However, they are intrinsically the same. The naming convention used in this thesis is similar to one used to describe the MILP model of the SOS set of tools [99] [100] [22].

1.2.1.4 Time Representation

Some MILP formulations assume that the *time events* occur at instant of time which are integer multiples of an *time quantum* δt (*integer grid*). Other models allow the time to be a continuous value, meaning that *time events* may happen in any possible instant of time (*continuous grid*).

1.2.1.5 Timing Variables

Independent of the particular definition of time used in an model, the following set of variables is commonly found in most MILP models for system-level synthesis of multiprocessors [16] [83] [99] executing in a *non-periodic* mode (Figure 1.6):

i. $T_{SS}(S_i)$: Time when task S_i starts execution.

ii. $T_{SE}(S_i)$: Time when task S_i ends execution.

iii. $T_{CS}(DT_k)$: Time when the data transfer DT_k from task S_i to task S_j starts.

iv. $T_{CE}(DT_k)$: Time when the data transfer DT_k from task S_i to task S_j ends.

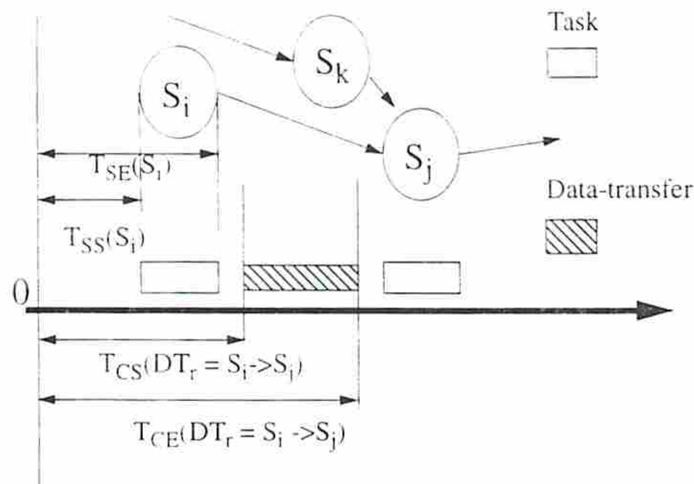


Figure 1.6: Timing variables

1.2.1.6 System Latency

The minimal time to execute a TFG for a given allocation and schedule of tasks and data transfers is defined to be the *system latency* T_{SYS}, which is equal to the minimum amount of time needed to all tasks complete.

$$\forall \; tasks S_x \;\; T_{SYS} \geq T_{SE}(S_x) \tag{1.3}$$

1.2.1.7 Binary Variables

A common characteristic of MILP models for system-level design and multiprocessor scheduling is to represent task and data transfer allocation and scheduling by means of a set of Boolean variables [16] [99] [100] [22].

1.2.1.7.1 Task Allocation

Task allocation is represented by a variable $\sigma_{i\,x}$ meaning that $\sigma_{i\,x}$ is *true* if and only if task S_i is allocated to processor p_x, otherwise $\sigma_{i\,x}$ is *false*, where p_x is a member of the *processor set* P_{S_i}, the set of processors to which task S_i can be allocated. For a given task S_i

$$\sum_{p_x \in P_{S_i}} \sigma_{i\,x} = 1 \tag{1.4}$$

1.2.1.7.2 Processor Selection

A processor p_x is selected if at least one task S_i is allocated to p_x. If p_x is selected then the Boolean variable β_{p_x} is *true*, otherwise it is *false*.

$$\beta_{p_x} = \bigvee_i \sigma_{i\,x} = \sigma_{i_1\,x} \vee \sigma_{i_2\,x} \vee \ldots \tag{1.5}$$

1.2.1.8 Parallelism in a Task-Flow Graph

A task S_a is parallel to a task S_b if and only if there is not a direct path from S_a to S_b or from S_b to S_a in the task-flow graph, i.e. the two tasks are data independent.

1.2.1.9 Task Scheduling

For each pair of parallel tasks S_i and S_j such that

$$P_{S_i} \cap P_{S_i} \neq \emptyset \tag{1.6}$$

where

$$P_{S_k} = \{set\ \ of\ \ processors\ \ able\ \ to\ \ execute\ \ S_k\}$$

a Boolean variable $\alpha_{i\,j}$ is defined. This variable is used in the constraints below, which enforce the *non-overlap* between the computation *time intervals* of two parallel tasks.

$$\forall p_x \in P_{S_i} \cap P_{S_j}$$

$$\sigma_{i\,x}\sigma_{j\,x}T_{SE}(S_i) \leq \alpha_{i\,j}\sigma_{i\,x}\sigma_{j\,x}T_{SS}(S_j) \tag{1.7}$$

$$\sigma_{i\,x}\sigma_{j\,x}T_{SE}(S_j) \leq (1 - \alpha_{i\,j})\sigma_{i\,x}\sigma_{j\,x}T_{SS}(S_i) \tag{1.8}$$

Note that if $\sigma_{i\,x}\sigma_{j\,x} = False$, $\alpha_{i\,j}$ may be set to either $False$ or $True$.

1.2.1.10 Modeling Data Transfers

Given a pair of tasks S_i and S_j, such that there is a *data transfer* from S_i to S_j ($S_i \longrightarrow S_j$), the variable $\gamma_{i\,j}$ indicates if this data transfer is *local* (S_i and same processor) or *remote* (data transfer between different processors).

$$\gamma_{i\,j} = \sum_{p_x \in P_{S_i} \cap P_{S_j}} \sigma_{i\,x}\sigma_{j\,x} \tag{1.9}$$

$$\gamma_{i\,j} = True \Longleftrightarrow \quad Data\ tranfer\ \ is\ \ local \tag{1.10}$$

$$\gamma_{i\,j} = False \Longleftrightarrow \quad Data\ tranfer\ \ is\ \ remote \tag{1.11}$$

1.2.1.10.1 Data Transfer Allocation

It is assumed that a set of *communication links* is available to allow exchange between tasks allocated to different processors. In a similar way as for a task, the variable $\eta_{r\,l}$ denotes the allocation of *non-local* data transfer $DT_r = S_i \longrightarrow S_j$ to *communication link* l. The variable $\eta_{r\,l}$, introduced in this dissertation, allows the use of a multiple

shared buses instead of just one bus or point to point lines as in the MILP models of Prakash and Parker [99], which is not present in the work of Chu et al. [16] either.

$$\sum_{l} \eta_{r\,l} = \gamma_{i\,j} \tag{1.12}$$

Communication links can be either shared buses or dedicated *point-to-point* connections between processors.

1.2.1.10.2 Data Transfer Scheduling

For each pair of *parallel* data transfers $DT_i = S_a \longrightarrow S_b$ and $DT_j = S_c \longrightarrow S_d$ such that they may be allocated to the same *communication link*, there is a Boolean variable $\phi_{i\,j}$ used in used in the following constrains enforcing non-overlap of the *communication time interval* [1.2].

$$\eta_{i\,l}\eta_{j\,l}\gamma_{a\,b}\gamma_{c\,d}T_{CE}(DT_i) \leq \phi_{i\,j}\eta_{i\,l}\eta_{j\,l}\gamma_{a\,b}\gamma_{c\,d}T_{CS}(DT_j) \tag{1.13}$$

$$\eta_{i\,l}\eta_{j\,l}\gamma_{a\,b}\gamma_{c\,d}T_{CE}(DT_j) \leq (1 - \phi_{i\,j})\eta_{i\,l}\eta_{j\,l}\gamma_{a\,b}\gamma_{c\,d}T_{CS}(DT_i) \tag{1.14}$$

1.2.1.11 Relative Scheduling

A valid Boolean assignment of the variable types α and type-ϕ [1.3] corresponds to a set of *feasible* schedules for the tasks and data transfers in the TFG. These schedules have in common the property that the tasks and data transfers are all performed in the same *relative* order. The values of the *timing variables* must be consistent with the *relative scheduling* implied by the Boolean variables.

1.2.1.12 Performance and Cost Information

In order to find the optimal solution for an MILP model for system-level design, a set of parameters must be provided by the designer:

$\omega_{i\,x}$ is the execution time of task S_i on processor p_x.

[1.2] Non-preemptive communication is assumed for non-local data transfers.
[1.3] A valid assignment is one that respect all constraints of the MILP model.

V_{DT_i} is the volume of data being transferred by data transfer DT_i,

D_{CL_x} is the *communication delay* per unit of data for a *local* data transfer between tasks allocated to processor p_x,

D_{CR_l} is the *communication delay* per unit of data for a data transfer allocated to communication link l,

PC_x is the cost of processor p_x and

LC_l is the cost of communication link l.

1.2.1.13 Execution Time of a Task

The *time interval* $[T_{SS}(S_i), T_{SE}(S_i)]$ is constrained by

$$T_{SE}(S_i) = T_{SS}(S_i) + \sum_{p_x \in P_{S_i}} \omega_{i\,x}\sigma_{i\,x} \tag{1.15}$$

1.2.1.14 Communication Time of a Data Transfer

In a similar way for a *data transfer*

$$T_{CE}(DT_i = S_r \longrightarrow S_s) = T_{CS}(DT_i) + Delay(DT_i) \tag{1.16}$$

$$Delay(DT_i) = V_{DT_i}(\gamma_{r\,s} \sum_{p_x \in P_{S_i}} D_{CL_x}\sigma_{r\,x}\sigma_{s\,x} + (1 - \gamma_{r\,s}) \sum_{l} D_{CR_l}\eta_{i\,l}) \tag{1.17}$$

1.2.1.15 Representing a Non-pure Data-Flow Model

In order to allow the use of a task-flow graph representation not limited by the data-flow model constraints of *input data dependency* and *output availability* the following set of parameters must be provided for each task S_i with inputs $IN(S_i) = \{in_1^i, in_2^i, \ldots\}$ and outputs $OUT(S_i) = \{out_1^i, out_2^i, \ldots\}$

$$R_{IN(S_i)} = \{r_{in_1}^i, r_{in_2}^i, \ldots\} \tag{1.18}$$

$$R_{OUT(S_i)} = \{r_{out_1}^i, r_{out_2}^i, \ldots\} \tag{1.19}$$

Such that

$$0 \leq r^i_{in_k} \leq 1 \; ; \; r^i_{in_k} \; is \; a \; real \; number \tag{1.20}$$

$$0 \leq r^i_{out_k} \leq 1 \; ; \; r^i_{out_k} \; is \; a \; real \; number \tag{1.21}$$

$r^i_{in_k}$ is the percent of processing, that task S_i is allowed to execute before input in^i_k is required to be available.

$1 - r^i_{out_k}$ is the percent of processing, that task S_i has to perform in order to generate output out^i_k.

These parameters are used in the following set of constraints:

$$T_{CE}(DT^i_{in_k}) \leq T_{SS}(S_i) + r^i_{in_k} \sum_{p_x \in P_{S_i}} \omega_{i\,x} \sigma i\,x \tag{1.22}$$

$$T_{CS}(DT^i_{out_k}) \geq T_{SE}(S_i) - r^i_{out_k} \sum_{p_x \in P_{S_i}} \omega_{i\,x} \sigma i\,x \tag{1.23}$$

Where DT_{in_k} DT_{out_k} is the data transfer associated with input in^i_k (output out^i_k) of task S_i.
For a pure data-flow graph computation model.

$\forall in^i_k \;\; r^i_{in_k} = 0$ if execution of task S_i starts only after inputs are available

$\forall out^i_k \;\; r^i_{out_k} = 0$ if outputs are available only after task S_i completes.

The parameters $r^i_{in_k}$ and $r^i_{out_k}$ are assumed to *processor-independent* constants defined by the particular algorithm to be implemented by task S_i.

1.2.1.16 Linearization Techniques

Some of the constraints in a typical MILP model for system design have to be linearized. Among the most common nonlinearities are the following [1.4]:

[1.4] In the following linearizations the logical value *False* (*True*) is expressed by the integer value 0 (1).

i. Logical AND

$$z = x_1 x_2 \dots x_n \;;\; x_1, \dots, x_n \in 0, 1 \tag{1.24}$$

is linearized to

$$\begin{aligned} z &\le x_1 \\ &\vdots \\ z &\le x_n \\ z &\ge \textstyle\sum_i x_i - n + 1 \end{aligned} \tag{1.25}$$

ii. Logical OR

$$z = x_1 \vee x_2 \vee \dots \vee x_n \;;\; x_1, \dots, x_n \in 0, 1 \tag{1.26}$$

is linearized to

$$\begin{aligned} z &\ge x_1 \\ &\vdots \\ z &\ge x_n \\ z &\le \textstyle\sum_i x_i \end{aligned} \tag{1.27}$$

iii. Non-overlap constraints

These constraints enforce non-overlap of execution time (communication time) for parallel tasks (data transfers). [1.5] They have, as it was discussed before, the following typical structure:

$$T_A \le x_1 \dots x_n T_B \tag{1.28}$$

Which can be linearized to

$$T_A \le T_{MAX}(n - \sum_i x_i) + T_B \tag{1.29}$$

Where T_A and T_B are timing variables and $x_1, \dots, x_n \in 0, 1$.

[1.5] Non-preemptive execution of tasks and communication is assumed.

T_{MAX} is a constant large enough to be greater than any possible value for T_A or T_B [1.6].

A possible upper bound for T_{MAX} is

$$T_{MAX} = \sum_i \Omega(S_i) + \sum_j \Omega(DT_j) + 1 \tag{1.30}$$

Where

$\Omega(S_i)$ is the maximum possible computation time for $S_i = max_{p_x \in P_{S_i}}(\omega_{i\,x})$.
$\Omega(DT_j)$ is the maximum possible communication delay for DT_j

$$\Omega(DT_j) = V_{DT_j} max_l(D_{CL_x}) \tag{1.31}$$

1.2.1.17 Objective Function

The *objective function* in a MILP model for system-level design can be a *linear combination* of cost and performance (*system latency* T_{SYS}), or include only cost or performance.

$$f_{obj} = W_{cost}Cost + W_{T_{SYS}}T_{SYS} \tag{1.32}$$

$$Cost = \sum_{p_x} PC_x \beta_{p_x} + \sum_{l_i} LC_i \beta_{l_i} \tag{1.33}$$

$$\beta_{p_x} \Longleftrightarrow \text{processor } p_x \text{ is selected} \tag{1.34}$$

$$\beta_{l_i} \Longleftrightarrow \text{communication link } l_i \text{ is selected} \tag{1.35}$$

1.2.2 Modeling Uncertainty

Although MILP models for system level design were originally designed for deterministic problems, i.e. input parameters are constants (e.g. processor cost, time

[1.6] Practical experience on using MILP solvers indicates that the lowest possible value for T_{MAX} should be used to minimize the computational cost (CPU-time) of finding the optimal solution for the model.

of execution of a task in a processor), the same formalism can be used in expressing design in the presence of uncertainty. In this case, the input parameters will correspond to *random variables* [1.7], whose distributions may not be fully specified.

As available MILP solvers are restricted to deterministic MILP formulations, special solving methods have to be used in order to allow a *probabilistic optimization* of the model, as shown in Chapter 8.

1.3 Thesis Organization

The main goals of this thesis are the development of practical computer tools for system-level design as well as their formal mathematical bases. The proposed mathematical models in this thesis aim for the more general case where not all performance/cost parameters are deterministic, i.e. they may be random variables with a known *probability density distribution* (p.d.d.) or estimated (p.d.d. not fully specified). However some emphasis is given as well to the deterministic case.

This thesis is organized in the following manner. Chapter 2 describes the related work on mathematical formulations for system-level design, performance bounds theory, genetic algorithms, probabilistic approaches and imprecise computation.

Chapter 3 describes how a mixed-integer-linear-programming (MILP) model for system-level design of multiprocessors executing in a non-periodic fashion can be extended to handle the periodic *macro-pipelined* case, as well as how retiming can be used in system-level design of heterogeneous multiprocessors.

Chapter 4 describes how a particular property of the MILP models for system-level design can be used to derive a family of possible heuristic approaches, among them a genetic algorithm for design of application-specific multiprocessors. Chapter 5 discusses the motivations for applying genetic algorithms into system-level design problems, presents a brief survey on general genetic algorithms, and introduces a new template for variable size population genetic algorithms. Chapter 6 presents MEGA, which is a software package containing a set of genetic algorithms for system-level design of heterogeneous multiprocessors with non-negligible communication costs

[1.7] An approach not present in works of Chu et al. [16] and Prakash and Parker [99]

but deterministic parameters for cost and performance. Chapter 7 incorporates the *imprecise computation* approach in system-level design of multiprocessors. Imprecise computation of tasks allows a more detailed level of trade off between performance and cost. A genetic algorithm, based on an MILP formulation allowing imprecise computation, is presented. Chapter 8 describes how MEGA was extended to handle design in the presence of uncertainty, i.e. parameters for cost and performance are random variables with known or partially known probability density distributions.

Chapter 9 discusses possible improvements in the available analytical performance bound theory to build fast performance/cost predictors for system-level design. A set of predictors, allowing an easy and fast pre-evaluation by the designer of the many possible target multiprocessor systems, is presented.

Finally, Chapter 10 summarizes the main results of this thesis, highlighting possible future research topics in system-level design of application specific heterogeneous multiprocessors.

Chapter 2

Related Work

The problem of synthesis of application-specific heterogeneous multiprocessors is a part of the subject of system synthesis. Previous work in system synthesis includes graph theoretical approaches, analytical modeling approaches, probabilistic modeling approaches, mathematical programming formulations, application of heuristics, and performance bounds of solutions found by algorithms or heuristics for system-level design. A brief survey of relevant work on these and other subjects is presented below.

2.1 MILP Models

Chu et al. published one of the first mixed integer-linear programming (MILP) models for a sub-problem of system-level design, scheduling. Recently the program SOS (Synthesis of Systems), including a compiler of MILP models, [99] [100] was developed, based on a comprehensive MILP model for system synthesis. It takes a description of an task-flow graph, the processor library and some cost and performance constraints, and generates an output file with an MILP model to be optimized by the MILP solver BOZO [50]. The SOS tool generates MILP models for the design of non-periodic (non-pipelined) heterogeneous multiprocessors. The models share a common structure which is an extension of the previous work by Hafer and Parker for high level synthesis of digital systems [99].

The use of MILP theory for synthesis of both homogeneous and heterogeneous systems is not really new. Early efforts include those done by soviet researchers since the beginning of the 70's such as Linsky and Kornev [76], Vayradyan et al [131],

Barskiy [4] [5],Koryachko [66] and Venkova [132], where each model only included a subset of the aspects the entire problem considered by Prakash and Parker. However, all these works and others mentioned before, including one by Prakash and Parker, address only the non-periodic case.

2.2 Macro-Pipelined Design

Previous work on design of *macro-pipelined* (periodic) multiprocessors has been restricted to *homogeneous* multiprocessors having negligible communication costs. The present survey divides the past contributions according to the execution mode: *preemptive* or *non-preemptive*.

2.2.1 Non-Preemptive Mode of Execution

The non-preemptive mode of execution assumes that each task is executed without interruption. It is used quite often in low cost implementations.

2.2.1.1 Lower Bound on the Initiation Interval

The minimum possible value for the *initiation interval* T_I for a task-flow graph G given an unlimited number of processors and no communication costs was found by Renfors and Neuvo [107].

$$T_{I_{min}} = max(T_{c_1}, T_{c_2}, \ldots, T_{c_n}) \tag{2.1}$$

$$T_{c_i} = \frac{\sum_{S_x \in c_i} \omega(S_x)}{\sum_{e \in c_i} w(e)} \tag{2.2}$$

where c_i is a cycle in G, $\omega(S_x)$ is the computation time of task S_x and $w(e)$ (edge weight) is equal to the number of delays in edge e.

2.2.1.2 Full Static Scheduling

Parhi [96] [97] proposed the use of loop unfolding to reach Renfors and Nuevos's bound, i.e. *rate optimal* scheduling. A *rate-optimal* data-flow graph is one that can be executed with an iteration interval equal to the *iteration bound*.

Parhi proves that any *cyclic strongly-connected* data-flow graph can be *unfolded* f_{Parhi} times to become *rate optimal.*

$$f_{Parhi} = lcm(D_1, D_2, \ldots, D_n) \tag{2.3}$$

$$D_k = \sum_{e \in Cycle_k} w(e) \tag{2.4}$$

Where D_k is the number of delays on the kth-*cycle*, and $lcm(m_1, m_2, \ldots, m_n)$ is the *least common multiplier* of $m_1, m_2, \ldots, m_n$.

Parhi's work does not consider communication costs or task execution times. Wang and Hu [134] [135] proved that a lower *unfolding factor* f_{HU} can be used instead, taking advantage of the information about task execution times. They extended the concept of *perfect-rate data-flow programs* formulated by Parhi and they proposed the concept of *generalized perfect rate data-flow programs.*

Wang and Hu use heuristics for the allocation and scheduling of *generalized perfect-rate task-flow graphs* on homogeneous multiprocessors. Wang and Hu apply *planning*, an artificial intelligence based method, into the task scheduling problem. The processor allocation problem is solved by a *conflict-graph-based* approach. Both works assume *full static scheduling*, meaning that each task is executed on the same processor for all iterations.

2.2.1.3 Cyclic Static Scheduling

Gelabert and Barnwell [39] developed an optimal method to design *macro pipelined* homogeneous multiprocessors using *cyclic-static scheduling*, where the *task-to--processor* mapping is not *time-invariant* as in the *full static* case, but is periodic, i.e. the tasks are successively executed by all processors.

Gelabert and Barnwell assume that the delays for *intra-processor* and *inter-processor* communications are the same, which is an idealistic scenario. Their approach is able to find an optimal implementation (minimal iteration interval) in exponential time in the worst case.

2.2.1.4 Graph Theory Applied to Macro-Pipelined Design

Hanen and Murier [53] model the periodic scheduling problem by a set of *uniform constraints* represented by a uniform constraint graph G. They show that when G is circular, it is possible to find the optimal schedule in polynomial time (minimal initiation interval) on a homogeneous multiprocessor without communication costs and *full static* task-to-processor mapping. They use the concept of *tie-breaking* graphs to find a set of *dominant* schedules as possible candidates to the optimal solution. Chretienne studied the *periodic scheduling* problem with deadlines [20], a key issue in real-time systems.

2.2.1.5 Retiming, Unfolding and Rotation

Retiming [75] and *unfolding* are among the most commonly used loop/TFG transformations for achieving *rate-optimal* implementations. Chao et al. [13] proposed a set of heuristics using retiming, unfolding and task rotation to minimize the *unfolding* factor and maximize the chances of finding *rate-optimal* schedules. Their approach was developed to be applied to high-level design of data paths and it is, therefore, limited to homogeneous multiprocessors without communication costs.

2.2.1.6 Functional Pipeline

Park and Parker [94] [95] introduced the theoretical basis for optimal design of data paths with *functional pipelines*, without inter-iteration data dependencies, which limits their applicability to periodic scheduling of acyclic task-flow graphs.

2.2.1.7 Multidimensional Periodic Multiprocessors

The problem of periodic multidimensional scheduling is addressed by Verhaegh [133]. His thesis uses an integer-linear programming (ILP) model to handle the design of homogeneous multiprocessors without communication costs implementing data-flow programs with nested loops. His work evaluates the complexity of the scheduling and allocation problems for the multidimensional case, which were both found to be NP-complete. Verhaegh proposes a set of heuristics to handle both problems.

Passos and Sha [98] evaluate the use of multi-dimensional retiming for synchronous data-flow graphs. However, their formalism can only be applied to homogeneous multiprocessors without communication costs.

2.2.2 Preemptive Mode of Execution

The optimal static allocation of periodic tasks with precedence constraints and preemption on a homogeneous multiprocessor is addressed by Feng and Shin [30]. Their approach has an exponential time complexity. Ramamrithan [103] developed a heuristic method which has a more reasonable computational cost. *Rate-monotonic scheduling* RMS is a commonly used method for allocating periodic real-time tasks in distributed systems [102]. The same method can be used in homogeneous multiprocessors.

2.3 Soft Computing Methods - Genetic Algorithms

Genetic algorithms are becoming an important tool for solving the highly nonlinear problems related to system-level synthesis. They incorporate the ideas of natural evolution of Darwin - survival of the fittest [34]. They were originally developed by Holand and his students at the University of Michigan in the late 1960s [137]. The use of genetic algorithms in optimization is well discussed in [89], where formulations for problems such as bin packing, processor scheduling, traveler salesman and system partitioning are outlined. More theoretical issues on the subject of genetic algorithms are available in many references [105] [56] [8] [9] [3] [113] [33].

Research works involving the use of genetic algorithms to system-level synthesis problems are starting to be published, as for example the results of

- Hou et al. [57] - scheduling of tasks in a homogeneous multiprocessor without communication costs;
- Wang et al. [136], Singh and Youssef [123], Shroff [121] - scheduling of tasks in heterogeneous multiprocessors with communication costs but not allowing *cost versus performance* trade-off, i.e. all processors have the same cost;

- Ravikumar and Gupta [104] - mapping of tasks into a reconfigurable homogeneous array processor without communication costs; and
- Tirat-Gefen and Parker [127] [2.1] - a genetic algorithm for design of application-specific heterogeneous multiprocessors with non-negligible communications costs and a *non-pure data-flow* (*task-flow graph*) non-periodic paradigm of computation as discussed in Section 1.2.1.1.

2.4 Imprecise Computation

The main results in *imprecise computation* theory are due to Liu et al. [119] [18] [79] who developed polynomial time algorithms for optimal scheduling of preemptive tasks in homogeneous multiprocessors without communications costs. Ho et al. [55] proposed an approach to minimize the *total error*, where the *error* of a task being *imprecisely* executed is proportional to the amount of time that its *optional* part was not allowed to execute, i.e., the time still needed for its full completion. Polynomial time-optimal algorithms were derived for some instances of the problem [79].

2.5 Probabilistic Models and Stochastic Simulation

Many probabilistic models for solving different sub-problems in digital design have been recently proposed. Sastry [115] [116] [117] developed a stochastic approach for estimation of wireability (routability) for gate arrays. Kurdahi [70] created a discrete probabilistic model for area estimation of VLSI chips designed according to a standard cell methodology. Küçükçakar [67] [68] [69] introduced a method for partitioning of behavioral specifications onto multiple VLSI chips using probabilistic area/performance predictors. Statistical approaches to circuit design are exemplified by Wojciehowski and Vlach [138].

The problem of task and data-transfer scheduling on a multiprocessor when some tasks (data transfers) have non-deterministic execution times (communication-times)

[2.1] This work is actually a pre-print of some of the research results of this dissertation.

can be modeled by PERT networks, which were introduced by Malcolm et al. [84] along with the critical path method (CPM) analysis methodology [84].

A survey on Pert networks and their generalization to *conditional PERT networks* is done by Elmaghraby [27]. In system-level design, the completion time t_c of a PERT network corresponds to the *system latency* T_{SYS}, whose cumulative probability distribution (c.d.f.) is a non-linear function of the probability density distributions of the computation times of the tasks and the communication times of the data transfers in the task-flow graph.

The exact computation of the cumulative probability distribution function (c.d.f.) of the completion time is computationally expensive for large PERT networks, therefore it is important to find approaches that approximate the value of the expected time of the completion time and its c.d.f. One of the first of these approaches was due to Fulkerson [37], who derived an algorithm to find a tight estimate (lower bound) of the expected value of the completion time. Robillrad and Trahan [111] proposed a different method using the characteristic function of the completion time in approximating the c.d.f. of the completion time ($\mathcal{F}_{cdf}(t_c)$).

Mehrotra et al. [88] proposed an heuristic for estimating the moments of the probabilistic distribution of t_c. An approach based on Markov processes was developed by Kulkarni and Adlakha [71] for the same problem. Hagstrom [52] introduced an exact solution for the problem when the random variables modeling the computation and communication times are finite discrete random variables. Kamburowski [62] developed a tight upper bound on the expected completion time $E(t_c)$ of a PERT network.

Another approach for estimating the c.d.f. of the completion time is the development of upper and lower bounds for $\mathcal{F}_{cdf}(t_c)$ as proposed by Kleindorfer [65] for general PERT networks and improved by Shogan [120] for the particular case where the edge delays (activities) of the PERT network are modeled as discrete random variables.

Series-parallel PERT networks have particular properties that allow the evaluation of $\mathcal{F}_{cdf}(t_c)$ without a high computational cost. Martin [85] was the first to propose an exact analysis for series-parallel PERT networks. Dodin introduced a method for evaluation of upper and lower bounds for $\mathcal{F}_{cdf}(t_c)$ by reducing a general PERT network to a series-parallel approximation.

Van Styke [130] proposed the use of Monte Carlo simulation, a type of stochastic simulation, for finding $\mathcal{F}_{cdf}(t_c)$. Burt and Graman [12] investigated how the use of *conditioning* could speed up the Monte Carlo Simulation, i.e. some random variables are assumed to be constants during the simulation. Ripley [110] authored a book describing the different stochastic simulation methods and possible approaches to decrease their computational costs.

An algorithm to find the probability that a path is critical in a PERT network was developed by Fisher et al. [32]. Numerical operators for PERT-Critical Path analysis were studied by Ringer [109]. An example of optimization using Monte Carlo Simulation is reported by Ruppert et al. [114] for a problem in the fishing industry.

An approach using random graphs to model distributed computations was introduced by Indurka et al. [60], whose theoretical results were improved by Nicol [93]. Formal methods applied to concurrent systems with a probabilistic behavior were proposed by Purushotaman and Subrahmanyam [101]. An example of modeling using queueing networks instead of PERT networks is given by Thomasian and Bay [129]. Estimating errors due to the use of PERT assumptions in scheduling problems is discussed by Lokaszewicz [81].

The shortest route problem, a close relative of the critical path evaluation problem, was studied by Kamburowski [63] and Sigal et al. [122]. Linear programming and perturbation analysis applied to probabilistic models described by a system of linear equations were studied by Elsayed and Ettouney [29]. Their approach is limited to problems with a small number of random variables, which limits their applicability to task scheduling in the presence of uncertainty in the computation and communication times.

2.6 Performance Bounds Theory

Lower bounds on the performance and execution time of task-flow graphs mapped to a set of available processors and communication links were developed by Liu and Liu [78] for the case of heterogeneous processors but no communication costs and by Hwang et al. [59] for homogeneous processors and communication costs. Tight lower bounds on the number of processors and execution time for the case of homogeneous

processors and presence of communication costs were developed by Al-Mouhamed [90]. The use of performance and cost lower bounds allows smaller solution spaces to be searched by the MILP solver. which leads to faster computation of the optimal solution. The previous work done on performance bounds is detailed in Chapter 9.

2.7 Summary

Research results on the subject of MILP models applied to system-level design, macro-pipelined design. retiming. genetic algorithms. imprecise computation. probabilistic approaches applicable to application-specific multiprocessors design, and performance bounds theory were reviewed. Emphasis was given to previous work related to the theoretical developments to be presented in the following chapters.

Chapter 3

Macro-Pipelined System Level Design

3.1 Motivation

This chapter presents our research results on the subject of design of macro-pipelined heterogeneous multiprocessors.

Macro-pipelining is a typical design style found in application-specific hardware for high performance digital signal processing (DSP) applications such as high definition television (HDTV), video compression (MPEG/JPEG) and multimedia. A multiprocessor is *macro-pipelined* if it allows the parallel execution of more than one iteration of the task-flow graph to be implemented. A good macro-pipelined (periodic) design is one that is optimized for successive execution of these iterations.

As an example of *macro-pipelined* design, consider the processing of a stream of video frames. Each video frame will correspond to an iteration of the task-flow graph representing the functions to be implemented in hardware. There are a infinite number of video frames to be processed. The period or *initiation interval* is the inverse of the frame frequency in this example.

As the sharing of processors and communication links among successive iterations may produce cheaper implementations, methods for optimal or near-optimal design should be able to consider periodic scheduling and allocation simultaneously. Schedule and allocation heuristics/algorithms for non-periodic design do not take advantage of particular properties of the periodic scheduling/allocation problems which allow certain optimizations.

3.2 Definitions

The following definitions will be useful in describing our research on macro-pipelined heterogeneous multiprocessors.

3.2.1 Instance of Task (Data Transfer)

The *instance* $S_r[i]$ ($DT_r[i]$) of task S_r (data transfer DT_r) corresponds to its execution on iteration i.

For example

$$
\begin{array}{l}
\text{for } i = 0 \text{ to } \infty \text{ begin} \\
\qquad S_A(\textbf{in } a[i], b[i]; \textbf{ out } c[i+1]) \\
\qquad S_B(\textbf{in } c[i]; \textbf{ out } d[i]) \\
\text{end}
\end{array}
$$

is equivalent to

$$S_A(\textbf{in } a[0], b[0]; \textbf{ out } c[1])$$
$$S_B(\textbf{in } c[0]; \textbf{ out } d[0])$$

$$S_A(\textbf{in } a[1], b[1]; \textbf{ out } c[2])$$
$$S_B(\textbf{in } c[1]; \textbf{ out } d[1])$$

$$\vdots$$

$$S_A(\textbf{in } a[i], b[i]; \textbf{ out } c[i+1])$$
$$S_B(\textbf{in } c[i]; \textbf{ out } d[i])$$

3.2.2 Full Static and Cyclic Static Allocation

Full static task allocation means that if the *task instance* $S_r[i]$ is allocated to processor p on iteration i, the same is true for $S_r[j]$ on iteration j, $j \neq i$. The definition is similar for communication links and data transfers.

In a *cyclic static* allocation, the processor $p[i]$ allocated to *task instance* $S_r[i]$ on iteration i, where $p[i]$ is also a cyclic function of i with a constant period T_c, i.e.

$$S_r[i] \text{ is allocated to processor } p[i] = p_j, \text{ where}$$

$$j = (i \;\; mod \;\; T_c) + K_p(S_r) \tag{3.1}$$

T_c is the *allocation cycle period* and $K_p(S_r)$ is an function of task S_r, indicating the processor allocated to instance $S_r[0]$.
Data transfers and communication links are handled in a similar way.

$$DT_r[i] \text{ is allocated to link } l[i] = l_j, \text{ where}$$

$$j = (i \;\; mod \;\; T_c) + K_l(DT_r) \tag{3.2}$$

and $K_l(S_r)$ is a function of data transfer DT_r.

Although *cyclic static* allocation may lead to faster implementations [39] than by using *fully static* allocation, the former requires more expensive hardware/software support, to allow each task to be periodically shifted among different processors. This thesis is oriented towards low cost/high performance designs, therefore only *fully static* allocation is considered.

3.2.3 Preemptive Execution

Given a set of tasks $S_{TASK} = \{S_r, \ldots, S_s\}$ allocated to the same processor p. The set of possible task schedules can be divided into two subsets:

Non-preemptive: Each *task instance* is executed till completion without interruption.

Preemptive: A *task instance* can suffer a *context-switch*, i.e. it may be interrupted temporarily, surrending the processor to be used to a *task instance* of higher *priority*, where the *priority* is defined by the particular scheduling algorithm being used.

The same concept of preemption can be applied to a set of data transfers allocated to a communication link l. The use of *preemption* when scheduling tasks and/or data transfers usually requires expensive hardware/software support. This thesis considers only *non-preemptive* methods for design of application-specific multiprocessors.

3.2.4 Static Scheduling

This dissertation assumes that the schedule of tasks and data transfers is defined at *design time*, as indicated by the equations below:

$$T_{SS}(S_r[i+1]) = T_{SS}(S_r[i]) + T_I(i) \tag{3.3}$$

$$T_{SE}(S_r[i+1]) = T_{SE}(S_r[i]) + T_I(i) \tag{3.4}$$

$$T_{CS}(DT_s[i+1]) = T_{CS}(DT_s[i]) + T_I(i) \tag{3.5}$$

$$T_{SS}(DT_s[i+1]) = T_{SS}(DT_s[i]) + T_I(i) \tag{3.6}$$

If T_I is made equal to a constant T_I, the same is called *initiation interval*, which corresponds to a fully static (*time-invariant*) schedule of tasks and transfers.

$$\forall i \ . \ T_I[i] = T_I = \ \textit{initiation interval} \tag{3.7}$$

Static schedules are usually less expensive than *dynamic schedules*, which are dependent on run-time support. This thesis assumes the use of *static scheduling* unless otherwise mentioned.

3.2.5 Initiation Interval, System Latency and Parallelism Degree

Figure 3.1 illustrates a task-flow graph executing in a *macro-pipelined* multiprocessor using a *full static* allocation and schedule when in *steady state* after the prologue of the loop has been executed [13]. In Figure 3.1 Iteration i is the *base iteration* [22].

The *base iteration* is a moving time reference. The *timing variables* of the other iterations are expressed as linear functions of the *timing variables* of the *base iteration*, as discussed in Section 3.2.4.

As a *full static* allocation and scheduling with a constant *initiation interval* T_I are used, all iterations are *synchronous*, i.e. each iteration is started in regular intervals of T_I units of time. Task instances are also displaced by multiples of T_I as seen in Figure 3.1.

The duration of an iteration is called the *iteration latency* T_{SYS}. The number of iterations executing in parallel with the *base iteration* is given by n_{par} (*parallelism degree*).

3.2.5.1 Finite and Infinite Horizons

An algorithm/heuristic attempting to map TFGs to *macro-pipelined* application-specific multiprocessors can be classified by the maximum allowed *parallelism degree.*

3.2.5.1.1 Infinite Horizon

For infinite horizons, n_{par} is bounded only by the maximum allowed implementation cost and/or *system latency*. If there is no limit on the processor cost/system latency, the algorithm/heuristic will try to schedule/ allocate as many possible iterations in parallel subject to the *iteration bound* (Section 2.2.1.1) if the task-flow graph is cyclic. However, *infinite* horizon methods do not allow the use of an MILP formulation for the *non-overlap* constraints as in the non-periodic case described in Section 1.2.

3.2.5.1.2 Finite Horizon

With finite horizons n_{par} is bounded by the designer. Therefore, there is a limit on the maximum number of iterations to be executed in parallel. *Finite horizon* methods allow the use of MILP formulations such as the ones discussed in Section 1.2.

3.2.6 Overlapping Factor

Assuming a *full static* schedule and allocation with a constant *initiation interval* and *finite horizon*, the designer can express the maximum level of parallelism by the following constraints:

$$T_{SYS} \leq nT_I$$
$$n_{par} \leq 2n - 1 \tag{3.8}$$

where n is the *overlapping factor*. The *overlapping factor* can be understood as a limit on the number of iterations starting after the *base iteration*, while the latter is still being executed.

For a *overlapping factor* of n, there will be at most $2n - 1$ iterations executing concurrently as seen in Figure 3.1, i.e. at most $n - 1$ after and at most $n - 1$ before the *base iteration*. The overlapping factor limits the *horizon* to be considered by the TFG mapping algorithm/heuristic.

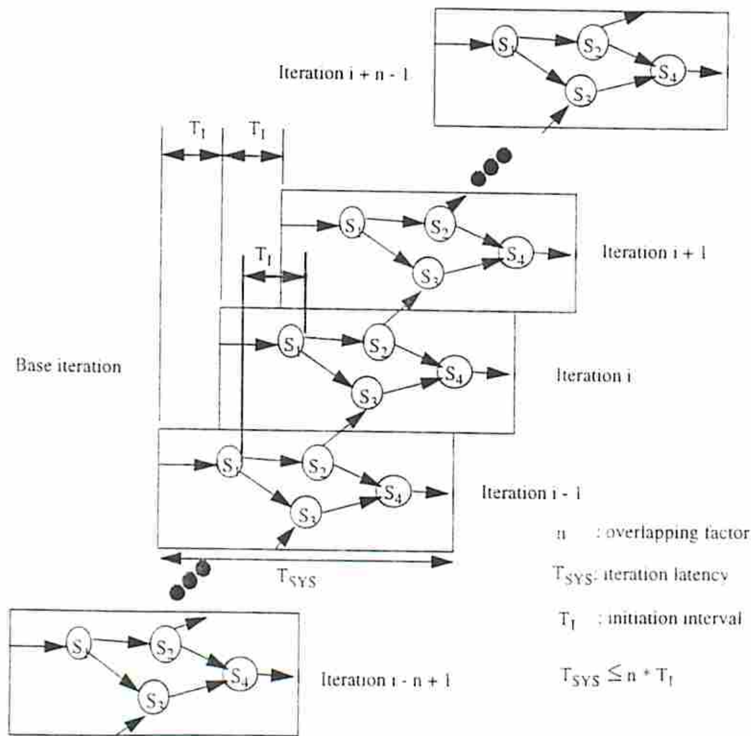


Figure 3.1: Overlapping factor, iteration latency and initiation interval

3.2.7 Task-flow Graph Transformations

An important issue in system-level *macro-pipelined* design is the use of TFG transformations. Given a task-flow graph specifying an application to be implemented by a dedicated heterogeneous multiprocessor, we are looking for transformations that could be applied to the TFG in order to find *transformed* TFGs leading to designs with lower implementation costs than one derived directly from the original TFG.

The transformed TFG_{tr} must be functionally equivalent to the original TFG, i.e. the transformations to be applied in the original TFG should be *behavior preserving*. In other words, we are trying to find transformations in the application specification that would potentially lead to superior designs but at the same time fulfill all the requirements of the original specification. Figure 3.2 illustrates this concept, where the original TFG is *pipelined*. The results of *task instance* $S_1[i]$ are sent to the iteration $i+1$, however the computational effect is the same as in the original TFG.

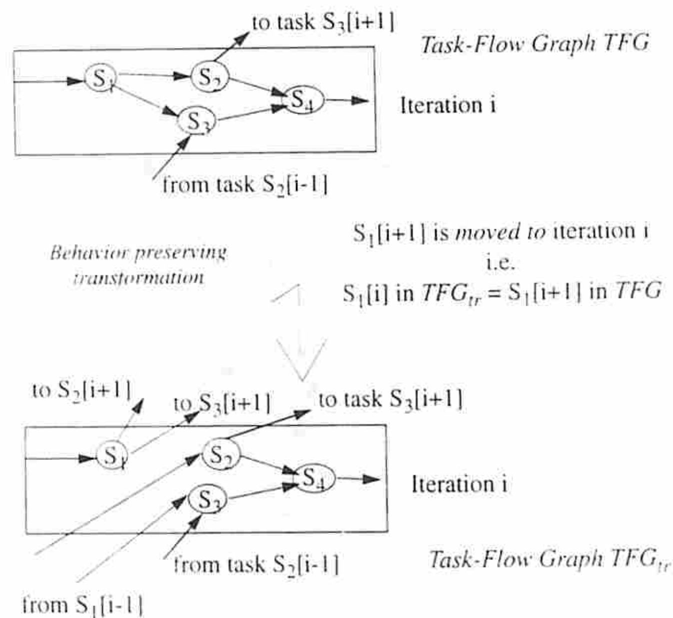


Figure 3.2: A behavior preserving transformation

3.2.7.1 Unfolding

A straightforward transformation is *unfolding*, where for an unfolding factor f_{unfold} also called *unrolling factor* [22], f_{unfold} iterations of the original TFG are grouped together in a *macro-iteration* as seen in the following example:

Before unfolding

for $i = 0$ to ∞

$S_A(in\ a[i];\ out\ b[i],\ c[i]);$

$S_B(in\ b[i];\ out\ d[i],\ h[i]);$

$S_C(in\ d[i-1];\ out\ g[i]);$

end

After unfolding by f iterations

for $j = 0$ to ∞

$S_{A_0}(in\ a_0[j];\ out\ b_0[j],\ c_0[j]);$

$S_{B_0}(in\ b_0[j];\ out\ d_0[j],\ h_0[j]);$

$S_{C_0}(in\ d_0[j-1];\ out\ g_0[j]);$

$S_{A_1}(in\ a_1[j];\ out\ b_1[j],\ c_1[j]);$

$S_{B_1}(in\ b_1[j];\ out\ d_1[j],\ h_1[j]);$

$S_{C_1}(in\ d_1[j-1];\ out\ g_1[j]);$

$\vdots$

$S_{A_{f-1}}(in\ a_{f-1}[j];\ out\ b_{f-1}[j],\ c_{f-1}[j]);$

$S_{B_{f-1}}(in\ b_{f-1}[j];\ out\ d_{f-1}[j],\ h_{f-1}[j]);$

$S_{C_{f-1}}(in\ d_{f-1}[j-1];\ out\ g_{f-1}[j]);$

end

Where the *stream-element* $x_k[j]$ (after unfolding) corresponds to the *stream-element* $x[f*j+k]$ (before unfolding) for $x = a, b, c, d, h$ or g.

3.2.7.2 Retiming

In *retiming*, the differences of subscripts of the data transfer (edge weights) are changed in order to allow more parallelism in the TFG $= (V, E)$. The *retiming*

function is defined as a function $V \longrightarrow \mathcal{Z}$, where V is the set of nodes (
TFG, E is the set of edges and $\mathcal{Z}$ is the set of integer numbers.

The weight of the edge $e = S_u \longrightarrow S_v$ after retiming $(w_r(e))$ is given by

$$w_r = w(e) + r(S_v) - r(S_u) \quad ($$

A valid *retiming* is one where

$$\forall e \in \quad w_r \geq 0 \tag{3.10}$$

Where constraint (3.10) is needed in order to enforce *causality*. Figure 3.3 contains an example of a valid retiming. *Pipelining* can be understood as a particular type of retiming [75].

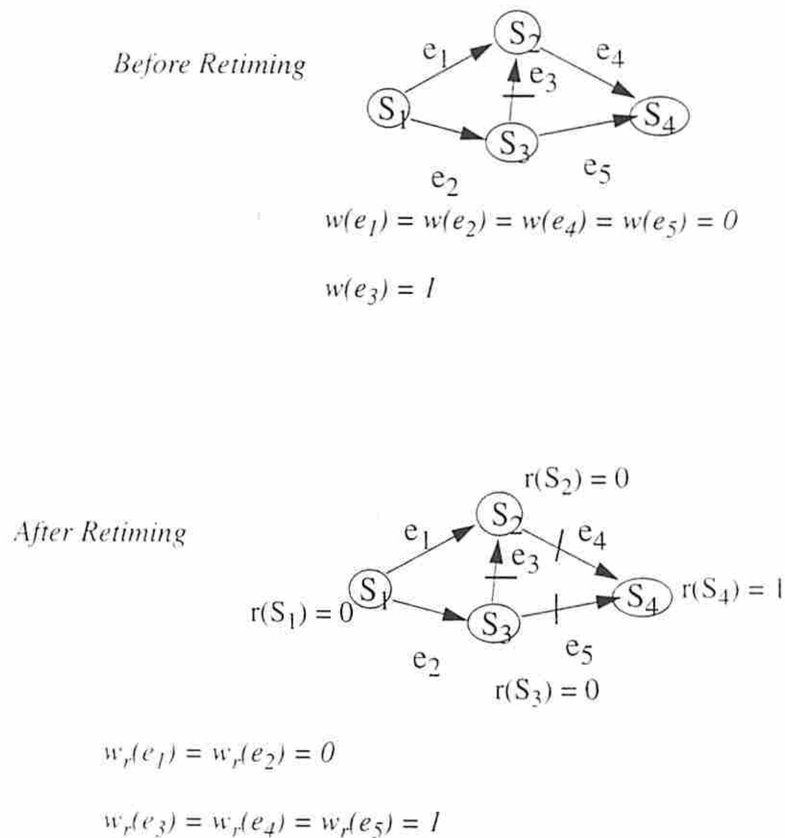


Figure 3.3: Applying retiming to a TFG

3.3 A MILP Model for Macro-Pipelined Design with Finite Horizon

An extension of the MILP model described in Section 1.2.1 is presented here. The MILP formulation assumes the use of an *overlapping factor* n supplied by the designer and it allows the simultaneous optimization of cost, system latency T_{SYS} and initiation interval T_I by using the following objective function:

$$f_{obj} = w_{T_I} * T_I + w_{T_{SYS}} * T_{SYS} + w_{Cost} * Cost \tag{3.11}$$

where T_{SYS} and T_I are correlated by the maximum *inter-iteration overlap constraint.*

$$T_{SYS} \leq n * T_I \tag{3.12}$$

and w_{T_I}, $w_{T_{SYS}}$ and w_{Cost} give the relative importance of minimizing T_I, T_{SYS} and the overall cost.

The MILP model discussed is this section is used by the tool *pipelined-SOS* [22] which handles optimal design of macro-pipelined heterogeneous multiprocessors with communication costs.

3.3.1 Unrolling Factor

Better designs in terms of cost (sharing of resources) and performance (better use of idle resources) may be achieved if two or more consecutive iterations are merged together to form a *macro-iteration*, i.e. by using *unrolling (unfolding)*. In the present model, it is assumed that unrolling has already been done at the task-flow-graph level.

3.3.2 Synchronism between Iterations

Full static mapping is assumed, that implies that β_{p_x}, $\sigma_{i\,x}$, $\gamma_{i\,j}$ and $\eta_{l\,x}$ are *time-invariant* (see Section 1.2.1), i.e. all instances of a task (data transfer) are allocated to the same processor (link).

$$
\begin{aligned}
T_{SS}(S_r[k]) &= T_{SS}(S_r[0]) + k * T_I \\
T_{SE}(S_r[k]) &= T_{SE}(S_r[0]) + k * T_I \\
T_{CS}(DT_s[k]) &= T_{CS}(DT_s[0]) + k * T_I \\
T_{SS}(DT_s[k]) &= T_{SS}(DT_s[0]) + k * T_I
\end{aligned} \tag{3.13}
$$

Where $S_r[0]$ ($DT_s[0]$) denotes a task (data-transfer) instance in the *base iteration.*

3.3.3 Overlap Detection Windows

Both tasks and data transfers (inter-task communications) share the available processors and communication links between processors. They should be allocated and scheduled respectively to processors and links in such a way that no overlap (conflict) happens, i.e., no more than one computation (communication) is being carried in a processor (link) at any instant of time.

The *pipelined-SOS* MILP model assumes the use of different *overlap windows* for detecting overlap among task and data transfers. The concept of *overlap detection windows* is a direct consequence of the use of a bounded *overlapping factor*, full static allocation and scheduling (time-invariant mapping) and constant initiation interval T_I.

3.3.3.1 Iteration and Time-Invariance

The concept of *base iteration* allows a reduction in the number of non-overlap constraints to be required in the MILP formulation.

Theorem 3.1 *There is only a need to detect overlap between a pair of task (data-transfer) instances of only the type $S_x[0]$ and $S_y[i]$ ($DT_x[0]$ and $DT_y[i]$) where $S_x[0]$ ($DT_x[0]$) denotes a task (data transfers) in the* base iteration.

Proof:

A full static mapping with constant T_I is used.

Overlapping between $S_x[r]$ and $S_y[s]$ ($DT_x[r]$ and $DT_y[s]$) $\Longleftrightarrow$ Overlapping between $S_x[0]$ and $S_y[s-r]$ ($DT_r[0]$ and $DT_y[s-r]$). □

In other words, there is no need to enforce non-overlap between task instances that are not in the base iteration, their overlap, if it exists, corresponds to an overlap where at least one of the task instances is in the base iteration.

3.3.3.2 Task Overlap Detection Window

Due to the *steady state* restriction $T_{SYS} \leq n * T_I$, the *task overlap detection* window will have at most $n-1$ iterations above and $n-1$ iterations below the *base iteration*, with a total of at most $2n-1$ iterations being executed in parallel in the time span covered by the window as seen in Figure 3.4.

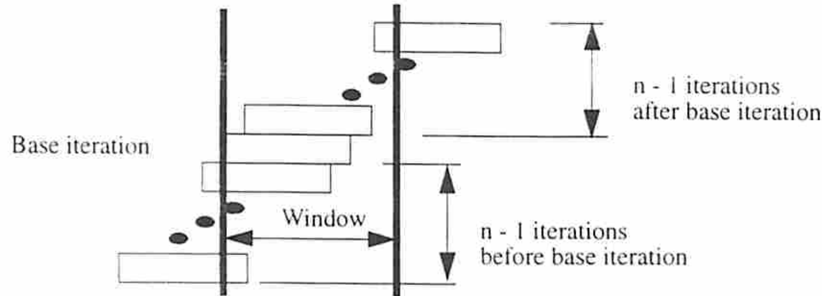


Figure 3.4: Task execution overlap detection window

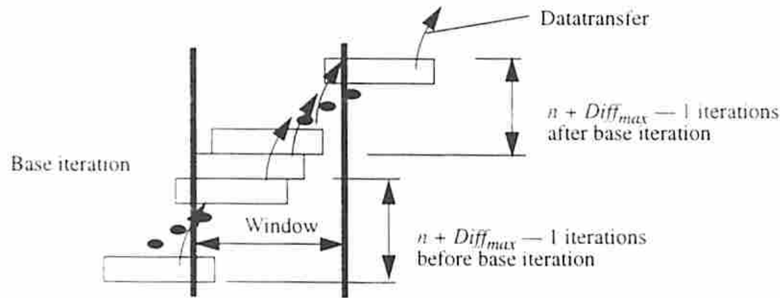


Figure 3.5: Data transfer overlap detection window

3.3.3.3 Data Transfer Overlap Detection Window

For the case where there are data transfers among *task instances* in different iterations, i.e. there are *data dependencies* between iterations, the length of the *data transfer overlap detection* window is a function of n, the overlapping factor, and $Diff_{max}$, the maximum subscript difference, defined as follows:

Definition 3.1 *Let task instance $S_i[r]$ be data dependent on task instance $S_j[s]$ by data transfer DT_k. The subscript difference of this data transfer is $d_k = r - s = w(e = S_s \longrightarrow S_r)$. $Diff_{max}$ is given by*

$$Diff_{max} = max(|d_1|, \ldots, |d_{ndt}|)$$

Where ndt is the number of data transfers (edges) of the task-flow graph.

$$if\ Diff_{max} = 0 \Longleftrightarrow there\ are\ no\ inter\text{-}iteration\ data\ transfers.$$

Theorem 3.2 *In order to ensure non-overlap of data transfers at the communication links, it is sufficient to include* $n + Diff_{max} - 1$ *above and below the* base iteration *in the* data-transfer overlap detection window.

Proof:
There are four steps in the proof.

i. Let DT_i be a data transfer with a subscript difference equal to $0 \leq d \leq Diff_{max}$, i.e. $DT_i[0] = S_r[0] \longrightarrow S_s[d]$. Without loss of generality, let the *base iteration* start at instant of time 0:

$$0 \leq T_{SS}(S_r[0])\ ,T_{SE}(S_r[0])\ ,T_{SS}(S_s[0])\ ,T_{SE}(S_s[0]) \leq T_{SYS} \tag{3.14}$$

$$T_{SE}(S_s[d]) = T_{SE}(S_s[0]) + d * T_I \tag{3.15}$$

By 3.14 $T_{SE}(S_s[0]) \leq T_{SYS}$, that applied to 3.15 leads to

$$T_{SE}(S_s[d]) = T_{SE}(S_s[0]) + d * T_I \leq T_{SYS} + d * T_I \tag{3.16}$$

As

$$0 \leq T_{SS}(S_r[0]) \leq T_{CS}(DT_i[0]) \leq T_{CE}(DT_i[0]) \leq T_{SE}(S_s[d]) \tag{3.17}$$

By 3.14, 3.16 and 3.17

$$0 \leq T_{CS}(DT_i[0]) \leq T_{CE}(DT_i[0]) \leq T_{SYS} + d * T_I \tag{3.18}$$

Therefore it is sufficient to enforce non-overlap between $DT_i[0]$ and any other data-transfer instance $DT_j[k]$ starting or ending in the interval $[0\ ,\ T_{SYS}+d*T_I]$. As $0 \leq d \leq Diff_{max}$, the interval of interest is $\mathcal{I}(0) = [0\ ,\ T_{SYS}+Diff_{max}*T_I]$ for all data-transfer instances starting in the *base iteration*.

ii. From 3.18, all data transfers starting on iteration $k \in \mathcal{Z}$ occur in a sub-interval of $\mathcal{I}(k) = [k * T_I \, . \, T_{SYS} + (Diff_{max} + k) * T_I]$, where $k = 0$ corresponds to the base iteration and $k < 0$ ($k > 0$) denotes an iteration before (after) the *base iteration*.

iii. By the definition of *overlapping factor*: $0 < T_{SYS} \leq n * T_I$

iv. By inspection only iterations in the range $-n - Diff_{max} + 1 \leq k \leq n + Diff_{max} - 1$ have $\mathcal{I}(k)$ with an non-zero length overlap with $\mathcal{I}(0) = [0 \, . \, T_{SYS} + Diff_{max} * T_I]$. Therefore a *data-transfer overlap detection* window of $n + Diff_{max} - 1$ iterations above the base iteration and $n + Diff_{max} - 1$ iterations below is sufficient. □

3.3.3.4 Sufficiency of the Overlap Detection Windows

Theorem 3.3 *If there is not an* overlap *of tasks (data transfers) inside the* overlap detection windows *and* $0 \leq T_I \leq T_{SYS}$*, there will not be any overlap between tasks (data transfers) outside the overlap window.*

Proof:

By reduction to absurd assume an overlap between two parallel task (data transfer) instances allocated to the same processor p (link l) on instant of time t. A *time-invariant* mapping with constant *initiation interval* T_I is used.

i. Overlapping on $t \Longrightarrow$ Overlapping on $t' = t + k * T_I$, $k \in \mathcal{Z}$. $\mathcal{Z}$ is the set of integer numbers.

ii. The *overlap detection window* has *non-zero* length, i.e. it is at least equal to T_{SYS}

iii. $0 < T_I \leq T_{SYS}$

Without loss of generality, it can be assumed that the base iteration starts at *instant of time* 0. By (i), (ii) and (iii), it is possible to find a value for k such that t' is inside the *overlap detection window*, i.e.

$$0 \leq t' = t \;\; mod \;\; T_I \leq T_{SYS} \tag{3.19}$$

$$0 \leq k = \frac{(t - t')}{T_I} \tag{3.20}$$

Therefore if there is no overlap inside the window, there is no overlap outside. □

3.3.3.5 Independence between Overlap Detection Windows

The *overlap avoidance* constraints for tasks involve variables of the type $T_{SS}(\bullet)$ and $T_{SE}(\bullet)$ and for data transfers involve $T_{CS}(\bullet)$ and $T_{CE}(\bullet)$. The detection of overlap between tasks can be made independent of the overlap detection between data transfers and vice-versa, once the values for the timing variables are known.

3.3.4 Binary Variables

All binary variables present in the MILP model for *macro-pipelined* (periodic) design are present in the MILP model for the non-periodic case discussed in Section 1.2.1. However the variable types α and Φ are redefined for the micro-pipelined case.

3.3.4.1 Redefinition of Type α Variables

Definition 3.2 $\alpha_{i\ j}[a,\ b]$

- *$\alpha_{i\ j}[a,\ b]$ is True if task instance $S_i[a]$ finishes before the start of task instance $S_j[b]$ and tasks S_i and S_j are allocated to the same processor.*
- *$\alpha_{i\ j}[a,\ b]$ is False is task instance $S_j[b]$ finishes before the start of task instance $S_i[a]$ and tasks S_i and S_j are allocated to the same processor.*
- *$\alpha_{i\ j}[a,\ b]$ is True if S_i and S_j are not allocated to the same processor.*

3.3.4.2 Redefinition of Type Φ Variables

Definition 3.3 $\Phi_{k\ l}[a,\ b]$

- *$\Phi_{k\ l}[a,\ b]$ is True if data-transfer instance $DT_k[a] = S_i[a] \longrightarrow S_j[a+u]$ finishes before the start of data-transfer instance $DT_l[b] = S_r[b] \longrightarrow S_s[b+v]$ and data-transfers DT_k and DT_l are allocated to the same communication link.*

- $\Phi_{k\,l}[a,\ b]$ *is False if data-transfer instance $DT_l[b]$ finishes before the start of data-transfer instance $DT_k[a]$ and data transfers DT_k and DT_l are allocated to the same communication link.*

- $\Phi_{k\,l}[a,\ b]$ *is True if DT_k and DT_l are not allocated to the same communication link or at least one of them is* local *(intra-processor data transfer).*

3.3.4.3 Properties of $\alpha_{i\,j}[a,\ b]$ and $\Phi_{i\,j}[a,\ b]$

The following theorems and properties are a consequence of the use of a full static mapping with constant *initiation interval* T_I.

3.3.4.3.1 $\alpha_{i\,j}[a.\ b]$

Theorem 3.4 $\alpha_{i\,j}[a.\ b] \Longrightarrow \alpha_{i\,j}[a.\ b+k]$ *for any positive integer k.*

Proof:

i. S_i and S_j are not allocated to the same processor. The theorem is trivial from Definition 3.2.

ii. $S_i[a]$ finishing before $S_j[b]$ implies that $S_i[a]$ finishes before $S_j[b+k]$ which is executed $k * T_I$ time units after $S_j[b]$ on the processor p_x where both S_i and S_j are allocated. □

Theorem 3.5 $\alpha_{i\,j}[a.b] = \alpha_{i\,j}[a+k.b+k]$ *for any integer k.*

Proof:

It is a consequence of the *full static* mapping with constant T_I hypothesis, i.e. the relative scheduling of task instances is time invariant. □

The following two properties are derived from Definition 3.2.

Property 3.1 $\alpha_{i\,j}[a.b] = 1 - \alpha_{j\,i}[b.a]$ *if S_i and S_j are allocated to the same processor.*

Property 3.2 $\alpha_{i\,j}[a.b] = \alpha_{j\,i}[b.a]$ *if S_i and S_j are not allocated to the same processor.*

The following lemma shows that only variables of the type $\alpha_{i\,j}[0, k]$, where k is an integer, are sufficient to ensure non-overlap of tasks assigned to the same processor.

Lemma 3.1 *For each pair of tasks S_i and S_j having a common non-empty set of processors $P_{i\,j}$ able to execute both ($P_{i\,j} = P_{S_i} \cap P_{S_j}$), the following set of linearly independent Boolean variables, when present in the MILP model, is sufficient to ensure non-overlap between any instances of S_i and S_j.*

$$\mathcal{B}_{\alpha_{i\,j}} = \{\alpha_{i\,j}[0, n-1], \ldots, \alpha_{i\,j}[0,1], \alpha_{i\,j}[0,0], \alpha_{i\,j}[1,0], \ldots, \alpha_{i\,j}[n-1,0]\}$$

where n is the overlapping factor.

Proof:

i. By Theorems 3.4 and 3.5 and Properties 3.1 and 3.2, the variables in $\mathcal{B}_{\alpha_{i\,j}}$ are linearly independent, i.e., there are not enough constraints to make them linearly dependent. Figure 3.6 gives an example of linear independence among type α variables.

ii. Sufficiency.

    By Theorem 3.1, the variables in $\mathcal{B}_{\alpha_{i\,j}}$ cover all possible overlaps between tasks inside the *task overlap detection window*.

iii. They may be not necessary.

    If $S_j[k]$ is data-dependent on $S_i[0]$, $\alpha_{i\,j}[0, k]$ is always true. This can be found at *compile time* by the MILP model compiler. There is no need for the variable $\alpha_{i\,j}[0, k]$. □

Lemma 3.2 $\alpha_{i\,j}[0, k] = \alpha_{i\,j}[-k, 0]$, *where k is an integer.*

Proof:

It follows from Theorem 3.5. □

Lemma 3.3 $\alpha_{i\,j}[0, n-1] \Longleftarrow \ldots \Longleftarrow \alpha_{i\,j}[0,1] \Longleftarrow \alpha_{i\,j}[0,0] \Longleftarrow \alpha_{i\,j}[1,0] \Longleftarrow \ldots \Longleftarrow \alpha_{i\,j}[n-1, 0]$

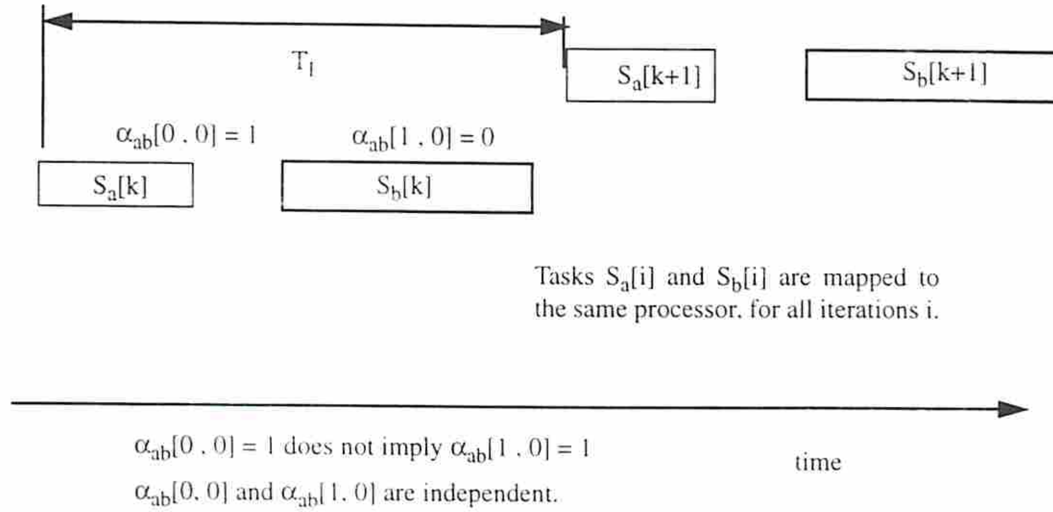


Figure 3.6: Independence between α variables

Proof:

From Lemma 3.2

$$\alpha_{i\ j}[n-1, 0] = \alpha_{i\ j}[0, -n+1]$$

From Theorem 3.4

$$\alpha_{i\ j}[0, -n+1] \Longrightarrow \alpha_{i\ j}[0, -n+2]$$

By induction and from Lemma 3.2

$$\alpha_{i\ j}[0, n-1] \Longleftarrow \ldots \Longleftarrow \alpha_{i\ j}[0, 1] \Longleftarrow \alpha_{i\ j}[0, 0] \Longleftarrow \alpha_{i\ j}[1, 0] \Longleftarrow \ldots \Longleftarrow \alpha_{i\ j}[n-1, 0]$$

□.

3.3.4.3.2 $\Phi_{i\ j}[a, b]$

Theorem 3.6 $\Phi_{i\ j}[a, b] \Longrightarrow \Phi_{i\ j}[a, b+k]$ *for any positive integer* k.

Proof:

Similar to Theorem 3.4. □

Theorem 3.7 $\Phi_{i\,j}[a, b] = \Phi_{i\,j}[a+k, b+k]$ *for any integer* k.

Proof:

Similar to Theorem 3.5. □

The following three properties are derived from Definition 3.3.

Property 3.3 $\Phi_{i\,j}[a, b] = 1 - \Phi_{j\,i}[b, a]$ *if* S_i *and* S_j *are allocated to the same processor.*

Property 3.4 $\Phi_{i\,j}[a, b] = \Phi_{j\,i}[b, a]$ *if* S_i *and* S_j *are not allocated to the same processor.*

Property 3.5 *Only variables of the type* $\Phi_{i\,j}[0, k]$, *where* k *is an integer, are needed.*

Lemma 3.4 *For each pair of data transfers* $DT_i = S_u[0] \longrightarrow S_v[k]$ $(k > 0)$ *and* $DT_j = S_r[0] \longrightarrow S_s[l]$ $(l > 0)$ *having a common non-empty set of communication-links* $CL_{i\,j}$ *able to carry both, the following set of linearly independent Boolean variables, when present in the MILP model, is sufficient to ensure non-overlap between any instances of* DT_i *and* DT_j.

$$\mathcal{B}_{\Phi_{i\,j}} = \{\Phi_{i\,j}[0, m-1], \ldots, \Phi_{i\,j}[0, 1], \Phi_{i\,j}[0, 0], \Phi_{i\,j}[1, 0], \ldots, \Phi_{i\,j}[m-1, 0]\}$$

Where $m = n + Diff_{max} - 1$, n *is the overlapping factor and* $Diff_{max}$ *is the maximum difference of subscripts among data transfers.*

Proof:

Similar to Lemma 3.1. □

Lemma 3.5 $\Phi_{i\,j}[0, k] = \Phi_{i\,j}[-k, 0]$, *where* k *is an integer.*

The proof follows from Theorem 3.7. □

Lemma 3.6 $\Phi_{i\,j}[0, m-1] \Longleftarrow \ldots \Longleftarrow \Phi_{i\,j}[0, 1] \Longleftarrow \Phi_{i\,j}[0, 0] \Longleftarrow \Phi_{i\,j}[1, 0] \Longleftarrow \ldots \Longleftarrow \Phi_{i\,j}[m-1, 0]$

The proof is similar to Lemma 3.3. □

3.3.5 Constants and Constraints of the MILP Model

The macro-pipelined model requires a set of constants including execution time, volume of data transferred, delay for remote data transfer per unit of data and delay for local data transfer per unit of data, as defined in Section 1.2.1.

The main constraints in the MILP model cover processor selection, remote data transfer, processor allocation, link allocation, task allocation, task completion, task start, data transfer completion, data transfer start, non-overlap of tasks in a processor (processor usage exclusion), non-overlap of data transfers in a communication link (link usage exclusion) and non-overlap of data transfers over a shared bus (bus usage exclusion). Additional constraints were added to ensure numerical convergence of the MILP model.

3.3.5.1 Pruning Unnecessary Constrains

A large number of the constraints of the MILP model are made of non-overlap (usage-exclusion) relations. In order to decrease the CPU time to compute the optimal solution some effort should be made to prune the number of these relations. This can be done by considering the data dependencies in the task-flow graph (TFG). Tasks that are connected by a directed path in the TFG are necessarily executed in a sequential order so there is no possibility of overlap. A simple transitive-closure algorithm [19] can be used to detect tasks that never overlap. The same applies for non-overlap on communication links, i.e., sequentiality can be extracted considering a dual graph of the TFG, where tasks become hyperedges and data transfers become nodes.

3.3.6 Summary of the MILP Model for Macro-Pipelined design

The MILP model for macro-pipelined design is summarized below. [3.1]

i . System latency

$$\forall S_x \; T_{SYS} \geq T_{SE}(S_x[0]) \tag{3.21}$$

[3.1] The naming convention of Section 1.2.1 is adopted.

ii. Processor selection

$$\beta_{p_x} = \bigvee_i \sigma_{i\,x} = \sigma_{i_1\,x} \vee \sigma_{i_2\,x} \vee ... \tag{3.22}$$

iii. Task to processor allocation

$$\sum_{p_x \in P_{S_i}} \sigma_{i\,x} = 1 \tag{3.23}$$

iv. Data transfer to communication-link allocation

$$\sum_l \eta_{r\,l} = \gamma_{i\,j} \tag{3.24}$$

v. Data transfer end

$$T_{CE}(DT^i_{in_k(S_i)}[0]) \leq T_{SS}(S_i[0]) + r^i_{in_k} \sum_{p_x \in P_{S_i}} \omega_{i\,x}\sigma_{i\,x} \tag{3.25}$$

vi. Data transfer start

$$T_{CS}(DT^l_{out_k(S_i)}[0]) \geq T_{SE}(S_i[0]) - r^i_{out_k} \sum_{p_x \in P_{S_i}} \omega_{i\,x}\sigma_{i\,x} \tag{3.26}$$

vii. Task completion

$$T_{SE}(S_i[0]) = T_{SS}(S_i[0]) + \sum_{p_x \in P_{S_i}} \omega_{i\,x}\sigma_{i\,x} \tag{3.27}$$

viii. Data transfer completion

$$T_{CE}(DT_l[0]) = T_{CS}(DT_i[0]) + Delay(DT_i) \tag{3.28}$$

$Delay(DT_i)$ is given by Equation 1.17.

3.3.6.1 Tasks Overlap Avoidance Constraints

For each pair of tasks S_i and S_j such that $P_{S_i} \cap P_{S_j} \neq \emptyset$
For $-n+1 \leq k \leq n-1$, $p_x \in P_{S_i} \cap P_{S_j}$

$$\sigma_{ix}\sigma_{jx}T_{SE}(S_i[k]) \leq \alpha_{ij}[k,0]\sigma_{ix}\sigma_{jx}T_{SS}(S_j[0]) \tag{3.29}$$

$$\sigma_{ix}\sigma_{jx}T_{SE}(S_j[0]) \leq (1-\alpha_{ij}[k,0])\sigma_{ix}\sigma_{jx}T_{SS}(S_i[k]) \tag{3.30}$$

which are linearized in the following way:

$$T_{SE}(S_i[0]) \leq T_{MAX}(3 - \alpha_{i\,j}[0,k] - \sigma_{i\,x} - \sigma_{j\,x}) + T_{SS}(S_j[0]) + k * T_I \tag{3.31}$$

$$T_{SE}(S_j[0]) \leq T_{MAX}(2 - \alpha_{i\,j}[0,k] - \sigma_{i\,x} - \sigma_{j\,x}) + T_{SS}(S_i[0]) - k * T_I \tag{3.32}$$

where T_{MAX} is a sufficient large constant.

3.3.6.2 Data-Transfer Overlap Avoidance Constraints

For each pair of data transfers $DT_r = S_a \longrightarrow S_b$ and $DT_s = S_c \longrightarrow S_d$ such that they have a non-empty set $L_{r\,s}$ of common data links to which both can be allocated, for $-m+1 \leq k \leq m-1$, $m = n + Diff_{max} - 1$, $l \in L_{r\,s}$:

$$\gamma_{ij}\gamma_{rs}T_{CE}(S_i[k], S_j[k+u]) \leq \Phi_{ijursv}[k,0]\gamma_{ij}\gamma_{rs}T_{CS}(S_r[0], S_s[v]) \tag{3.33}$$

$$\gamma_{ij}\gamma_{rs}T_{CE}(S_r[0], S_s[v]) \leq (1 - \Phi_{ijursv}[k,0])\gamma_{ij}\gamma_{rs}T_{CS}(S_i[k], S_j[k+u]) \tag{3.34}$$

which are linearized in the following way:

$$T_{CE}(DT_r[0]) \leq T_{MAX}(3 - \Phi_{r\,s}[0,k] - \eta_{r\,l} - \eta_{s\,l}) + T_{CS}(DT_s[0]) + k * T_I \tag{3.35}$$

$$T_{CE}(DT_s[0]) \leq T_{MAX}(2 - \Phi_{i\,j}[0,k] - \eta_{r\,l} - \eta_{s\,l}) + T_{CS}(DT_r[0]) - k * T_I \tag{3.36}$$

3.3.6.3 Additional Constraints to Ensure Numerical Convergence of the MILP Model

The following constraints were added to the model to allow convergence to an optimal solution. In the non-periodic case, they are implicit in the mathematical model and therefore there is no need to include them in the constraint system.

$$T_I \leq T_{SYS} \tag{3.37}$$

$$(n + Diff_{max} - 1) * T_I \leq T_{MAX} \tag{3.38}$$

$$(n + Diff_{max} - 1) * T_{SYS} \leq T_{MAX} \tag{3.39}$$

$$T_{SYS} + (n-1) * T_I \leq T_{MAX} \tag{3.40}$$

For the *macro-pipelined* case an upper bound for T_{MAX} is given by

$$T_{MAX} = n * \sum_i \Omega(S_i) + (n + Diff_{max} - 1) * \sum_j \Omega(DT_j) + 1 \tag{3.41}$$

3.3.7 Software Implementation

The tool *pipelined-SOS* was developed based on the MILP described in this section. It has around 5,000 lines of C code. Experimental results with *pipelined-SOS* are described at the end of the chapter.

3.4 Retiming and System-Level Design

3.4.1 Motivation

The approach used in the MILP model for macro-pipelined design with a finite horizon described in Section 3.3 does not consider any *behavior-preserving* transformations to be applied to the *task-flow graph* specifying the application to be mapped to a dedicated multiprocessor. The *pipelined-SOS* tool, which is based on the MILP model described in 3.3 assumes the task-flow graph as an input parameter and it makes no attempt to find an equivalent task-flow graph that would allow a less expensive implementation. *Pipelined-SOS* is also limited to a preset maximum number of concurrent iterations equal to $2n-1$ where n is the *overlapping factor*, i.e. a *finite horizon* approach.

3.4.2 Retiming as a Task-Flow Graph Transformation

An alternative method using *retiming* [75] is presented here. This approach allows task-flow graph transformations by means of changes of the subscript differences of the data transfers between tasks, and places no limit on the number of iterations to be handled in parallel. This method has, however, the drawback of not allowing a straightforward way to express the overlap avoidance constraints. Note also that *retiming* is used as a task-flow-graph level (system-level) transformation which is a higher level of abstraction than the gate-level formulation proposed by Leiserson and Saxe [75].

The proposed *retiming* integer-linear programming (ILP) based formulation represents time events in a *discrete grid* and it assumes no sharing of processors by multiple tasks in order to keep the size of the resulting ILP model reasonable. Communication costs are also not present in the present version. Nevertheless, this work

is a good basis for understanding how retiming can be incorporated into the framework of a ILP model for design of macro-pipelined heterogeneous multiprocessors. Other applications of this approach would include the realm of high-level synthesis, where the processors and tasks are respectively substituted by functional modules (e.g., multipliers and adders) and operations (e.g. add and multiplication). [3.2]

3.4.3 Mathematical Model

The model assumes that the task-flow graphs are composed of vertices (*tasks*) and edges (*data transfers*) with associated weights, where a task only starts to execute after the arrival of all its inputs and a weight $w(e)$ in an edge e connecting vertex S_i and S_j denotes that there is a data transfer from task instance $S_i[k - w(e)]$ to task instance $S_j[k]$ for all iterations k. using the same notation of previous sections, i.e. $w(e)$ is the subscript difference of the data transfer. By changing the edge weights of a task-flow graph through *retiming* it may be possible to find transformed (*retimed*) TFGs that would lead to designs with lower implementation costs.

3.4.3.1 The Function $d : S_i \longrightarrow \mathcal{R}$ - Computation Time of a Task

The *propagation delay* $d(S_i)$ [75] of a vertex S_i in this model corresponds to the computation time of the task S_i. The assumption of time invariance of allocation over different iterations is implicit in the mathematical model, i.e., task instances $S_i[k]$ and $S_i[k+j]$, $j \neq 0$, are allocated to the same processor because they correspond to the same vertex in the task-flow graph.

The model outlined above can be formalized in the following way. A task-flow graph to be executed in a macro-pipelined way can be represented by a directed graph $TFG =< V, E, d, w >$ such that V is the set of vertices (*tasks*), E is the set of edges (*data transfers*), d is a function giving the computation (execution) time of each vertex (task) and w is a function giving the weight of each edge before retiming [75]. Based on these assumptions, the main theorem of Leiserson and Saxe for *retiming* [75] is rewritten, having in mind its use in system-level design.

[3.2] A similar model using retiming and algorithm/architecture selection for DSP computations specified by synchronous data-flow graphs is given in one of our recent papers [23].

3.4.3.2 Leiserson and Saxe's Retiming Theorem

Definition 3.4

$T_I(TFG_r)$ is the initiation interval after retiming

Definition 3.5

W_p is the sum of edge weights in the path p before retiming

Definition 3.6

D_p is the sum of computation (execution) times of the vertices (tasks) in the path p

Definition 3.7

$W(S_u, S_v) = \min\{W_p$: p is a path from $S_u = source(p)$ to $S_v = sink(p)\}$

Definition 3.8

$D(S_u, S_v) = \max\{D_p$: p is a path from $S_u = source(p)$ to $S_v = sink(p)$ such that $W_p = W(S_u, S_v)\}$

Theorem 3.8

Let $TFG = < V, E, d, w >$ be a task-flow graph [75], let c be an arbitrary positive real number, and r be a function from V to $\mathcal{Z}$ (set of integer numbers), then r is a legal retiming of TFG such that $T_I(TFG_r) \geq c$ where (TFG_r is the task-flow graph after retiming) if and only if

(a) $r(S_u) - r(S_v) \leq w(e)$ for every edge $e = S_u \rightarrow S_v$ of TFG, and

(b) $r(S_u) - r(S_v) \leq W(S_u, S_v) - 1$ for all vertices $S_u, S_v \in V$ such that $D(S_u, S_v) \geq c$

Where

S_u *and* S_v *denotes tasks (vertices) of the TFG.* Retiming does not change the structure of the task-flow graph, i.e. vertices and edges are kept. The edge weights are the only things allowed to change. The weight $w_r(e)$ of an edge $e = S_u \rightarrow S_v$ after retiming is given by the expression below [75], where $w(e)$ is the weight before retiming.

$$w_r(e) = w(e) + r(S_v) - r(S_u) \tag{3.42}$$

Proof: Similar to proof in reference [75]. □

3.4.3.3 An ILP Model for Simultaneous Retiming and Processor Selection

We reformulated the result of Leiserson and Saxe [75] by developing an ILP model suitable for system-level design in the following way, where the goal is to find a valid *retiming* of a TFG such that the transformed (*retimed*) TFG would lead to a design of lower implementation cost than ones derived directly from the original TFG.

The function w_r defines a task-flow graph transformation where the new task-flow graph will have a different set of data-transfer subscript differences , but it will have the same *behavior* as the original task-flow graph.

3.4.3.3.1 Edge Weights after Retiming

Assuming that the designer expects that no edge should have a weight more than w_M, where

$$w_M \geq max_e(w(e)) , e \in E \tag{3.43}$$

$w_r(e)$ can be expressed as

$$w_r(e) = \sum_{i=1}^{w_M} z_{e,i} \leq w_M \quad , z_{e,i} \in \{0, 1\} \tag{3.44}$$

$$z_{e,i} \geq z_{e,i+1} \quad , i = 1, \ldots, w_M - 1 \tag{3.45}$$

$z_{e,i} = 1$, $1 \leq i \leq w_M$ denotes that at least $w_M - i + 1$ delays will be present in edge e after retiming, where the total number of delays in e after retiming is given by w_r. Similarly, $z_{e,i} = 0$, $1 \leq i \leq w_M$ means that at most $w_M - i$ delays will be present in edge e after retiming.
The constraints 3.44 and 3.45 guarantee that condition (a) of Theorem 3.8 will always be satisfied.

3.4.3.3.2 Computation Time of a Task

The computation time d_{S_u} of a task (vertex) S_u is given by

$$d_{S_u} = \sum_i d_{S_u,i} \sigma_{S_u,i} , \sigma_{S_u,i} \in \{0, 1\} \tag{3.46}$$

Where

$$\sum_i \sigma_{S_u,i} = 1 \tag{3.47}$$

3.4.3.3.3 Cost of a Task to Processor Assignment

In a similar way, the cost $Cost_{S_u}$ of a task (vertex)

S_u is given by:

$$Cost_{S_u} = \sum_i C_{S_u,i} \sigma_{S_u,i} \tag{3.48}$$

Where $C_{S_u,i}$ and $d_{S_u,i}$ are the *cost* and *execution time* when mapping task S_u to a processor of type i.

3.4.3.3.4 Functions D_p and W_p

Let be p a path from task (vertex) S_u to task (vertex) S_v ($u \xrightarrow{p} v$). D_p and W_p are defined as:

$$D_p = \sum_{S_a \in p} d_{S_a} = \sum_{S_a \in p} \sum_i d_{S_a,i} \sigma_{S_a,i} \tag{3.49}$$

$$W_p = \sum_{e \in p} w(e) \tag{3.50}$$

W_p can be precomputed. However, D_p depends on the choice of which processors are assigned for the tasks in the path.

3.4.3.3.5 Circular Walk of a TFG

Due to the fact that MILP/ILP solvers usually either handle only binary ($\{0, 1\}$) integer variables or have a faster convergence on average when the integer variables are restricted to the $\{0, 1\}$ domain, the proposed ILP model express retiming by changes in the assignment of variables $z_{e,i}$, $1 \leq i \leq w_M$, instead of using the retiming function r. This avoids the existence of integer variables whose domains are not binary. However, it is necessary to enforce that the sums of delays in all circular paths of the task-flow graph have the same value before and after retiming, as discussed below.

For every path p connecting task (vertex) S_u to task (vertex) S_v.

$$r(S_u) - r(S_v) = \sum_{e \in p} w(e) - \sum_{e \in p} \sum_{i=1}^{w_M} z_{e\,i} \tag{3.51}$$

$$r(S_u) - r(S_v) = W_p - \sum_{e \in p} \sum_{i=1}^{w_M} z_{e\,i} \tag{3.52}$$

The expression above can be generalized to a *circular walk* p_c in TFG. A *circular walk* is defined in the following way.

Definition 3.9 *There is a* circular walk *starting in task (vertex)* S_u *if and only if there is a corresponding cycle in the undirected graph* TFG_o *derived from* TFG *by not considering the directions of the edges of* TFG *and assuming the same weight values* $(w(e))$ *as in the* TFG.

$$W_{p_c} = \sum_{e \in p_c} sign(e, p_c) w(e) - \sum_{e \in p_c} sign(e, p_c) \sum_{i=1}^{w_M} z_{e\,i} \tag{3.53}$$

$$\forall \; circular\ walk\ p_c\ in\ TFG.\ W_{p_c} = 0 \tag{3.54}$$

Where

Definition 3.10 *The* $sign(e, p_c)$ *function*

i. $sign(e, p_c) = +1$ *if* e *has the same direction of* p_c

ii. $sign(e, p_c) = -1$ *if* e *has direction opposite to* p_c

3.4.3.3.6 Fundamental Cycles of a TFG

The maximum size set of linearly independent equations involving circular walks in TFG can be determined by finding a spanning tree G_o of the TFG_o. Each edge $e_* = (S_u, S_v)$ not in the spanning tree will define a unique cycle made by the edge itself and the path between S_u and S_v in the spanning tree. where the direction of the cycle is given by the direction of e_* (see Figure 3.7). Each cycle corresponds to a circular walk in TFG.

Let a *fundamental cycle* c_{e_i} of TFG_o be defined as the cycle formed by a single edge e_i not in the spanning tree G_o and the path connecting $source(e_i)$ to $sink(e_i)$ by using only edges in the spanning tree G_o. For each pair of *fundamental cycles* c_{e_i} and c_{e_j}, $e_i \neq e_j$, we have $c_{e_i} \neq c_{e_j}$, i.e. they differ in at least one edge. For each cycle c_{e_i}, the sum of edge weights of the correspondent circular path in TFG is given by $W_{c_{e_i}}$, where by definition:

$$W_{c_{e_i}} = 0 \tag{3.55}$$

Theorem 3.9 *The set of* fundamental cycles *corresponds to a set SW_{fund} of linearly independent equations*

Proof: Each fundamental cycle corresponds to a equation similar to Equation 3.55. Each pair of equations differs at least by a term $w(e_k)$, where e_k is a edge not in the spanning tree. In other words,

$$SW_{fund} = \{y \mid y \; is \; an \; equation \; of \; type \; W_{c_{e_i}} = 0 \; where \; c_{e_i} \; is \; a \; fundamental \; \mathbf{cycle}\} \tag{3.56}$$

As the the edge e_k is unique to the *fundamental cycle* c_{e_k}, the equation $W_{c_{e_k}} = 0$ is linearly independent of the other equations in SW_{fund}. □

Theorem 3.10 *The set SW_{fund} of linearly independent equations has maximal size.*

Proof: This set has maximal size because any *non-fundamental* cycle c corresponds to a circular walk with an equation $W_{p_c} = 0$ that is a linear combination of the equations in the set (see Figure 12). The size of this set will be given by $O(|E| - |V| + 1)$, where $|E| \geq |V| - 1$ (G_o is a connected graph). In the case of the task-flow graph is a tree, there will be no circular walks equations, i.e. the size of SW_{fund} is zero. □

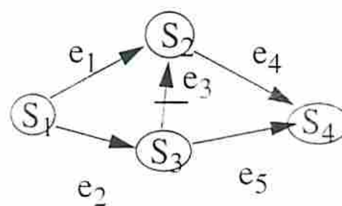
3.4.3.3.7 Linearization of the Model

Condition (b) of Theorem 3.8 is related to particular paths $p(S_u \longrightarrow S_v) \in TFG$ such that D_p is maximum among all paths from S_u to S_v with minimum W_p. As $w(e)$, $e \in E$, is known, it is possible to find the set of paths $P(S_u, S_v) = \{p \mid W_p \; is \; minimum\} = \{p \mid W_p = W(S_u, S_v)\}$ by executing the *all shortest paths algorithm*[19] on TFG. However it is not possible to decide which path $p \in P(S_u, S_v)$ will have maximum D_p before the particular processor assignment of each task $S_{v_o} \in p$ is decided. Therefore condition (b) needs to be checked for each path $p \in P(S_u, S_v)$. This can be accomplished to by means of the following linearization, assuming the *initiation interval lower bound* c to be a positive integer, and taking advantage of the fact that time is represented by a *discrete grid* in the proposed ILP model.

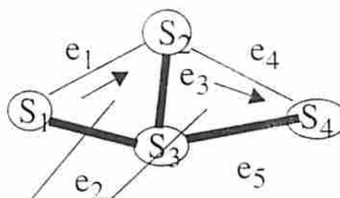
$\forall p \in P(S_u, S_v)$

$$D_p > c \Longleftrightarrow D_p \geq c + 1 \tag{3.57}$$

TFG



Spanning Tree
G_o



$G_0 = \{ e_2, e_3, e_5 \}$

Fundamental cycles:

$c_{e_1} = (e_1, e_3, e_2)$

$c_{e_4} = (e_3, e_4, e_5)$

$p_c = (e1, e4, e5, e2)$

p_c *is a circular walk but not a fundamental cycle*

$Wp_c = Wc_{e_1} + Wc_{e_2} = 0$

Figure 3.7: Spanning tree of a TFG and fundamental cycles

Introducing a variable $\alpha_p \in \{0, 1\}$ to indicate that D_p is either greater than c ($\alpha_p = 1$) or less or equal to c ($\alpha_p = 0$). Equation 3.57 can be linearized by

$$D_p \geq c + 1 - (1 - \alpha_p)\Omega_{\alpha_p} \tag{3.58}$$

$$-D_p \geq -c - \alpha_p \Omega_{\alpha_p} \tag{3.59}$$

Where

Ω_{α_p} is a sufficient large constant, $\Omega_{\alpha_p} \geq max(c + 1 - D_p, D_p - c)$ for all possible choices of processors.

In a similar way, the variable $\beta_{S_u,S_v} \in \{0, 1\}$ is introduced to indicate if either $W(S_u, S_v)$ is greater than $r(S_u) - r(S_v)$ ($\beta_{S_u,S_v} = 1$) or $W(S_u, S_v)$ is less or equal to $r(S_u) - r(S_v)$ ($\beta_{S_u,S_v} = 0$).
$\forall p \in P(S_u, S_v)$

$$r(S_u) - r(S_v) \leq W(S_u, S_v) - 1 + (1 - \beta_{S_u,S_v})\Omega_{\beta_{S_u,S_v}} \tag{3.60}$$

$$-r(S_u) + r(S_v) \leq -W(S_u, S_v) + \beta_{S_u,S_v}\Omega_{\beta_{S_u,S_v}} \tag{3.61}$$

Where $\Omega_{\beta_{S_u,S_v}}$ is a sufficient large constant.

By Equation 3.52. and as $p \in P(S_u, S_v) \iff W(S_u, S_v) = W_p$, the following tighter constraints are derived:

$$0 \leq \sum_{e \in p} \sum_{i=1}^{w_M} z_{e,i} - \beta_{S_u,S_v} \tag{3.62}$$

$$\sum_{e \in p} \sum_{i=1}^{w_M} z_{e,i} \leq \beta_{S_u,S_v} * w_M * |\{e, e \in p\}| \tag{3.63}$$

Where $|A|$ is the number of elements of the set A.

Inequality 3.63 leads to the following additional constraint:

$$\forall e \in p, \; z_{e,i} \leq \beta_{S_u,S_v}, \; 1 \leq i \leq w_M \tag{3.64}$$

From the development above, condition (b) of Theorem 3.8 can be expressed by the following constraint:

$$\forall p \in P(S_u, S_v), \; \alpha_p \leq \beta_{S_u,S_v} \tag{3.65}$$

Constraints (3.58) and (3.58) can be expressed in a tighter form as

$$D_p \geq c + 1 - (1 - \alpha_p)\Omega^0_{\alpha_p} \tag{3.66}$$

$$-D_p \geq -c - \alpha_p \Omega^1_{\alpha_p} \tag{3.67}$$

Where

$$\Omega^0_{\alpha_p} = max(0, -c + D^{max}_p) \tag{3.68}$$

$$\Omega^1_{\alpha_p} = max(0, c + 1 - D^{min}_p) \tag{3.69}$$

Definition 3.11 *D^{max}_p and D^{min}_p*
D^{max}_p is the maximum *possible value of D_p*
D^{min}_p is the minimum *possible value of D_p*

By inspection

$$D^{max}_{path}(u, v, p) \leq \sum_{S_{vo} \in p} d^{max}_{S_{vo}} \tag{3.70}$$

$$D^{min}_{path}(u, v, p) \geq \sum_{S_{vo} \in p} d^{min}_{S_{vo}} \tag{3.71}$$

3.4.3.3.8 Summary of the Model

The proposed ILP formulation can be briefly described in the following way.

Given a positive integer c, upper bound on the *initiation interval*,

minimize $\sum_{u \in V} Cost_{S_u}$

subject to the constraints given by 3.43-3.50, 3.53-3.54 and 3.62-3.71,
given the Boolean variables $\sigma_{S_u,i}, z_{e,i}, \alpha_p, \beta_{S_u,S_v}$ such that $S_u, S_v \in V$, $e \in E$, $p \in P(S_u, S_v)$.

3.4.4 Software Implementation

The program *retime* implements the ILP model discussed in this section. *Retime* is coded in C, having around 6,000 lines of code. It is a ILP compiler taking as input the task-flow graph, its edge weights, and costs of task to processor assignments. The output is an ILP model to be optimized by an ILP solver. Experimental results generated by *retime* are described in the next chapter.

3.5 Experimental Results

We present here some experiments performed with the tools *pipeline-SOS* and *retime*. In all experiments, the MILP solver *BOZO* [50] was used for the optimization of the MILP and ILP models generated respectively by *pipeline-SOS* and *retime*.

3.5.1 Experiments with *pipelined-SOS*

We applied *pipelined-SOS* to the example in Figure 3.8, where the initiation interval is minimized, i.e., performance is maximized, subject to constraints on the maximum allowed cost of the solution. In other words, we investigated the trade off between *initiation interval and total cost*. The architecture was assumed to be a shared bus with *local data transfer delay* = 0 and *remote data transfer delay* = 1 (normalized values). The *volume of data transferred between tasks* is assumed to be equal to 1 for all transfers. Below we have the *Processor Types Table* (Table 3.1), giving details of the available processor cost and performance (time to execute a task in a processor). Note that normalized values should be used to allow faster execution of the MILP solver. The allocation results of our experiments are shown in the *Solutions Table* (Table 3.2). The schedule and allocation for some of the final solutions is shown in a Gannt chart in Figure 3.9.

The solutions returned by SOS for different maximum allowed cost and overlapping factors give an idea of possible trade offs. For a cost less than or equal to 14 the maximum achievable overlapping factor is n= 2, i.e., solutions for n = 3 and n = 4 have the same cost (13), initiation interval (15) and iteration latency (21). For a cost less or equal to 28 the maximum achievable overlapping factor is n = 4 which leads to a solution with an initiation interval of 5, iteration latency of 20 and

Processor types	Processor Cost	S_1	S_2	S_3	S_4
P_1	6	5	5	--[a]	15
P_2	7	15	5	10	5
P_3	3	--	15	5	--

a. Entry '--' means task cannot execute on this type of processor.

Table 3.1: Processor types - cost and performance information

cost of 22. One interesting finding is that the run time of *pipelined-SOS*, dominated by the MILP solver execution time, is not necessarily increasing with n: i.e., for a maximum cost less than or equal to 14, n= 4 has a run time lower than for n = 3.

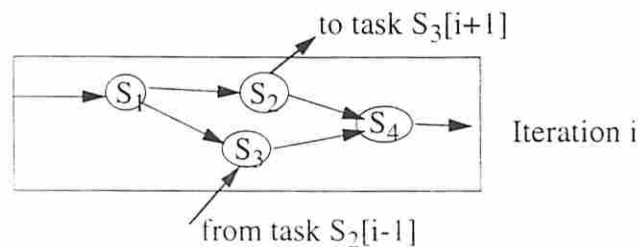


Figure 3.8: Task-flow graph

3.5.2 Experiments with *retime*

The tool *retime* was applied to the task-flow graph in Figure 3.10. Tasks S_{in} and S_{out} have a *computation time* equal to zero. They are *dummy* tasks added in order to allow all edges (data transfers) in the TFG to have a *source* and a *sink* task, as required in the ILP formulation used by *retime*.

Design	n [a]	Run Time (sec) [b]	Maximum Allowed Cost	Solution Total Cost	T_I [c]	T_{SYS}	$\#P_1$	$\#P_2$	$\#P_3$
1	2	44	28	22	9	18	2	1	1
2	2	71	16	16	10	17	1	1	1
3	2	63	14	13	15	21	1	1	0
4	3	257	28	22	7	20	2	1	1
5	3	181	14	13	15	21	1	1	0
6	4	382	28	22	5	20	2	1	1
7	4	130	14	13	15	21	1	1	0
8	5	844	28	22	5	20	2	1	1
9	5	564	14	13	15	21	1	1	0

a. Unrolling factor m for all designs is equal to 1

b. CPU-time on a Sun4-SPARC workstation.

c. Initiation interval is the performance parameter to be optimized (minimized).

Table 3.2: Solutions - Cost and Performance Trade-off

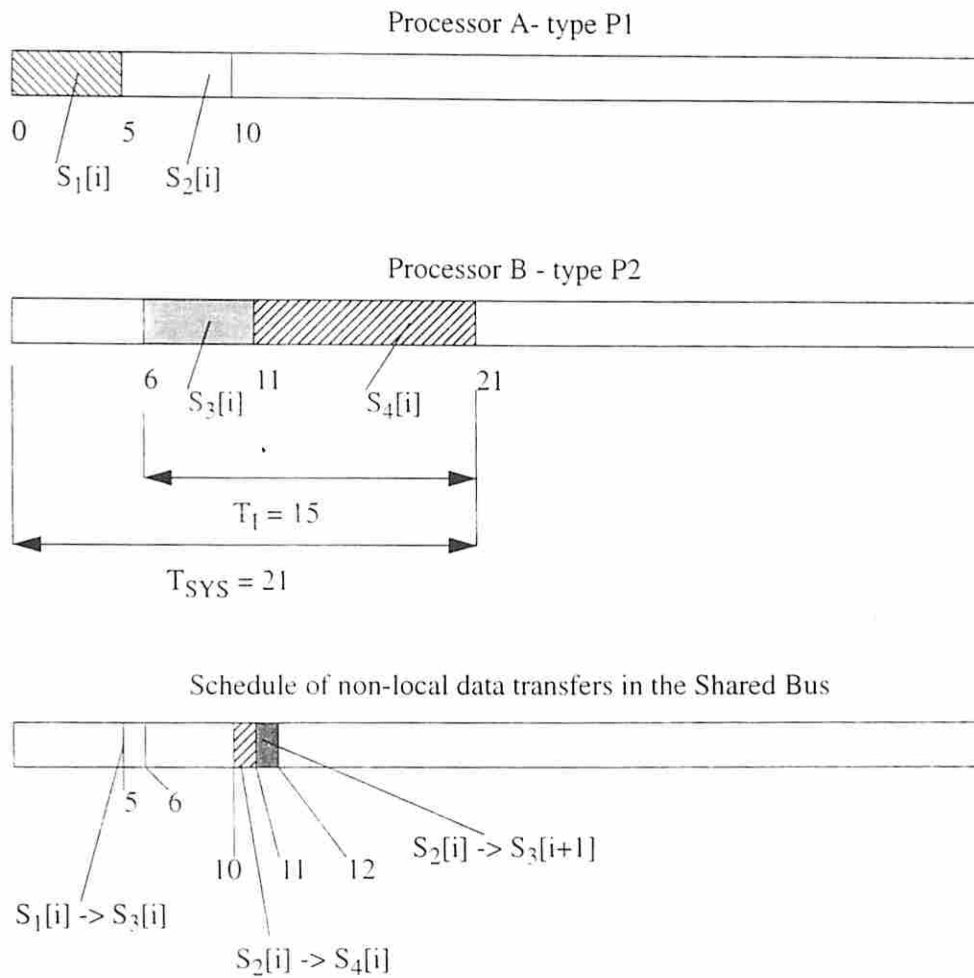


Figure 3.9: Gannt Chart for Designs 3, 5, 7 and 9

The *initiation interval* is the performance parameter to be minimized. The cost is assumed to be constrained by the designer. Table 3.3 provides the computation time of each task for each available *processor type*. Three different types of processors are assumed to be available. each one with an associated instance cost.

Ten progressively cost-constrained cases were considered in this experiment as seen in Table 3.4, that has the final *task-to-processor* mapping for each case. Designs 1 and 10 correspond respectively to the most and least expensive possible designs for the data given in Table 3.3. The final results indicated that many runs found the same *task-to-processor* mapping:

- Designs 1 and 2.
- Designs 4 and 5,
- Designs 6 and 7.
- Designs 9 and 10.

Only two transformations of the original task-flow graph in Figure 3.10 were found in the final optimal solutions as seen in Figures 3.11 and 3.12, that correspond to pipelined versions of the original task-flow graph. The computation cost (runtime) of finding each optimal design was found to be unrelated to the value of the *maximum allowed cost* as seen in Table 3.4. where Design 4 was found with a computational cost four time higher than Designs 1 and 10. the extreme points of the cost range. The original task-flow graph was not preserved in any of the final designs. what indicates the soundness of allowing task-flow graph transformation to be done concurrently with limited cost and performance optimization.

Processor Types	Processor Cost	S_{in}	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{out}
P_1	$6	—	5	—	15	5	—	15	5	—	15	—
P_2	$7	—	5	10	5	5	10	5	5	10	5	—
P_3	$3	—	15	5	—	15	5	—	15	5	—	—

Note: Dash indicates that task cannot execute on this type of processor.

Table 3.3: Processor types - cost and performance information

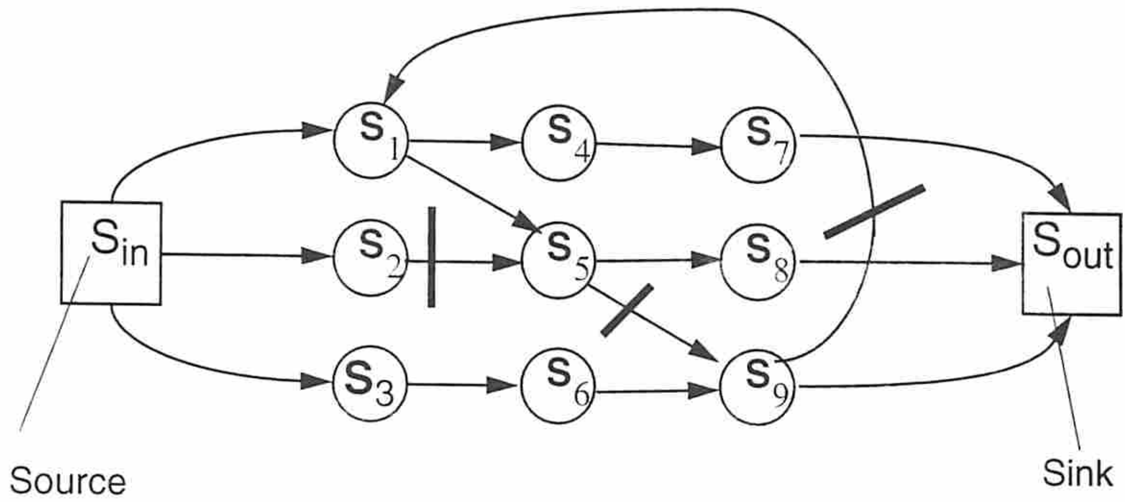


Figure 3.10: Task-flow graph before retiming

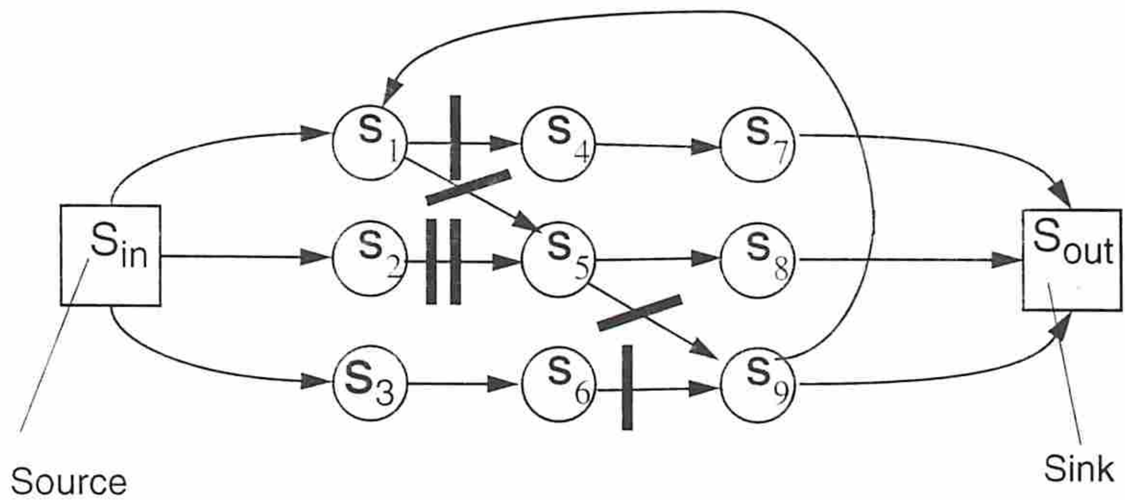


Figure 3.11: Transformed TFG after retiming - Design 1, 2, 3, 6, 7, 8, 9 and 10

Design	Run Time (sec)[a]	Maximum Allowed Cost	Solution Total Cost	T_I[b]	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
1	51.8	63	59	10	P_2	P_3	P_2	P_2	P_2	P_2	P_2	P_2	P_2
2	52.2	60	59	10	P_2	P_3	P_2	P_2	P_2	P_2	P_2	P_2	P_2
3	52.1	57	57	11	P_1	P_3	P_2	P_2	P_2	P_2	P_1	P_2	P_2
4	215.9	54	51	15	P_2	P_3	P_2	P_2	P_2	P_2	P_3	P_3	P_2
5	219.7	51	51	15	P_2	P_3	P_2	P_2	P_2	P_2	P_3	P_3	P_2
6	169.8	48	42	20	P_3	P_3	P_2	P_3	P_3	P_1	P_2	P_3	P_2
7	164.3	45	42	20	P_3	P_3	P_2	P_3	P_3	P_1	P_2	P_3	P_2
8	84.1	42	42	20	P_3	P_3	P_1	P_2	P_3	P_2	P_3	P_3	P_2
9	89.4	39	36	30	P_3	P_3	P_1	P_3	P_3	P_1	P_3	P_3	P_1
10	51.3	36	36	30	P_3	P_3	P_1	P_3	P_3	P_1	P_3	P_3	P_1

a. CPU-time on a Sun4-SPARC workstation.

b. Initiation interval is the performance parameter to be optimized (minimized).

Table 3.4: Solutions - Cost and Performance Trade-off

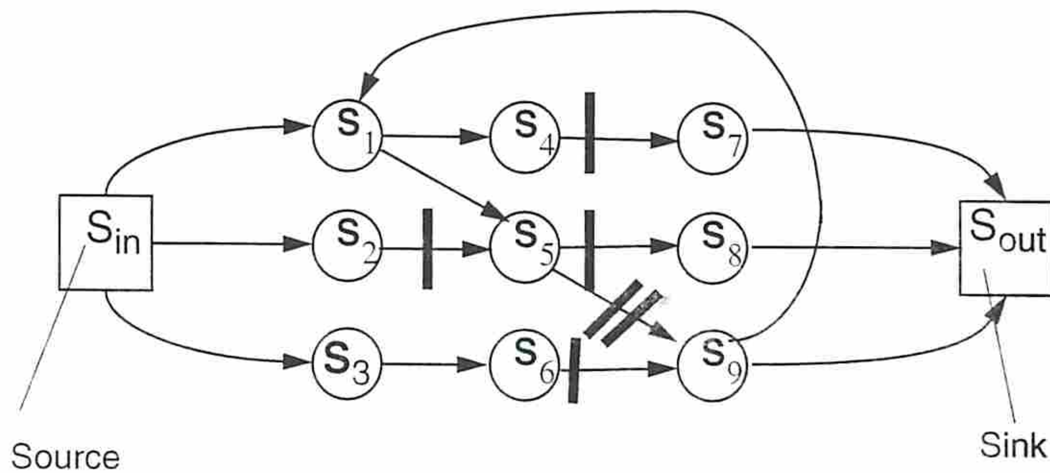


Figure 3.12: Transformed TFG after retiming - Designs 4 and 5

3.6 Summary of the Chapter

An MILP model for design of macro-pipelined of heterogeneous multiprocessors with non-negligible communication cost was introduced in this chapter. This MILP model allows the optimization of a design for a *limited horizon* in time. An ILP model for system-level design of heterogeneous multiprocessors without communication costs or resource sharing but allowing simultaneous *retiming* and processor selection is outlined in the second part of this chapter. Two software tools were developed based on described mathematical models: *pipelined-SOS* and *retime*.

Chapter 4

System of Difference Constraints and System-Level Synthesis

4.1 Motivation

A large part of the work done in this thesis is related to MILP models for design of non-periodic and periodic (macro-pipelined) application-specific heterogeneous multiprocessors. An important finding that affects many of the following chapters is that the constraint set of the MILP models can be reduced to a *system of difference constraints* once all Boolean variables (Section 1.2.1) are assigned values that do not violate any of the Boolean constraints (e.g. processor selection/allocation). This happens because once all *relative scheduling* (Section 1.2.1.11) and allocation variables are determined. the Boolean variables receive a value assignment consistent with the constraint set. The only unknown variables are the real variables associated with time events. As it can be seen from Section 1.2.1. all timing constraints have one of the forms below:

$$T_a - T_b = w^i_{a\,b} \tag{4.1}$$

or

$$T_a - T_b \geq w^i_{a\,b} \tag{4.2}$$

Where $T_a - T_b = w^i_{a\,b}$ is equivalent to

$$\begin{aligned} T_a - T_b &\leq w^i_{a\,b} \\ \textit{and}\;\; T_b - T_a &\leq -w^i_{a\,b} \end{aligned} \tag{4.3}$$

w^i_{ab} corresponds to the ith-constraint between *timing* variables T_a and T_b, and it is a function of the performance/cost parameters and the assignment given to the Boolean (integer) variables of the MILP model being used.

4.2 Timing Constraint Graph G_{constr}

Definition 4.1 *Timing Constraint Graph*

A direct weighted graph $G_{constr} = (V, E_{constr})$ called a timing constraint graph *(Figure 4.1) can be derived from the system of difference constraints involving the timing variables of the MILP model in the following way:*

i. Each timing variable T_x corresponds to a vertex $v_x \in V$, the set of vertices of G_{constr}.

ii. Each constraint $T_a - T_b \leq w^i_{ab}$ corresponds to an edge $e \in E_{constr}$, the set of edges of G_{constr}, where

$$\begin{aligned} source(e) &= v_b \\ sink(e) &= v_a \\ &and \\ weight(e) &= w^i_{ab} \end{aligned} \tag{4.4}$$

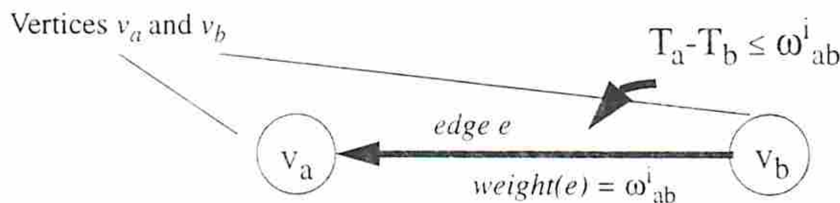


Figure 4.1: Timing Constraint graph

4.2.1 Solving the Constraint Graph G_{constr}

The value of the *timing variables* can be found by the Bellman-Ford *single source shortest-paths* algorithm [4.1] with a polynomial time complexity that is function of n and m, where n is the number of timing variables in the MILP model and m is the number of timing constraints.

Theorem 4.1 *Given a system of difference constraints $\mathcal{A}\mathcal{T} \leq \mathcal{W}$, where $\mathcal{A}$ is a matrix with each row having one 1 and one -1 and the remaining entries are equal to 0; $\mathcal{T} = (T_1, \ldots, T_n)^T)$ is a column vector* [4.2], *and its associated* timing constraint graph *is $G_{constr} = (V, E_{constr})$. There is at least a solution for the system if and only if G_{constr} has no negative-weight cycles, i.e. the sum of the edge weights in a cycle is always positive. This solution is given by the expression below:*

$$\mathcal{T} = (\delta(v_0, v_1), \ldots, \delta(v_0, v_n))^T$$

Where the vertex v_0 is added to the graph G in order to be its source vertex, i.e. there is a directed path from v_0 to all other vertices v in V, weight$(e = v_0 \longrightarrow v)$ $= 0$, and $\delta(v_i, v_j)$ is the weight of the shortest path from v_i to v_j in G, i.e. the sum of the edge weights in the shortest path. All other feasible solutions of the system will be in the form $\mathcal{T}' = (\delta(v_0, v_1) + c, \ldots, \delta(v_0, v_n) + c)^T$, where c is an arbitrary constant.

Proof: See [19].

4.3 Special cases

An important fact is that once all Boolean variables receive a consistent assignment, the cost of the final design corresponding to this assignment is also defined. The cost of the design depends only the values of the Boolean variables in the MILP formulation. What remains to be done is to optimize the performance of the design.

[4.1] Dijkstra's algorithm cannot be used because some of the edge weights may be negative.
[4.2] $T_1, \ldots, T_n$ are the timing variables of the MILP model

The sub-optimal approach of first setting the values of the Boolean variables for cost minimization followed by performance maximization will be especially helpful in developing *genetic algorithms* for system-level design, as presented in the next chapters. Below, the particular properties of the *timing constraint graph* G_{constr} are analyzed for the non-periodic and periodic (macro-pipelined) cases.

4.3.1 Non-Periodic Case

The objective is to minimize the iteration latency T_{SYS} which is given by

$$T_{SYS} = max(T_{SE}(S_1), ..., T_{SE}(S_{n_T})) \tag{4.5}$$

where $T_{SE}(SE_i)$ is the *completion time* of task S_i (Section 1.2.1), $0 \leq i \leq n_T - 1$, where n_T is the number of tasks.

The above constraint is linearized by the following constraints:

$$\forall\ tasks\ S_i \quad T_{SE}(S_i) - T_{SYS} \leq 0 \tag{4.6}$$

4.3.1.1 The Source Node of G_{constr}

Theorem 4.2 *T_{SYS} corresponds to the source vertex $v_o = v_{SYS}$ of the associated* timing constraint *graph $G_{constr} = (V, E_{constr})$, i.e., v_{SYS} (T_{SYS}), vertex of G_{constr}, serves as a source vertex required by Theorem 4.1, with no need to add a source vertex to G_{constr}.*

Proof: The following steps are needed to prove that T_{SYS} corresponds to the source vertex even for the *non-pure data-flow case*, as discussed in Section 1.2.1.15.

i. By (4.6) There is an edge from vertex v_{SYS} to vertex v_{SE_i} associated with timing variable $T_{SE}(S_i)$.

ii. For each task S_i

$$T_{SS}(S_i) - T_{SE}(S_i) \leq -\omega(S_i) = -\sum_{p_x \in P_{S_i}} \sigma_{i\,x} \omega i\,x \tag{4.7}$$

there is a edge from v_{SE_i} to v_{SS_i} (associated with timing variable $T_{SE}(S_i)$). Therefore there is a path from $v_o = v_{SYS}$ to v_{SS_i}.

iii. For a data transfer

$$DT_l = out_k(S_r) \longrightarrow in_{k'}(S_s) \tag{4.8}$$

(k-output of S_r is sent to k'-input of S_s).

$$T_{CE}(DT_l) - T_{SS}(S_s) \leq r^s_{in_{k'}} \sum_{p_x \in P_{S_s}} \omega_{i\,x}\sigma_{i\,x} \tag{4.9}$$

$$T_{CS}(DT_l) - T_{CE}(DT_l) \leq -Delay(D_l) \tag{4.10}$$

Let v_{CS_l} (v_{CE_l}) be the vertex in G_{constr} associated with timing variable T_{CS} (T_{CE}). From 4.9 and 4.10, (i) and (ii) there is a path from $v_{sys} = v_o$ to vertices v_{CE_l} and v_{CS_l}.

iv. There is not a timing variable T_x [4.3] in the MILP model in Section 1.2.1 such that

$$T_{SYS} - T_x \leq 0$$

By (i), (ii), (iii) and (iv) v_{sys} meets all the requirements to be a source vertex of G_{constr}. □

4.3.1.2 The Relaxed Graph G^{relax}_{constr}

Theorem 4.3 *By relaxing the* task computation time *equality constraint for each task S_i to the inequality*

$$T_{SE}(S_i) \geq T_{SS}(S_i) + \sum_{p_x \in P_{S_s}} \omega_{i\,x}\sigma_{i\,x} \tag{4.11}$$

and the data-transfer communication-time *equality constraint for each data-transfer DT_l to the inequality*

$$T_{CE}(DT_l) \geq T_{CS}(DT_l) + Delay(DT_l) \tag{4.12}$$

[4.3] It is assumed that tasks without successors do not have out-going data-transfers.

The associated timing constraint graph $G^{relax}_{constr} = (V, E^{relax}_{constr})$ *is a* directed acyclic graph *(DAG).*

Proof:

i. The original task-flow graph has no cycles, i.e. a task cannot be data-dependent on itself.

ii. By (i) a data-transfer $DT_l = S_r \longrightarrow S_s$ is always between two different tasks $(r \neq s)$.

iii. An edge in E^{relax}_{constr} is a member of one of the following sets:

$$E_{type\ I} = \{e \mid e = v_{SYS} \longrightarrow v_{SE_i}\}$$
$$E_{type\ II} = \{e \mid e = v_{SE_i} \longrightarrow v_{SS_i}\}$$
$$E_{type\ III} = \{e \mid e = v_{SS_i} \longrightarrow v_{CE_l}\}$$
$$E_{type\ IV} = \{e \mid e = v_{CE_l} \longrightarrow v_{CS_l}\}$$
$$E_{type\ V} = \{e \mid e = v_{CS_l} \longrightarrow v_{SE_j}\}$$

As these edges follow a reverse topologic order of the original task-flow graph, there are no cycles in G^{relax}_{constr}. □

4.3.1.3 An Algorithm for Solving G^{relax}_{constr}

Theorem 4.4 *The following algorithm, based on an variation of the Bellman-Ford* single-source shortest paths *algorithm for DAGs [19], solves* G^{relax}_{constr} *in* $O(n+m)$ *time, where* $n = |V|$ *and* $m = |E^{relax}_{constr}|$.

Algorithm 4.1 Let $Timing = \{T_1, \ldots, T_n\}$ where T_1 corresponds to T_{SYS} and T_x to vertex v_x in V.

$Topol \longleftarrow$ Topological order of G^{relax}_{constr}.
/* Set all variables to 0 */
for $i \longleftarrow 1$ to $|Timing|$ do
  $T_i \longleftarrow 0$;
/* $|Topol| = |Timing|$ */
for $i \longleftarrow 1$ to $|Topol|$ do
  $v_i \longleftarrow Topol[i]$ (I); /* i-element of $Topol$ */.

/* v_{SYS} is the first element */
for each vertex v_j such that
$\exists e \in E^{relax}_{constr} \mid e = v_i \longrightarrow v_j$
begin
if $T_j > T_i + weight(e)$ (II)
$T_j \longleftarrow T_i + weight(e)$
end
end /* for $i \longleftarrow 1$ to $|Topol|$ do */
/*All timing variables have negative values */
$T_{min} = min_{\; T_i \in Timing}(T_i)$
/*Translation of time events to positive time */
for $i \longleftarrow 1$ to $|Timing|$ do
$T_i = T_i - T_{min}$
/*Evaluation of T_{SYS} */
$T_{SYS} \longleftarrow -T_{min}$
end

Proof:

i. Statement (I) is executed $O(n)$ times. Conditional (II) is tested $O(m)$ times. The overall complexity of the algorithm is $O(n+m)$.

ii. Algorithm 4.1 corresponds to an *as late as possible* (ALAP) algorithm for assignment of positive values for the timing variables of the MILP model. □

Note that ALAP scheduling always produces optimal schedules for the critical path, although the same is not always true for non-critical paths.

Corollary 4.1 $T_{SYS}(G_{constr}) = T_{SYS}(G^{relax}_{constr})$

Proof:
Algorithm 4.1 returns the same timing assignment for tasks and data-transfers in a *critical-path* as solving G_{constr} by the general Bellman-Ford *single-source shortest paths* algorithm [19], since it is a special case of the general algorithm. □

4.3.2 Macro-Pipelined Case

This case is not as straightforward as the non-periodic case. The reason for that is that many of the equations involving timing variables will have the right side term w_{ab} as a linear function of the *initiation interval* T_I. Moreover, the constraint graph G_{constr} may be cyclic.

4.3.2.1 Feasibility Interval of T_I

By theorem 4.1 the *system of difference constraints* is feasible if and only if the *constraint graph* $G_{constr} = (V, E)$ has no cycles with negative weights. However G_{constr} is not necessarily an acyclic graph (e.g. no inter-iteration data-dependencies implies an acyclic constraint graph). The following lemmas provide lower and upper bounds on the values of T_I regardless if G_{constr} is cyclic or not.

Lemma 4.1 *A lower bound for T_I is given by $T^0_{min} = max(\Omega_{max}, \Delta_{max})$, where $\Omega_{max} = max_i(\Omega(S_i))$ and $\Delta_{max} = max_j(Delay(DT_j))$ for a particular valid assignment of the Boolean variables in the MILP model being solved.*

Proof:

It is assumed that there is non-preemptive execution of tasks and non-preemptive communication for non-local data-transfers. The task (data-transfer) instance $S_i[k]$ ($DT_j[k]$) can only start after the task (data-transfer) instance $S_i[k-1]$ ($DT_j[k-1]$) has completed. As the computation (communication) time of task (data-transfer) S_i (DT_j) is equal to $\Omega(S_I)$ ($Delay(DT_j)$, T_{min}, a lower bound for T_I, is given by

$$max(max_i(\Omega(S_i)), max_j(Delay(DT_j)) \tag{4.13}$$

where

$$\Omega(S_i) = \sum_{p_x \in P_{S_i}} \sigma_{i,x}\omega_{i,x} \tag{4.14}$$

and $Delay(DT_j)$, communication delay for data-transfer DT_j, is given by equation 1.17. □

Lemma 4.2 *For a given valid assignment of the Boolean variables of the MILP model, $T^0_{max} = T^{n=1}_{SYS}$ is an upper bound of T_I, where $T^{n=1}_{SYS}$ is the system latency for a overlapping factor $n = 1$, i.e., non-overlap between iterations.*

Proof:

$T_I \geq T^{n=1}_{SYS}$ ensures that there are not violations of overlapping constraints among task (data-transfers) instances in different iterations, i.e. $n = 1$ corresponds to the non-periodic case. □

Definition 4.2 *The interval* $[T^0_{min}, T^0_{max}]$ *is the* feasibility interval *for* T_I, *where* $T_I < T^0_{min}$ *leads to an infeasible design and* $T_I > T^0_{max}$ *corresponds to a design with an unnecessary low throughput.*

4.3.2.1.1 Upper and lower Bounds for a Cyclic G_{constr}

It is possible to find a tighter *feasibility interval* for the case where the *timing constraint graph* G_{constr} is cyclic in the following way:

Definition 4.3 W^k_{cycle} *is the sum of the edge weights of the* k^{th}*-cycle of* G_{constr}.

Property 4.1 W^k_{cycle} *is a linear function of* T_I, *i.e.* $W^k_{cycle} = a_k T_I + b_k$

For a given valid assignment of the Boolean variables of the model, the edge weights are either constants or sums of constants to a multiple of T_I, i.e. $weight(e) = w^i_{ab} = cte + k * T_I$, where k is a integer, and for the *finite-horizon* case $|k| \leq n - 1$, n is the *overlapping factor*.

Property 4.2 *If* W^k_{cycle} *is equal to a constant* b_k *then such a constant is positive.*

This follows from theorem 4.1, i.e. if $b_k < 0$ then G_{constr} is infeasible.

Definition 4.4 W_{T_I} *is the set of terms* W^k_{cycle} *that are functions of* T_I, *i.e.*

$$W_{T_I} = \{W^k_{cycle} \mid W^k_{cycle} = a_k T_I + b_k\ ,\ \ a_k \neq 0\}$$

As forall $W^k_{cycle} \in W_{T_I}$, $W^k_{cycle} = a_k T_I + b_k \geq 0$ if G_{constr} is feasible.

Property 4.3 W_{T_I} *corresponds to the following set of constraints on* T_I *supplying lower and/or upper bounds on* T_I.

$$T_I \geq \frac{-b_k}{a_k} = c_k \quad if \quad a_k > 0 \tag{4.15}$$

$$T_I \leq \frac{-b_k}{a_k} = c_k \quad if \quad a_k < 0 \tag{4.16}$$

Let be $\mathcal{C}^+$ the set of constraints such that $a_k > 0$, and $\mathcal{C}^-$ the set of constraints such that $a_k < 0$

If $\mathcal{C}^+ \neq \emptyset$ then the lower bound T^i_{min} on T_I can be evaluated as

$$T^i_{min} = max_k(c_k) \mid constraint\ k \in \mathcal{C}^+ \tag{4.17}$$

otherwise

$$T^i_{min} = -\infty \tag{4.18}$$

If $\mathcal{C}^- \neq \emptyset$ then the upper bound T^i_{max} on T_I can be evaluated as

$$T^i_{max} = min_k(c_k) \mid constraint\ k \in \mathcal{C}^- \tag{4.19}$$

otherwise

$$T^i_{max} = \infty \tag{4.20}$$

Tighter upper and lower bounds on T_I can be therefore be found.

$$T^{ii}_{min} = max(T^0_{min}, T^i_{min}) \tag{4.21}$$

$$T^{ii}_{max} = min(T^0_{max}, T^i_{max}) \tag{4.22}$$

□

Lemma 4.3 *The feasibility interval* $[T^{ii}_{min}, T^{ii}_{max}]$ *for cyclic* constraint graphs G_{constr} *can be used to detect Boolean assignments leading to unrealizable designs.*

Proof:

$T^{ii}_{min} > T^{ii}_{max}$ indicates an infeasible Boolean assignment. □

4.3.2.2 The Source Node of G_{constr}

In the periodic case, G_{constr} might be cyclic, however the same constraints as the non-periodic case will be present in periodic case. This facts lead to following finding.

Theorem 4.5 T_{SYS} *corresponds to the source vertex* $v_o = v_{SYS}$ *of the associated* timing constraint *graph* $G_{constr} = (V, E_{constr})$ *in the* macro-pipelined *(periodic) case.*

Proof:

Similar to theorem 4.2. □

4.3.2.3 The Relaxed Graph G^{relax}_{constr}

Theorem 4.6 *Differently from the non-periodic case where the* relaxed timing constraint graph G^{relax}_{constr} *is a directed acyclic graph (DAG), the relaxed graph may be cyclic.*

Proof:

It follows from the fact that the task flow graph may be cyclic, leading to a *timing constraint* graph G_{constr} that might be cyclic as well. □

4.3.2.4 An Algorithm for Solving G^{relax}_{constr}

Theorem 4.7 *The following algorithm, adapted from the* single-source shortest paths *Bellman-Ford algorithm [19], solves G^{relax}_{constr} in $O(n*m)$ time, where $n = |V|$ and $m = |E^{relax}_{constr}|$. The final solution corresponds to an ALAP timing schedule, where $T_{SYS}(G_{constr}) = T_{SYS}(G^{relax}_{constr})$*

Algorithm 4.2 Let $Timing = \{T_1, \ldots T_n\}$ where T_1 corresponds to T_{SYS} and T_x to vertex v_x in V.

/* Set all variables to 0 */
for $i \longleftarrow 1$ to $|Timing|$ do
    $T_i \longleftarrow 0$;
for $i \longleftarrow 1$ to $|Timing|$-1 do
    for each $e \in E^{relax}_{constr} \mid e = v_i \longrightarrow v_j$ do
        if $T_j > T_i + weight(e)$ (II)
            $T_j \longleftarrow T_i + weight(e)$ (III)
/*Translate time events to positive time */
$T_{lowest} = min_{\,T_i \in Timing}(T_i)$
for $i \longleftarrow 1$ to $|Timing|$ do
    $T_i = T_i - T_{lowest}$
$T_{SYS} \longleftarrow -T_{lowest}$

Proof:

Similar to Theorem 4.4 and Corollary 4.1. □

4.3.2.5 Tradeoff T_I versus T_{SYS}

T_I^{min} provides a lower bound on the problem of minimizing T_I. It may be interesting to trade a higher *initiation interval* (T_I) against a lower *system latency* (T_{SYS}). This can be done by means of choosing values of T_I from inside the feasibility interval $[T_{min}^I, T_{max}^I]$ and choosing one that optimize (minimize) a linear objective function on T_I and T_{SYS} such as:

$$f_{obj} = A * T_I + B * T_{SYS} \quad A, B \geq 0 \tag{4.23}$$

4.4 Applications

The system of difference constraints approach will be applied in the following chapter to allow the development of straightforward genetic algorithms for system-level design. This approach is also the basis for the probabilistic modeling of system design in the presence of uncertainty, as discussed in Chapter 8, and *imprecise computation* found in Chapter 7. However there are many other uses of the formalism discussed here. One use is *timing verification*, i.e. it is possible to evaluate if a given design will fulfill its *timing* requirements by expressing the timing constraints as a system of difference constraints and applying the results above.

4.5 Summary of the Chapter

The particular properties of the MILP models for system-level design for the *non-periodic* and *macro-pipelined* (*periodic*) case were investigated. They are based on the idea of making the values of the *timing* variables functions of the cost and performance parameters of the MILP model and the assignment given to its Boolean (integer) variables. Once this is done, the *timing* variables are related by a *system of difference constraints* from which *constraint graphs* are derived. Variations of the Bellman-Ford *single-source shortest path* are applied to these graphs to find the values of the *timing* variables.

Chapter 5

Genetic Algorithms Applied to System-Level Design

5.1 Motivation

The problems of task scheduling and processor allocation in a heterogeneous multiprocessor become highly complex when solved concurrently, although the former is already known to be NP-complete in a homogeneous multiprocessor [43]. Attempts to use mixed integer-linear programming (MILP) to find optimal solutions have found the computational cost [100] [22] prohibitive for large task-flow graphs. However MILP models allow a more detailed representation of hardware behavior, as for example when using a *non-pure data-flow* computational model as discussed in Section 1.2.1.15.

5.1.1 Drawbacks of Using MILP Solvers

The use of MILP solvers to optimize MILP models has other drawbacks besides high computational cost. Many times the MILP solver mistakenly converges to an invalid solution (e.g. Boolean variables having fractional values), which is probably a consequence of numerical instability due to the linearization techniques being used.

A typical problematic linearization is one used for the non-overlap of parallel tasks in a processor (mutual-exclusion) constraint as seen below for the periodic (*macro-pipelined*) case:

$$T_{SE}(S_i[k]) \leq \alpha_{ij}[k,0]\sigma_{ix}\sigma_{jx}T_{SS}(S_j[0]) \tag{5.1}$$

$$T_{SE}(S_j[0]) \leq (1 - \alpha_{ij}[k,0])\sigma_{ix}\sigma_{jx}T_{SS}(S_i[k]) \tag{5.2}$$

which is linearized to

$$T_{SE}(S_i[k]) \leq T_{SS}(S_j[0]) + (3 - \alpha_{ij}[k,0] - \sigma_{ix} - \sigma_{jx})T_{MAX} \tag{5.3}$$

$$T_{SE}(S_j[0]) \leq T_{SS}(S_i[k]) + (2 - \alpha_{ij}[k,0] - \sigma_{ix} - \sigma_{jx})T_{MAX} \tag{5.4}$$

Genetic algorithms are able to handle nonlinear constraints without need of linearization. This can potentially decrease the chances of non-convergence as well as allow handling of large task-flow graphs without a high computational cost. The price to be paid is that genetic algorithms are not guaranteed to find an optimal solution.

5.1.2 Inadequacy of Nonlinear Programming Methods

The use of nonlinear programming methods [80], such as basic descent, conjugate gradient methods and the modified Newton method, does not help to solve the problem of system-level design of heterogeneous multiprocessors. Nonlinear programming methods assume that the optimization is done over a set of continuous variables subject to a set of nonlinear differentiable constraints. The system-level synthesis of heterogeneous multiprocessors involves the use of discrete variables (0-1) to represent the allocation and *relative* scheduling information and many non-continuous constraints as well as a discrete objective function, as seen in Section 1.2.1.

5.1.3 Probabilistic Optimization

The cost/performance numbers, e.g. processor cost and computation time of a task in a given processor, are sometimes just estimates (predictions) given by the system designer or a set of software prediction tools [67] [68] [69], especially when the system is under design. GAs may be useful tools when handling the case where the cost and performance numbers are modelled by random variables, as shown Chapter 8. Probabilistic models may introduce many more nonlinearities into the synthesis problem

making it harder to achieve meaningful MILP formulations. Heuristic approaches such as GAs might be the best way to make the problem treatable.

5.1.4 GAs and Pure *Branch-and-Bound* Optimization

A fine property of properly designed genetic algorithms is that is is possible to stop the execution of a GA on iteration i before completion of the algorithm. Each one of the solutions in the population $\mathcal{P}(i)$ will be associated with a feasible solution (not always optimal) for the problem being solved. This is very different from MILP solver, where, unless it is possible to control the internal *branch and bound* process, it is necessary to wait until it converges to an optimal solution or it stops due to infeasibility of the set of constraints of the MILP model being solved.

5.1.5 Parallelism of Genetic Algorithms

Genetic algorithms also allow easy parallel implementations. The genetic operators can be applied in parallel over all the solutions in population $\mathcal{P}(i)$. Each solution or subset of solutions can be assigned to a processor. The *island model* [89] is an example of a possible framework for parallel implementations. Subsets of $\mathcal{P}(i)$ are placed in different processors of an MIMD (multiple instruction multiple data) machine. Each processor runs a instance of the genetic algorithm being parallelized. Once in a while, solutions are exchanged between processors in order to ensure *genetic diversity*.

5.2 Overview of Genetic Algorithms

Algorithm 5.1 outlines a basic framework for development of specialized genetic algorithms for *system-level design* of application-specific multiprocessors. A key point in this algorithm is the fact that the best solution of each iteration is also present in the population of the next iteration. This *elitist* approach insures that the algorithm will converge in the average case to the global optimum [113]. It should be also noted that the population $P(t)$ of iteration t is not necessarily a set

in the strict sense. $P(t)$ may be a *bag*. A *bag* is a generalization of the concept of *set*, because repeated elements (twins) are allowed in a *bag*, as for example:

$$|\{\mathcal{B} + x_i\} + x_i| = |\mathcal{B} + x_i| + 1 \tag{5.5}$$

where B is a *bag*.

5.2.1 A Basic Template of a Genetic Algorithm for System-Level Design

Algorithm 5.1 *Procedure GA {*Genetic Algorithm *}*

begin

    $t := 0$
    initialize $P(t)$ *{*initial population of feasible solutions*}*
    evaluate fitness value *of each solution* s_i *in* $P(t)$
    repeat begin
    $t := t + 1$
        $P(t) = reproduce(P(t-1)$
        $bestsolution$ = *solution with* highest fitness value *in* $P(t)$
        for $i := 1$ *to* $|P(t)|/2$ *do*
            Let x *be a random number with an uniform probability density distribution in the interval* $[0, 1]$.
            if $x \leq p_c$ *then* {*p_c *is the probability of crossover* *}
                Choose at random two solutions s_k *and* s_j *in* $P(t)$
                $(s_k^*, s_j^*) = crossover(s_k, s_j)$
                $P(t) := P(t) - \{s_k, s_j\} + \{s_k^*, s_j^*\}$
        for $i := 1$ *to* $|P(t)|$ *do*
            Let y *be a random number with an uniform probability density distribution in the interval* $[0, 1]$
            if $y \leq p_m$ {*p_m *is the probability of mutation* *}
                Pick solution s_i *in* $P(t)$
                $s_i^* := mutation(s_i)$
                $P(t) := P(t) - \{s_i\} + \{s_i^*\}$

        evaluate fitness *of each solution* s_i *in* $P(t)$
        $worstsolution$:= *solution of* worst fitness value *in* $P(t)$
        { * $P(t)$ *is a* bag *allowing replicated solutions**}
        $P(t) := P(t) - \{worstsolution\} + \{bestsolution\}$
    until $P(t)$ *converges* ({**i.e.* $P(t) = P(t-1)$ *}) *or* $t > t_{max}$

end

Algorithm 5.2 *function reproduce(P_{old} : population of feasible solutions)* {* *This function uses a roulette wheel approach* *}

begin

$P_{new} := \emptyset$ {* *new population* *}

$F_{SUM} := \sum_{i=1}^{|P_{old}|} fitness(s_i)$

{* F_{SUM} *is the sum of the fitness values of each solution in* P_{old} *}

Let be a roulette wheel RW with F_{SUM} *slots*

For each solution s_i *in* P_{old}

$n := fitness(s_i)$ {* *fitness value of the solution* s_i *}

Allocate n *slots for solution* s_i *in RW*

For $i := 1$ *to* $|P_{old}|$

Pick a slot at random in RW. Let s_k *be the solution in the slot*

$P_{new} := P_{new} + \{s_k\}$

$return(P_{new})$

end

5.2.2 Chromosome Representation

Each solution for the problem being solved by the genetic algorithm is coded in a *chromosome representation*. where particular attributes needed to specify a solution are coded by means of genes. Sometimes the information coded in a chromosome is incomplete in order to allow efficient software implementations. and in this case two different solutions may have the same *chromosome representation*. However different *chromosome representations* will correspond to different solutions.

5.2.3 Mutation and Crossover Operators

The operators *mutation* and *crossover* are implementation dependent. Intuitively, they work in a way analogous to biological systems. The *mutation* operator corresponds to a random change in one or more *genes* of the *chromosomic representation* of a solution. The *crossover* operator corresponds to a exchange of genes between different solutions. The definition of these operators is highly dependent on the particular chromosome representation to be used in the genetic algorithm being implemented. The probabilities of mutation p_m and crossover p_c are given by the user and tailored by means of experimentation.

5.2.4 Genetic Algorithms and Random Search Methods

A genetic algorithm can be reduced to a trivial random search by making $p_c = p_m = 1$. Furthermore, genetic algorithms can be understood as an intelligent type of random search, where solutions with high fitness values are assumed to be closer to the optimum than others with lower fitness values and therefore preferred during the formation of the population of the next iteration $P(t+1)$. The convergence to local optimal points can be avoided by the presence of the *mutation* and *crossover* operators that increase the chances to search the entire solution space by ensuring *genetic diversity*.

5.2.5 Initial Solutions

The use of a *genetically diverse* set of initial solutions can facilitate the convergence of a genetic algorithm to the optimum. For the problem of system-level design, high diversity corresponds to the presence of initial solutions representing samples of different regions (subsets) of the design space.

5.2.6 Measuring the Diversity of a Population

This thesis introduces the following coefficients to be used in the evaluation of the *genetic diversity* of a population generated by the genetic algorithm oriented to system-level design.

Definition 5.1 *The coefficient $Q_{diversity}(\mathcal{S})$ is a measure of the overall diversity of a solution set $\mathcal{S}$.*

Definition 5.2 *The coefficient $Q^{Perf}_{diver}(\mathcal{S})$ is a measure of the diversity in performance of a solution set $\mathcal{S}$.*

Definition 5.3 *The coefficient $Q^{Cost}_{diver}(\mathcal{S})$ is a measure of the diversity in cost of a solution set $\mathcal{S}$.*

$$Q_{diversity}(\mathcal{S}) = \frac{n(\mathcal{S})}{|\mathcal{S}| - n(\mathcal{S}) + 1} * \left[W_{cost} * Q^{Cost}_{diver}(\mathcal{S}) + W_{perf} * Q^{Perf}_{diver}(\mathcal{S})\right] \qquad (5.6)$$

$$W_{cost}, W_{perf} \geq 0 \tag{5.7}$$

$$W_{cost} + W_{perf} = 1 \tag{5.8}$$

$$Q^{Cost}_{diver}(\mathcal{S}) = \frac{\sigma(Cost(\mathcal{S}))}{\mu(Cost(\mathcal{S}))} * g\left(Spread_{Cost}(\mathcal{S}), Cost_{max} - Cost_{min}\right) \tag{5.9}$$

$$Spread_{Cost}(\mathcal{S}) = max_{x_i \in \mathcal{S}}(Cost(x_i)) - min_{x_i \in \mathcal{S}}(Cost(x_i)) \tag{5.10}$$

$$Q^{Perf}_{diver}(\mathcal{S}) = \frac{\sigma(Perf(\mathcal{S}))}{\mu(Perf(\mathcal{S}))} * g\left(Spread_{Perf}(\mathcal{S}), Perf_{max} - Perf_{min}\right) \tag{5.11}$$

$$Spread_{Perf}(\mathcal{S}) = max_{x_i \in \mathcal{S}}(Perf(x_i)) - min_{x_i \in \mathcal{S}}(Perf(x_i)) \tag{5.12}$$

$$g(x, y) = \begin{cases} \frac{x}{y} & \text{if } y \neq 0 \\ 1 & \text{otherwise} \end{cases} \tag{5.13}$$

where

$\mathcal{S}$ is a solution set (bag)

W_{cost} (W_{perf}) is a weight proportional to the relative importance of the cost (performance).

$n(\mathcal{S})$ is the number of distinct elements in $\mathcal{S}$.

$|\mathcal{S}|$ is the number of elements in $\mathcal{S}$.

x_i is a solution in $\mathcal{S}$.

$Cost(x_i)$ is the cost of x_i.

$Perf(x_i)$ is the performance of x_i

$Cost_{max}$ ($Cost_{min}$) is the maximum (minimum) possible value for the cost of a solution.

$Perf_{max}$ ($Perf_{min}$) is the maximum (minimum) possible value for the performance of a solution.

$\mu(Cost(\mathcal{S}))$ ($\sigma^2(Cost(\mathcal{S}))$) is the average (variance) of the cost of the solutions in $\mathcal{S}$.

$\mu(Perf(\mathcal{S}))$ ($\sigma^2(Perf(\mathcal{S}))$) is the average (variance) of the performance of the solutions in $\mathcal{S}$.

5.2.7 Variations of the Basic GA Template for System-Level Design

Algorithm 5.1 is a basic template for a genetic algorithm for system-level design. Relevant variations of the proposed template are discussed in this section. Emphasis is given to alternative strategies leading to sound heuristics for the system-level design problem. A comparative study of the effectiveness of these different approaches is presented in Chapter 6.

5.2.7.1 Selection Schemes

Definition 5.4 *Given $\mathcal{P}(i)$ population of solutions for iteration i, and the fitness value $f(x_k)$ of solution $x_k \in \mathcal{P}(i)$, the* policy *to decide if x_k will be part of $\mathcal{P}(i+1)$ is called a* selection scheme.

There are many possible selection schemes in the genetic algorithm literature. Among the most relevant for system-level design are the following:

i. proportionate reproduction [105],

ii. tournament selection [105] and

iii. simulated annealing [121].

5.2.7.1.1 Proportionate Reproduction

The probability that x_k will be selected to be part of $\mathcal{P}(i+1)$ is given by [105]

$$p_{i,i+1}(x_k) = \frac{f(x_k)}{\sum_{x_k \in \mathcal{P}(i)} f(x_j)} \tag{5.14}$$

where $f(x_j)$ is the fitness value of solution x_j. An example of the *proportionate reproduction* scheme is exemplified by the routine *reproduce* (Algorithm 5.2).

5.2.7.1.2 Tournament Selection

This selection scheme [105] can be described by the following algorithm.

Algorithm 5.3 *Tournament*$(s, \mathcal{P}(i), m, \mathcal{P}(i+1))$

$\quad\mathcal{P}(i+1) \longleftarrow \emptyset$
$\quad$*for* $i = 1$ *to* m
$\quad\quad$*Choose at random s solutions* $x_1, x_2, \ldots, x_s \in \mathcal{P}(i)$
$\quad\quad$*Take solution* x_r *with best fitness value among* $x_1, x_2, \ldots, x_s$
$\quad\quad\mathcal{P}(i+1) \longleftarrow \mathcal{P}(i) + x_r$
$\quad$*end*

where s is the size of the tournament, $\mathcal{P}(k)$ is the population of solutions in iteration k. m is the desired size for population $\mathcal{P}(i+1)$.

The tournament selection method can have many variations as, for example, x_r, chosen in tournament $i = j$ $(1 \leq j < m)$ may be considered or not in the next tournaments $(j+1, \ldots, \leq m)$.

5.2.7.1.3 Selection by Simulated Annealing

In the selection by simulated annealing scheme [121], population $\mathcal{P}(i+1)$ is derived from $\mathcal{P}(i)$ by simulated annealing [89]. The basic idea is to progressively disallow offsprings with low fitness values to be part of the population $\mathcal{P}(i+1)$ by decreasing the temperature θ at each iteration i of the genetic algorithm.
Properties of selection by simulated annealing:

i. A solution x_r resulting from the mutation of a parent solution x_s is accepted in $P(i+1)$ with a probability

$$p_{i,i+1}(x_r) = min\left(e^{\frac{(f(x_r)-f(x_s))}{\theta_i}}, 1\right) \tag{5.15}$$

where

θ_i is the temperature on iteration i, and

$\theta_{i+1} = \kappa * \theta_i$, where $\kappa < 1$ gives the annealing schedule for the temperature.

ii. Two solutions x_{r_1} and x_{r_2} resulting from the crossover of parent solutions x_{s_1} and x_{s_2} are accepted in $P(i+1)$ with a probability

$$p_{i,i+1}(x_{r_1}) = p_{i,i+1}(x_{r_2}) = min\left(e^{\frac{(f(x_{r_1})+f(x_{r_1})-f(x_{s_1})-f(x_{s_1}))}{\theta_i}}, 1\right) \tag{5.16}$$

iii. A parent solution is not accepted in $\mathcal{P}(i+1)$ if one of its children was accepted.

iv. All childless solutions are accepted in $\mathcal{P}(i+1)$.

5.2.7.2 Scaling of the Fitness Function

The redefinition of the *fitness value* function (*scaling*) provides an additional mechanism to control the rate of convergence of a genetic algorithm. It has been shown that early convergence to a *local optimum* might be avoided by careful scaling [89]. Among the most commonly used techniques for scaling are the following:

i. linear scaling

$$f^{'}(x_j) = a * f(x_j) + b \tag{5.17}$$

where $f^{'}(x_j)$ ($f(x_j)$) is the *fitness value* of solution x_j after (before) *scaling*;

ii. sigma truncation

$$f^{'}(x_j) = f(x_j) + \mu(f(\mathcal{P}(t))) - c * \sigma(f(\mathcal{P}(t))) \tag{5.18}$$

where

c is a small constant.

$\mu(f(\mathcal{P}(t)))$ is the average value of the fitness value for solutions in $\mathcal{P}(t)$.

$\sigma^2(f(\mathcal{P}(t)))$ is the variance of the fitness value for population $\mathcal{P}(t)$; and

iii. power law scaling

$$f'(x_j) = f^k(x_j) \tag{5.19}$$

where k is close to 1 [89].

5.2.7.3 Avoidance of Replicated Solutions

Most of the selection schemes studied in Section 5.2.7.1 favor the existence of multiple copies of the *best solution* $x_{best} \in \mathcal{P}(t)$ as well as copies of the other solutions with high fitness values in the next iteration population $\mathcal{P}(t+1)$. This decreases the *genetic diversity*, what may lead to convergence to a *local optimum*. This can be avoided by not allowing the presence of repeated solutions in $\mathcal{P}(t)$ and $\mathcal{P}(t+1)$, which, therefore, become *proper sets*:

$$|\mathcal{P}(i) + x_i| = |\{\mathcal{P}(i) + x_i\} + x_i| \tag{5.20}$$

5.2.7.4 Variable Size Population

Another possible counter-measure against progressive loss of *genetic diversity* would be to allow the size of the population to vary in time. This thesis introduces the following algorithm, which outlines an approach to control the population size by observing the *diversity coefficient* $Q_{diversity}(\mathcal{P}(i))$.

Algorithm 5.4 *Procedure GAvarsize*(m, Q_-, Q_+, η) {**Genetic Algorithm with varying population size**}
  {**m is the size of the initial population* *}
  {**Q_- and Q_+ are threshold values* *}
  {*$\eta > 1.0$*}
  begin
    $t := 0$
    {**initial population of feasible solutions**}
    initialize $\mathcal{P}(t)$ *such that* $|\mathcal{P}(t)| == m$
    evaluate the fitness value *of each solution* s_i *in* $\mathcal{P}(t)$
    inc = 0;
    repeat begin
      evaluate $Q_{diversity}(\mathcal{P}(t))$
      if $Q_{diversity}(\mathcal{P}(t)) < Q^-$ /* *low diversity* */

```
            m = ⌈m * η⌉
            inc = inc + 1
      else if Q_diversity(P(t)) > Q+ and inc > 0 /* high diversity */
            m = ⌊m * η^-1⌋
            inc = inc - 1
      t = t + 1
      P(t) = reproduce(P(t - 1))
      bestsolution = solution with highest fitness value in P(t)
      for i := 1 to |P(t)|/2 do
            Let x be a random number with an uniform probability
            density distribution in the interval [0, 1].
            if x ≤ p_c then {*p_c is the probability of crossover *}
                  Choose at random two solutions s_k and s_j in P(t)
                  (s*_k, s*_j) = crossover(s_k, s_j)
                  P(t) := P(t) - {s_k, s_j} + {s*_k, s*_j}

      for i := 1 to |P(t)| do
            Let y be a random number with an uniform probability
            density distribution in the interval [0, 1].
            if y ≤ p_m {*p_m is the probability of mutation *}
                  Pick solution s_i in P(t)
                  s*_i := mutation(s_i)
                  P(t) := P(t) - {s_i} + {s*_i}

      If m > |P(t)| then
            Add m - |P(t)| random solutions to P(t).
      else
            Remove |P(t)| - m random solutions from P(t).
      evaluate fitness of each solution s_i in P(t)
      worstsolution := solution of worst fitness value in P(t)
      P(t) := P(t) - {worstsolution} + {bestsolution}
   until P(t) converges ({*i.e. P(t) = P(t - 1) *}) or t > t_max
end
```

5.3 Summary of the Chapter

An overview of the subject of *genetic algorithms* and how they can be used in system-level design of application-specific heterogeneous multiprocessors was provided. Emphasis was given to particular approaches suitable to be used along the tool set set described in the next chapter.

Chapter 6

The MEGA System

MEGA is a tool set for design of application-specific heterogeneous multiprocessors with non-negligible communication costs. This chapter and Chapters 7 and 8 describe the set of genetic algorithms implemented in the MEGA system. which is based on the previously discussed concepts and algorithms.

6.1 Covered Computational Paradigms

MEGA supports three computational models:

- *Deterministic* - where all cost and performance parameters are predefined constants:
- *Imprecise* - which allows the use of the *imprecise computation* model. introduced by Liu et al. [79] for applications in *real-time* programming: and
- *Probabilistic* - where at least one of the cost and performance parameters is modeled as a random variable. This model is used to allow design in the presence of uncertainty.

MEGA supports two execution modes:

- *Non-periodic* as discussed in Chapter 1.
- *Periodic (macro-pipelined)* with *finite* horizon as discussed in Chapter 3.

Each pair (x, y) where x is one of the allowed computational models and y is one of the two possible execution modes defines a computational paradigm supported by MEGA. The remainer of this chapter is oriented toward the deterministic paradigms:

i. $Paradigm_i$ = (deterministic, non-periodic)

ii. $Paradigm_{ii}$ = (deterministic, periodic)

Chapters 7 and 8 discuss respectively the *imprecise computation* and *probabilistic* paradigms.

6.2 Supported Interconnection Networks

The proper design of interprocessor communication networks is a key part of the performance optimization problem when designing an application-specific multiprocessor with non-negligible communication costs. MEGA allows two basic interconnection architectures:

- *shared buses* architecture, where the processors communicate among themselves through a set of shared bidirectional buses of different speeds and costs (heterogeneous communication network); and all processor can acess all buses.
- *dedicated* architecture, where each bus connects only a subset of the available processors. A *point-to-point* intercommunication network, for example, can be specified as one where each bus connects only two processors. An array processor can be specified as one where the dedicated *point-to-point* connections follow a regular arrangement [58] such as mesh or hypercube.

6.3 Task-Flow Graph Representation

In order to allow the development of structured genetic algorithms for the different supported paradigms, MEGA has a particular way to represent general task-flow graphs internally.

6.3.1 Data-Transfer Types

Inside MEGA, each data transfer is a connection between two tasks *only* (1 : 1 communication). Each data transfer corresponds, therefore, to a

- *data-dependency* [58] , where real data is transferred ($Volume(DT) > 0$) or
- *control-dependency* [58], where no real data is passed ($Volume(DT) = 0$).

6.3.2 Dummy Tasks and Twin Tasks

MEGA allows the use of *dummy tasks*, which are tasks with zero computation time which are allocable to any of the processors of the multiprocessor being implemented.

Property 6.1 *Dummy tasks can be overlapped by any other tasks, dummy or not.*

MEGA allows the use of *gang scheduling* [58] by means of the use of *twin* tasks. A set of *twin tasks* corresponds to a subset of the tasks in the task-flow graph whose elements must be allocated to the same processor.

6.3.3 Broadcast, Multicast and Dummy-Twin Data Transfers

In MEGA, *broadcast* and *multicast* [58] (1 : N communication, $N > 1$) are internally represented by the use of *dummy tasks*. For example, a *multicast* $MT = (S_a \longrightarrow S_b, S_c, \ldots, S_z)$, i.e., from task S_a to tasks $S_b, S_c, \ldots, S_z$ with a *data volume* $Vol(MT)$, is modeled by the use of the following the following *one-to-one* (1 : 1) *dummy-twin* data transfers:

$S_a \longrightarrow S_b, S_a \longrightarrow S_c, \ldots, S_a \longrightarrow S_d$ are *dummy-twin data transfers*
where $Vol(S_x \longrightarrow S_y)$ is volume of the data transfer between S_x and S_y:

$$Vol(S_a \longrightarrow S_b) = \ldots = Vol(S_a \longrightarrow S_z) = Vol(MT) \tag{6.1}$$

The following property of *dummy-twin data transfers* is used by MEGA in order to avoid unnecessary traffic over the interconnection network, derived from the fact that the m *dummy-twin data transfers* carry the same data, and therefore it is just necessary to send one of them over the bus.

Property 6.2 *If* $m > 1$ dummy-twin data transfers *are allocated to the same* bus (data-link*), there is only need to schedule only one of them over the bus.*

6.4 Input Data

MEGA takes five input files: the *processor types* library, the *bus types* library, the *task flow graph*, *performance/cost* constraints, *topological* constraints and genetic algorithm *parameters*.

6.4.1 Processor Types Library

The *processor types library* describes the set of available off-the-shelf and custom processor types (e.g. ASIC processors, Motorola 68000 or Intel Pentium). Each processor type has the following associated information:

- *name* of the processor: a unique identifier specifying manufacturer, model and version in case of an *off-the-shelf processor* or the proposed name of an *ASIC processor* already or still to be implemented;
- *cost* of the processor in terms of area, dollars, power or any other meaningful way to measure cost; and
- *local communication delay* which is the average delay for a local data transfer between consecutive tasks executing in the same processor. It is usually equal to 0 for an ASIC processor, and equal to the *context switch* time for an off-the-shelf processor.

6.4.2 Bus Types Library

The *bus types library* describes the set of available bus (data-link) types between the processors of the multiprocessor to be implemented. Each bus type has the following associated information:

- *name* of the bus: a unique identifier specifying, for example, the protocol used in the bus (e.g. Futurebus);

- *cost* of the bus in terms of area, dollars, power or any other possible measure;

- *latency* (τ_{bus}): transmission delay of the bus in units of time: and

- *Speed* (s_{bus}), where $D_{RC_{bus}} = s_{bus}^{-1}$ gives the transfer rate (delay per unit of data) for a data transfer allocated to a bus of this type. It is assumed that overall communication delay $Delay(DT)$ is a function only of the volume of data being transferred, speed and bus latency.

$$Delay_{remote}(DT, bus) = \tau_{bus} + Volume(DT) * D_{RC_{bus}} \tag{6.2}$$

6.4.3 Task Flow Graph

The *task-flow graph* represents the application to be implemented. There are two main entities in this input: tasks and data transfers.

6.4.3.1 Tasks

Each task S has the following associated information:

- *name* of the task: a unique identifier for each task in the task-flow graph: and

- *performance trade-offs*: a list of pairs $(p, \omega_p(S))$, where p is a processor type and $\omega_p(S)$ is the computation time of the task S in a processor of type p. If a processor type p' is not in the list, it means that the task cannot be executed by a processor of type p' ($\omega_{p'}(S) = \infty$).

6.4.3.2 Data Transfers

Each data transfer DT has the following associated information:

- *label* of the data transfer: a unique identifier for each data transfer in the task-flow graph:

- *source and sink(s)*: name of the source task S_i and the m sink tasks $S_{j_1}, \ldots, S_{j_m}$ associated with the data transfer DT.

- volume of the data transfer in units of data:

- coefficient $r^i_{out_k}$, where the data transfer DT corresponds to the kth-output of source task S_i using the same notation as Section 1.2.1.15; and

- list of coefficients $r^{j_l}_{in_k}$, $1 \leq l \leq m$, where the data transfer DT corresponds to the kth-input of sink task S_{j_l} using the same notation as Section 1.2.1.15.

In a multicast or broadcast, there are $m > 1$ sinks. However, there is always only one source task for a given *data transfer*.

6.4.3.3 Twin Tasks

Optionally, the user can specify disjoint sets of *twin* tasks, subject to the following constraint:
For every pair of twin tasks S_i and S_j:

$$\forall \ processor \ \ types \ p \ , \ \ \omega_p(S_i) < \infty \Longleftrightarrow \omega_p(S_j) < \infty \tag{6.3}$$

6.4.4 Performance/Cost Constraints

Performance/cost constraints input to the system include the following information:

- *Paradigm* to be used, i.e., pair (x, y) where x is one of the allowed computational models and y is one of the two possible execution modes as discussed in Section 6.1.

- *Maximum processor cost* ($Cost^{proc}_{max}$). The sum of the costs of all processors in the multiprocessor should be less than this number.

- *Maximum interconnection cost* ($Cost^{comm}_{max}$). The total cost of the interconnection network (sum of the costs of all buses/data-links) is bounded by this constraint.

- *Maximum overall cost:* ($Cost_{max}$) maximum cost for the multiprocessor to be implemented. This cost is less than or equal to the sum of the *maximum processor cost* and *maximum interconnection cost.*

- *The maximum number of processors of type p* ($NPROC^{max}(p)$) for each processor type.

- *The maximum number of processors* ($NPROC^{max}$), where

$$\sum_p NPROC^{max}(p) \geq NPROC^{max} \tag{6.4}$$

- *The maximum number of buses of type l* ($NBUS^{max}(l)$) for each bus type.

- *The maximum of number buses* ($NBUS^{max}$), where

$$\sum_l NBUS^{max}(l) \geq NBUS^{max} \tag{6.5}$$

- *The maximum* (T_{SYS}^{max})*and minimum* (T_{SYS}^{min}) acceptable values for the system latency T_{SYS}.

- *The maximum* (T_I^{max}) *and minimum* (T_I^{min}) acceptable values for the initiation interval T_I - available for the *macro-pipelined* mode of execution only.

- *Objective function* (f_{obj}) weights W_{cost}, $W_{T_{SYS}}$ and W_{T_I}, where

$$f_{obj} = W_{cost} * Cost + W_{T_{SYS}} * T_{SYS} + W_{T_I} * T_I \tag{6.6}$$

  $W_{T_I} = 0$ for the *non-periodic* case and $Cost$ is the overall cost of the multiprocessor.

- n_{over} *overlapping factor*, $1 \leq n_{over} \leq \infty$, where $n_{over} = \infty$ would correspond to an *infinite horizon* scheduling heuristics. This parameter is passed only for *macro-pipelined* paradigms.

6.4.5 Topological Constraints

MEGA allows the specification of predefined interconnection networks such as mesh, cube and ring. If there are no topological constraints MEGA allows each bus to be shared by any of the processors of the multiprocessor to be implemented.

In the topologically constrained mode MEGA accepts the following information:

i. A list of processors LP, where each processor $p_x \in LP$ has a list of allowed *processor types* $PT(p_x)$.

ii. A list of buses (data links) LB, where each bus $b_x \in LB$ is allowed to connect only a subset $Conn(b_x)$ of the processors in LP. Each bus in LB has an associated list $BT(b_x)$ which gives the possible bus types for b_x.

6.4.6 Genetic Algorithm Parameters

The *genetic algorithm parameters* control the kernel of the genetic algorithm used by MEGA. They include the following information:

i. number of chromosomes in the initial population,

ii. profile parameters for the initial population as discussed in Section 6.15,

iii. probability of crossover p_c and probability of mutation p_m, and

iv. maximum number of iterations of the genetic algorithm.

6.5 Output Data

MEGA outputs detailed allocation and scheduling *timing* information about the tasks and data transfers in the *task flow graph*, i.e., the processor (bus) to which each task (data transfer) is specified together with its start $T_{SS}(\bullet)$ ($T_{CS}(\bullet)$) and end times $T_{SE}(\bullet)$ ($T_{CE}(\bullet)$). For the *macro-pipeline* case, the *timing* information is related to the task and data transfer *instances* in the *base iteration*.

With proper post-processing of the output data, a set of Gannt charts [58] can be built describing the scheduling of tasks (data transfers) in each processor (bus), as seen in Section 6.20.

6.6 Main Data Structures

The data-structures in MEGA are oriented to a *non-pure* task-flow-graph model of computation. However, MEGA does not need to be a direct translation of the MILP models previously studied, because there is no need to linearize a genetic algorithm optimizer. This allows the use of straightforward data structures highly related to the system-level design problem as seen in following sections.

6.6.1 Processor Types Table

The *processor types* table stores the information regarding processor types. Each p-element of this table has the following attributes:

i. *Name*,

ii. *Cost*,

iii. *Local delay*, and

iv. $NPROC^{max}(p)$,

using the same naming convention as in Sections 6.4.1 and 6.4.4.

6.6.2 Processor Table

The *processor* table stores the *topological* constraints of each processor, i.e. the list $PT(p_x)$ of possible types for processor p_x. The size of this table is given by the parameter $NPROC^{max}$, where the information about the possible types is stored in the *processor types table*.

6.6.3 Bus Types Table

In a similar way, each l-element of the *bus types* table has the following attributes:

i. *Name*,

ii. *Cost*,

iii. *Latency*,

iv. *Speed*, and

v. $NBUS^{max}(l)$,

using the same naming convention as in Sections 6.4.2 and 6.4.4.

6.6.4 Bus Table

The list of possible types $LT(b)$ for a bus b and the list $Conn(b)$ of processors allowed to communicate through b is stored in the b-element of the *bus* table, which stores *topological* constraints regarding the *interconnection* network of the multiprocessor being implemented.

6.6.5 Task Table

Each element of the *task table* corresponds to either one of the tasks in the original *task-flow graph* given as an input to MEGA or a *dummy* task. An s-element of this table, corresponding to task S, has the following attributes:

i. *Name.*

ii. *List of pairs* $(p, \omega_p(S))$, where p is a *processor-type* where the task S can be executed.

iii. *List of outgoing data transfers.*

iv. *List of incoming data transfers,*

v. *List of tasks parallel to task* S not including *dummy* tasks.

vi. *List of tasks that are twin to* S, and

vii. A flag indicating if S is dummy or not.

6.6.6 Data-Transfer Table

Differently from general task-flow graphs, the MEGA internal representation restricts data transfers to 1 : 1 communications, as it can be seen from the list of attributes of a d-element of this table:

i. *Label.*

ii. *Source* task.

iii. *Sink* task (S_j).

iv. *Volume*,

v. *List of parallel data transfers*,

vi. *List of* dummy-twin *data transfers*,

vii. *Difference of subscripts* (macro-pipelined case),

viii. $r^i_{out_k}$, where the data transfer d corresponds to the kth-output of the source task (S_i), and

xix. $r^j_{in_{k'}}$, where the data transfer d corresponds to the kth-input of the sink task (S_j).

Each table element corresponds to one of the data transfers of the transformed TFG, i.e., ones obtained after the reduction of the original multicast flow to sets of $1:1$ data transfers.

6.6.7 Chromosome Representation in MEGA

The internal structure of the chromosomes in MEGA was designed having in mind the following goals:

- to allow simple and efficient implementations,
- to handle non-periodic and macro-pipelined (finite horizons) mode of execution, and
- to be flexible enough to allow easy extension to the *imprecise computation* and *probabilistic* computations models, as described in Chapters 7 and 8.

As opposed to standard genetic algorithms that typically only use binary (0, 1) genes [89], a MEGA genetic representation is *hybrid*. There are two types of genes:

i. *Discrete* genes whose domains are *isomorphic* to finite subsets of $\mathcal{N}$, the set of positive integer numbers. They are used to represent the allocation and relative scheduling information.

ii. *Continuous* genes whose domains are closed *intervals* of $\mathcal{R}$, the set of real numbers. They correspond to the *timing* variables of the corresponding MILP models.

6.6.7.1 Allocation

The θ genes and ϑ genes correspond to the Boolean variables representing task allocation in the MILP models for system-level design, but they may have domains with cardinality greater than 2, i.e., they are not necessarily Boolean variables themselves.

Definition 6.1 *The θ-genes set $\{\theta_1, \ldots, \theta_i, \ldots\}$ represents the* task *to* processor *mapping. $\theta_i = x$ denotes that task S_i is assigned to processor p_x.*

Definition 6.2 *The ϑ-genes set $\{\vartheta_1, \ldots, \vartheta_i, \ldots\}$ represents the* processor *to* processor type *mapping, where $\vartheta_x = t$ denotes that processor p_x is a processor of type t.*

The following property is derived from the definition of *twin* tasks.

Property 6.3 *For a set of* twin *tasks $\{S_a, S_b, \ldots, S_z\}$, the following constraint must be enforced in order to have a* consistent *Boolean assignment.*

$$\theta_a = \theta_b = \ldots = \theta_z \tag{6.7}$$

The following property is a consequence of Definitions 6.1 and 6.2 and Section 6.4.3.1.

Property 6.4 *For a set of tasks $\{S_a, \ldots, S_z\}$, the following constraint must be enforced in order to have a* consistent *Boolean assignment.*

$$\theta_a = \ldots = \theta_z = p \text{ and } \vartheta_p = t \implies \omega_t(S_a) + \ldots + \omega_t(S_z) < \infty \tag{6.8}$$

In a similar way, π-genes and ϖ-genes correspond to the Boolean variables dealing with the allocation of non-local data transfers to buses, but they are not Boolean variables themselves.

Definition 6.3 *The π-genes set $\{\pi_1, \ldots, \pi_i, \ldots\}$ represents the* data transfer *to* bus (data-link) *mapping. $\pi_i = x$ denotes that data transfer DT_i is assigned to bus (data-link) b_x. On the other hand, $\pi_i =$* dummy *denotes that the data transfer is* local, *and there is no need to use a bus.*

Definition 6.4 *The ϖ-gene set $\{\varpi_1, \ldots, \varpi_i, \ldots\}$ represents the* bus (data-link) *to* bus type *mapping, where $\varpi_x = t$ denotes that bus b_x is a bus of type t.*

6.6.7.2 Relative Scheduling

The α-gene (Gen_α) and Φ-gene (Gen_Φ) sets are directly related to the Boolean variable types α and Φ of the proposed MILP models. However their interpretation is dependent on the *execution mode*. They have the following properties:

i. The set $Alpha(i, j) = \{\alpha_{ij}[n-1\ 0], \ldots, \alpha_{ij}[00], \ldots \alpha_{ij}[0\ n-1]\}$ is a subset of the α-gene set. Each gene in $Alpha(i, j)$ is associated with an α-variable as defined in the proposed MILP model for *macro-pipelined* design with *finite horizon* (Chapter 3). $Alpha(i, j)$ indicates the relative order of execution among parallel instances of S_i and S_j, where $n = n_{over}$ = *overlapping factor* as discussed in Section 3.3.3.3 ($n = 1$ corresponds the non-periodic case).

$$Gen_\alpha = \bigcup_{\forall S_i // S_j} Alpha(i, j) \tag{6.9}$$

$$Alpha(i, j) \cap Alpha(r, s) = \{\} \tag{6.10}$$

for $i \neq r$ and $j \neq s$, or for $j \neq r$ and $i \neq s$; i.e., the $Alpha(\bullet)$ sets are disjoint.

ii. In a similar way, the set $Phi(i, j) = \{\Phi_{ij}[n-1\ 0], \ldots, \Phi_{ij}[00], \ldots \Phi_{ij}[0\ n-1]\}$ is defined for each pair of concurrent data transfers, i.e. a pair of data transfers having instances that are *parallel*, where $n = n_{over} + Diff_{max} - 1$, as discussed in Section 3.3.3.3.

$$Gen_\Phi = \bigcup_{\forall S_i // S_j} Phi(i, j) \tag{6.11}$$

$$Phi(i, j) \cap Phi(r, s) = \{\} \tag{6.12}$$

for $i \neq r$ and $j \neq s$, or for $j \neq r$ and $i \neq s$, i.e. the $Phi(\bullet)$ sets are disjoint.

6.6.7.3 Timing

Each timing variable of the MILP models discussed has an associated *continuous* domain gene, as discussed in Chapter 4. These genes, called *timing* genes, inherit all the properties of the timing variables. Therefore the same name used for a timing variable $T_x(\bullet)$ in the MILP model is used to identity its corresponding gene. *Dummy-twin* data transfers introduce the following additional additional property:

Property 6.5 *For a set of* dummy-twin *data transfers* $\{DT_a, DT_b, \ldots, DT_z\}$ *allocated to the same bus, the following constraint must be enforced in order to have a* consistent *Boolean assignment.*

$$\begin{aligned} T_{CS}(DT_a) &= \ldots = T_{CS}(DT_z) \\ T_{CE}(DT_a) &= \ldots = T_{CE}(DT_z) \end{aligned} \tag{6.13}$$

6.7 Fitness Evaluation

Once all discrete variables receive a valid (feasible) assignment, the set of constraints among genes is reduced to a system of difference constraints involving the values of *timing* genes. The fitness values can be made a function of cost, latency T_{SYS} and initiation interval T_I for the macro-pipelined case. T_I and T_{SYS} can be easily be evaluated by applying the techniques discussed in Chapter 4 for non-periodic and macro-pipelined cases.

In MEGA, the fitness is a function of the following form:

$$f(x) = F_{max} - A * Cost(x) - B * T_{SYS}(X) - C * T_I(x) \tag{6.14}$$

where $A, B, C \geq 0$ and F_{max} is large enough to ensure that $\forall x \;\; f(x) \geq 0$.

6.7.1 Fitness Scaling by Moving Window

The *moving window* algorithm is used to scale the fitness of population $\mathcal{P}(i)$ of iteration i.

Algorithm 6.1 *Moving window fitness scaling algorithm.*

m = number of solutions in $\mathcal{P}(i)$.

low = lowest fitness value in $\mathcal{P}(i)$.

for all $x \in \mathcal{P}(i)$ do

$$f'(x) = f(x) - \frac{(m+1) * low}{m}$$

where $f'(x)$ ($f(x)$) is the fitness after (before) scaling.

6.7.2 Normalized Fitness

All fitness values are normalized after scaling.

$$f''(x) = \frac{f'(x)}{\sum_x f'(x)} \tag{6.15}$$

where $f''(x)$ is the value of the fitness after normalization.

6.8 Selection Schemes

Section 6.20 compares the performance of different *selection schemes* when coded in MEGA. The following algorithm gives details of the implementation of the *proportionate reproduction* selection scheme used in MEGA.

Algorithm 6.2 *Proportionate reproduction*($\mathcal{P}(i), \mathcal{P}(i+1)$)

$m \longleftarrow |\mathcal{P}(i)|$

$\mathcal{P}(i+1) \longleftarrow \{\}$

for $i \longleftarrow 1$ *to* m *do*

  $rw[i] = \sum_{j=1}^{i} f''(x_j)$

for $i \longleftarrow 1$ *to* m *do*

  Call **Continuous Roulette Wheel**(rw, k, m) /*See [89] */

  $\mathcal{P}(i+1) \longleftarrow \mathcal{P}(i+1) + x_k \mid x_k \in \mathcal{P}(i)$

end

Algorithm 6.3 *Continuous Roulette Wheel*(rw, k, m)

/*Adapted from [89] */

/* rw is an real vector of m elements */

$x \longleftarrow$ *random real number between 0 and 1.*

if ($x \leq rw[0]$) *then* $k \longleftarrow 0$

else

  $leave \longleftarrow false$

  for $i \longleftarrow 1$ *to* m *and* **not leave**

    if ($x \leq rw[i]$) *then*

$k \longleftarrow i$

$leave \longleftarrow true$ / Leave loop */*

if (not leave) then / All solutions are equiprobable, what may happen if all solution have the same fitness */*

$k \longleftarrow$ random integer between 1 and m

end

6.9 Transitive Closure and Parallelism Detection

Variations of the transitive closure algorithm [19] are used in MEGA to detect pairs of parallel tasks (data transfers) as discussed in Section 1.2.1.8. Pruning these unnecessary *overlap avoidance* constraints among tasks (data transfers) that cannot overlap due to data dependency and/or control dependency minimizes the sizes of sets Gen_α and Gen_Φ, which may improve the rate of convergence of MEGA.

6.9.1 Macro-Pipelined (Finite Horizon) Case

The macro-pipelined case can be seen as an extension of the non-periodic case. Let G_o be a task flow graph with data-dependencies across different iterations.
Let $1 \leq n_{over} < \infty$ be the *overlapping factor*.

Definition 6.5 *The* unfolded graph G_α *(Figure 6.1) is defined as the task-flow graph obtained by unrolling G_o over $2*n-1$ iterations, where $n = n_{over} + Diff_{max} - 1$ as discussed in Section 3.3.3.3.*

i. Task instance $S_a^\alpha[0] \in G_\alpha$, $a = (2*n+1)*i+k+n-1$, corresponds to task instance $S_i^o[k]$ of the original task-flow graph G_o, where $-n+1 \leq k < n-1$. $S_a^\alpha[0]$ is also called a *replicated instance* (replication) of S_i^o.

ii. In a similar way task instance $S_a^\alpha[u] \in G_\alpha$, $a = (2*n+1)*i+k+n-1$, corresponds to task instance $S_i^o[k+(2*n-1)*u]$ of the original task-flow graph G_o.

iii. Data-transfer instance $DT_a^\alpha[0] \in G_\alpha$, $a = (2*n+1)*i+k+n-1$, corresponds to data-transfer instance $DT_i^o[k]$ of the original task-flow graph G_o, where $-n+1 \leq k < n-1$. $DT_a^\alpha[0]$ is also called a *replicated instance* (replication) of DT_i^o

iv. In a similar way data-transfer instance $DT_a^\alpha[u] \in G_\alpha$, $a = (2*n+1)*i+k+n-1$, corresponds to data-transfer instance $DT_i^o[k+(2*n-1)*u]$ of the original task-flow graph G_o, where $-n+1 \leq k < n-1$.

Definition 6.6 *The* synchronous unfolded *graph G_β (Figure 6.2) is defined as the task-flow graph derived from G_α by adding the following control dependencies:*

i. *For every task $S_i^o \in G_o$, add a* control-dependency *$S_a^\alpha \rightsquigarrow S_{a+1}^\alpha$, where $S_a^\alpha \in G_\alpha$, $S_{a+1}^\alpha \in G_\alpha$, $a = (2*n+1)*i+k+n-1$ and $-n+1 \leq k < n-1$.*

Definition 6.7 *The* myopic synchronous unfolded *graph G_γ (Figure 6.3) is defined as the task-flow graph derived from G_β by deleting data-dependencies across iterations. i.e., whose source and sink tasks in G_β are in different iterations.*

Lemma 6.1 *G_γ is a directed acyclic graph (DAG)*

Proof: It comes from the fact that G_γ is an unrolled (unfolded) version of G_o without considering data-dependencies with task instances outside the *data-transfer overlap window*. □.

6.9.1.1 Bounded Unrolled Graph

Definition 6.8 *Given G_β derived from G_α by insertion of* control dependencies *as discussed in Definition 6.6. the* bounded unrolled *graph G_δ (Figure 6.4) is one derived from G_β by doing the following transformation that substitutes each inter-iteration data transfer in G_β for an equivalent intra-iteration data transfer by adding* dummy tasks *to G_β. The final graph G_δ is a TFG without data transfers between different iterations that can be processed by the same algorithms used in the* non-periodic case. *Note that* dummy *tasks are necessary because all data transfers are required to have a single sink and a single source task in the mathematical model used in MEGA.*

For each data-transfer instance $DT_i[0] \in G_\beta$, either related to a data-dependency or control-dependency, connecting task instances in different iterations:

i. if $DT_i[0] = S_a[0] \longrightarrow S_b[k]$, where $k \neq 0$, create *dummy* task $S_b^{dummy_k}$, twin of S_b. Substitute $DT_i[0]$ for $DT_i'[j] = S_a[j] \longrightarrow S_b^{dummy_k}[j]$ for all iterations $-\infty \leq j \leq \infty$, i.e. this transformation makes $DT_i[0]$ to terminate in the base iteration (iteration 0) by insertion of the *dummy* task $S_b^{dummy_k}$.

ii. if $DT_i[0] = S_a[k] \longrightarrow S_b[0]$, where $k \neq 0$. Create *dummy* task $S_a^{dummy_k}$, twin of S_a. Substitute $DT_i[j]$ for $DT_i'[j] = S_a^{dummy_k}[j] \longrightarrow S_b[j]$ for all iterations $-\infty \leq j \leq \infty$, i.e. this transformation makes $DT_i[0]$ start in the base iteration (iteration 0) by insertion of the *dummy* task $S_b^{dummy_k}$.

From the above transformations, applied to G_β, we derive the following property:

Property 6.6 *G_δ is a directed acyclic graph without data-dependencies between different iterations.*

Algorithm 6.4 *Parallelism detection for the macro-pipelined (finite horizon) case*

```
Given G = TFG
Derive G_α from G
Derive G_β from G_α
Derive G_γ and G_δ from G_β
T = transitive closure(G_γ, m) /*See [19] */
/* T is m × m matrix with the transitive closure of */
/* the tasks in G_γ */
/* m is the number of tasks in G_γ */
/* T[a, b] = 0 denotes that S_a is concurrent to S_b */
for a ⟵ 1 to m
        for b ⟵ a + 1 to m
                if (T[a, b] = 0) and P_Si ∩ P_Sj ≠ {} then
                        /* P_Si ∩ P_Sj ≠ {} means that S_i and S_j can be */
                        /* allocated to the same processor */
                        S_i[k] is parallel to S_j[k'] in G_o
                        where
                                a = (2 * n + 1) * i + k + n_over − 1
                                b = (2 * n + 1) * j + k' + n_over − 1
/* Detect parallelism among data transfers */
/* m' is the number of data transfers in G_δ */
for a ⟵ 1 to m'
```

Let be $DT_a = S_r \longrightarrow S_s$
for $b \longleftarrow a + 1$ *to* m'
    Let be $DT_b = S_u \longrightarrow S_v$
    if both DT_a *and* DT_b *are transfers to/from a dummy task*
    or $(T[s, u] = 0)$ **and** $(T[v, r] = 0)$ *then*
        /*Detect possibility of overlapping */
        $DT_i[k]$ *may be parallel to* $DT_j[k']$ *in* G_o
        where

$$a = (2 * n + 1) * i + k + n - 1$$
$$b = (2 * n + 1) * j + k' + n - 1$$
$$n = n_{over} + Diff_{max} - 1$$

end

6.9.1.2 Twin Tasks and Iteration-Twin Data Transfers

Definition 6.9

The set of iteration-twin *data transfers of* $DT_i \in G_o$ *is given by* $ITwins(DT_i, G_\beta) = \{DT_a \in G_\beta \mid a = (2 * n + 1) * i + k + n - 1, \ -n + 1 \leq k < n - 1\}$.

The tasks (data transfers) of the *synchronous unfolded* graph G_β derived from G_α have the following properties:

i. *For every task* $S_i \in G_o$, *there are* $2 * n - 1$ twin tasks $S_a \in G_\alpha$, $a = (2 * n + 1) * i + k + n - 1$ *and* $-n + 1 \leq k < n - 1$.

ii . *For every data transfer* $DT_i \in G_o$, *there are* $2 * n - 1$ iteration-twin data transfers $DT_a \in G_\alpha$, $a = (2 * n + 1) * i + k + n - 1$, $-n + 1 \leq k < n - 1$ *and* $n = n_{over} + Diff_{max} - 1$ *as discussed in Section 3.3.3.3.*

Property 6.7

Two iteration-twin data transfers $DT_a \in ITwins(DT_i, G_\beta)$ *and* $DT_b \in ITwins(DT_i, G_\beta)$ *have the following common properties:*

i. Same data volume

$$Volume(DT_a) = Volume(DT_b) \tag{6.16}$$

ii. Same allocation

$$\pi_a = \pi_b \tag{6.17}$$

Lemma 6.2 *Two* iteration-twin data transfers $DT_a \in ITwins(DT_i, G_\beta)$ *and* $DT_b \in ITwins(DT_i, G_\beta)$ *cannot be merged together as if they were* dummy-twin *data transfers.*

Proof: They might refer to different data, i.e., they correspond to different instances of $DT_i \in G_o = TFG$. □

6.10 Fast Topologic Ordering

The following algorithm can be used to find the topological order of acyclic *timing constraint* graphs, given the original task-flow graph. It takes advantages of the following properties of the *timing constraint graphs*:

Lemma 6.3 *Given vertex* $v_{SE_i} \in G$ *associated with timing variable* $T_{SE}(S_i)$ *and* $v_{SS_i} \in G$ *associated with* $T_{SS}(S_i)$, *where* G *is an acyclic* timing constraint *graph,* v_{SE_i} *can be placed immediately before* v_{SS_i} *in a valid* topologic order *of the vertices of* G.

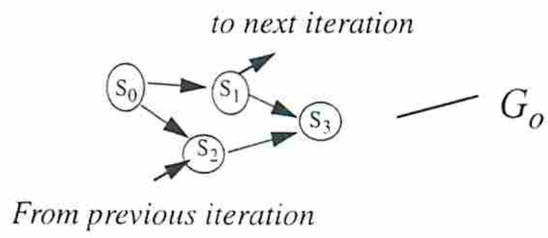
Proof: $v_{SE_i} \longrightarrow v_{SS_i}$ because of Constraint 4.11 □.

Lemma 6.4 *Given vertex* $v_{CE_i} \in G$ *associated with timing variable* $T_{CE}(DT_i)$ *and* $v_{CS_i} \in G$ *associated with* $T_{CS}(DT_i)$, *where* G *is an acyclic* timing constraint *graph,* v_{CE_i} *can be placed immediately before* v_{CS_i} *in a valid* topologic order *of the vertices of* G.

Proof: $v_{CE_i} \longrightarrow v_{CS_i}$ because of Constraint 4.12 □.

Algorithm 6.5 *Fast topological order for a directed acyclic graph task-flow graph* TFG *and a valid Boolean assignment for* Gen_α *and* Gen_Φ.

```
for each task S_i ∈ TFG do
        mark[S_i] ⟵ FALSE /*mark as not visited*/
for each data transfer DT_i ∈ TFG do
        mark[DT_i] ⟵ FALSE /*mark as not visited*/
Topol ⟵ {} /*Topol is a list */
Topol[1] ⟵ v_SYS /*v_SYS is made source vertex*/
k ⟵ 2
for each task S_i ∈ TFG do
        /* visit tasks and data transfers preceding S_i */
        if not mark[S_i] then DFS_topol(S_i, k)
end
```



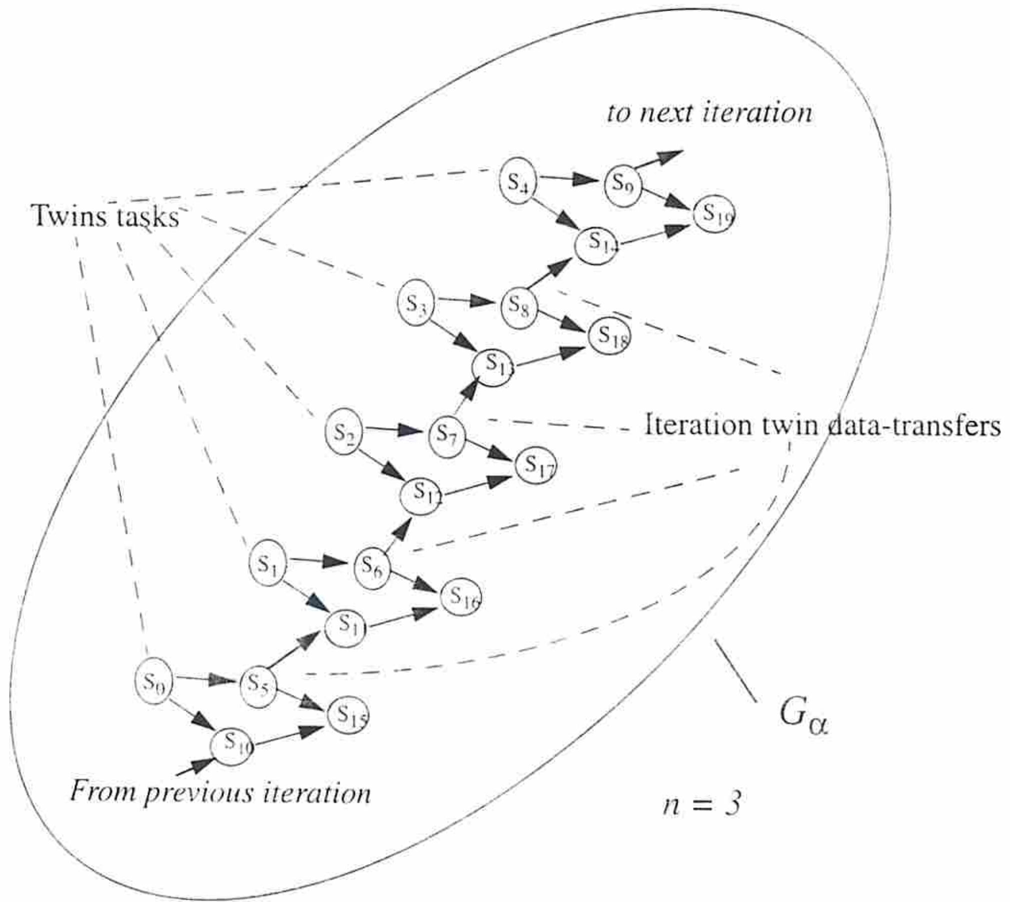


Figure 6.1: Unfolded graph G_α

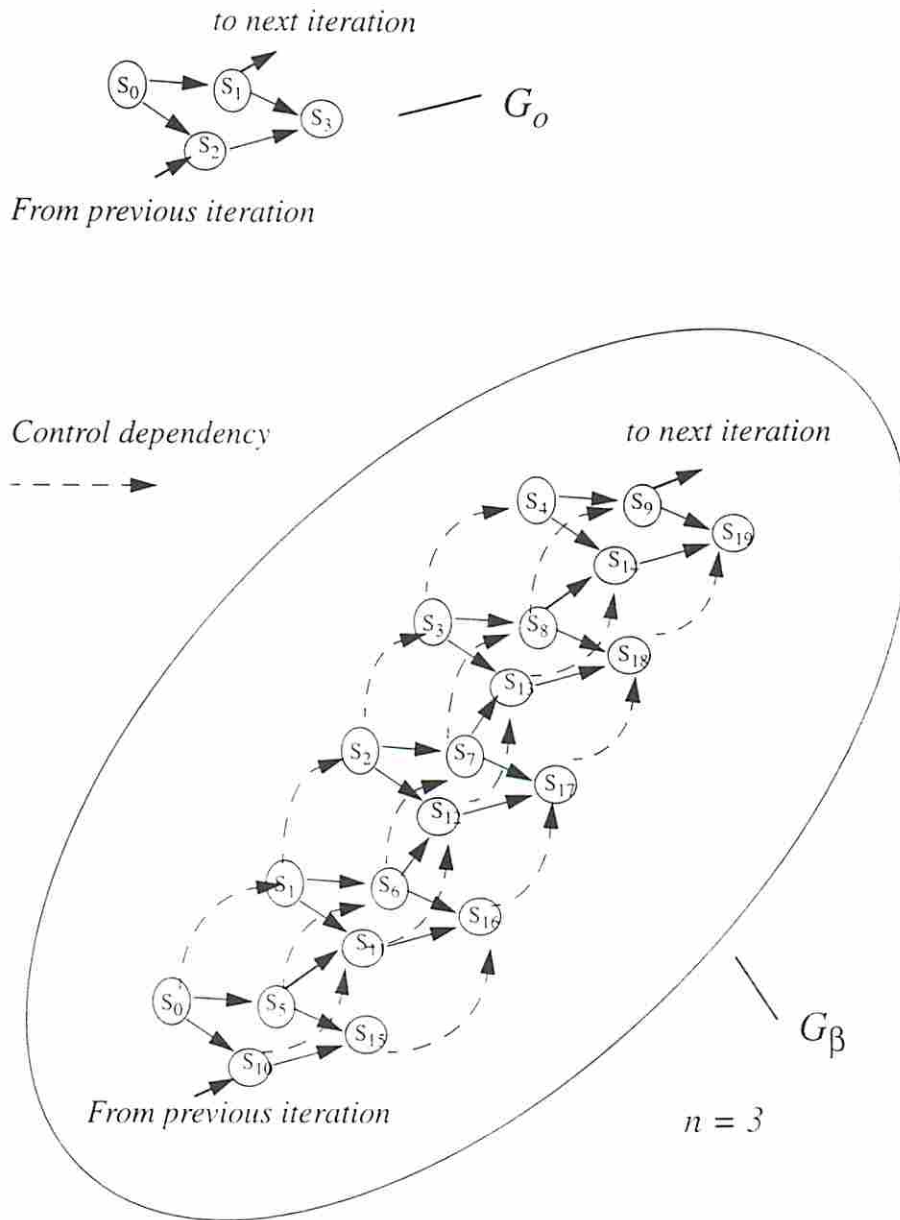


Figure 6.2: Synchronous unfolded graph G_3

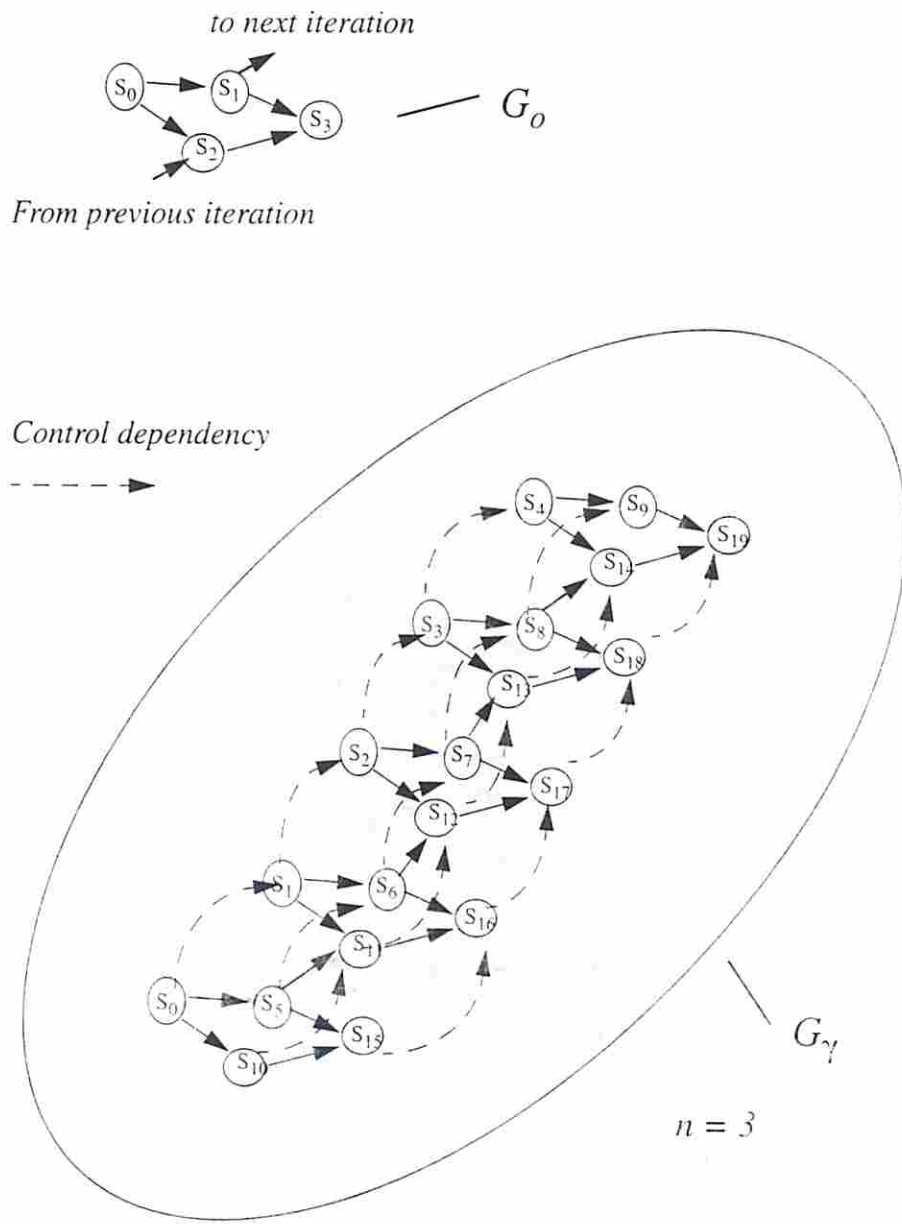


Figure 6.3: Myopic synchronous unfolded graph G_γ

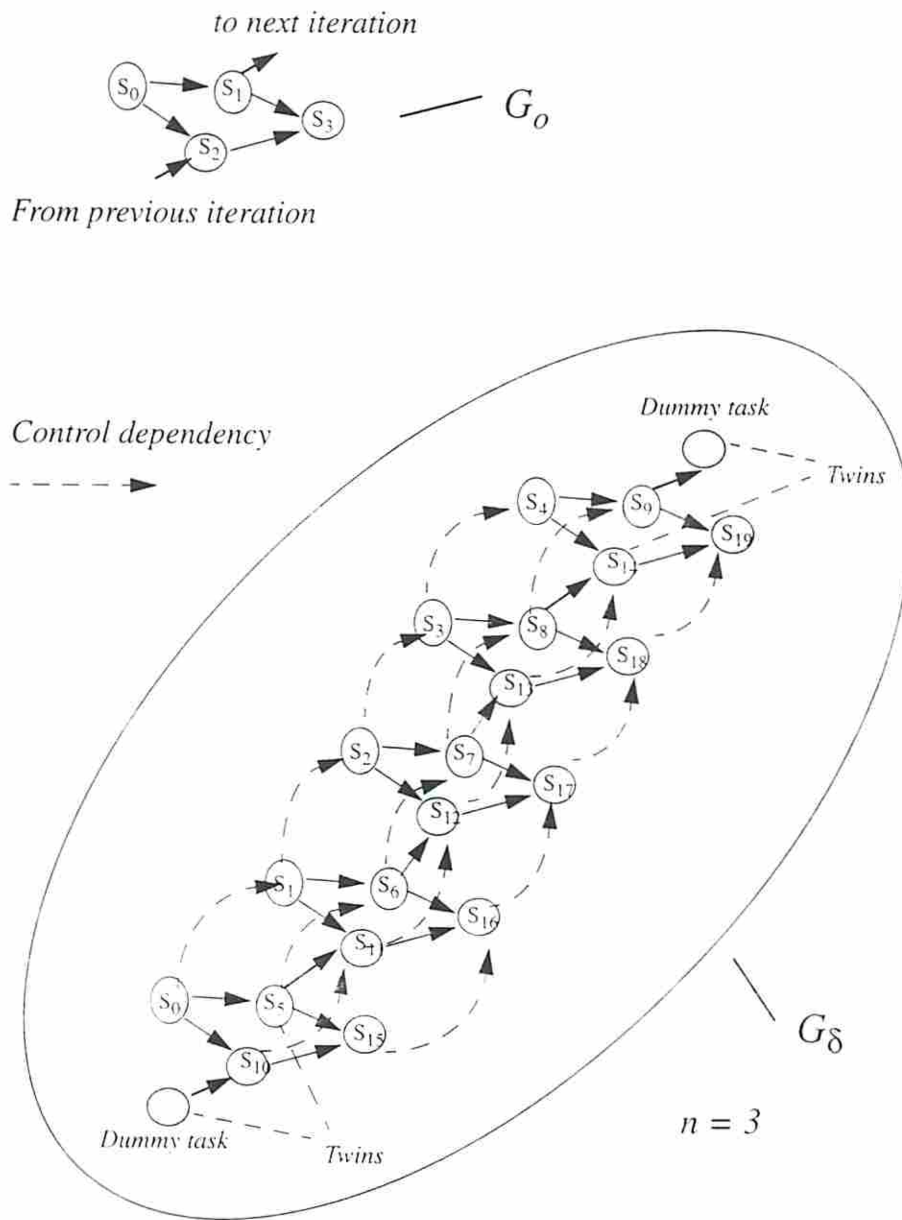


Figure 6.4: Bounded unrolled graph G_δ

Algorithm 6.6 $DFS_{topol}(u, k)$

$mark[u] \longleftarrow TRUE$ /*mark u as visited*/
if vertex u is a task then
  Let be $u = S_j$
  $Topol[k] \longleftarrow v_{SE_j}$ /* end of task S_j */
  $Topol[k+1] \longleftarrow v_{SS_j}$ /* start of task S_j */
  $k \longleftarrow k+2$
  for all incoming data transfers DT_i *to* S_j
    /* visit tasks and data transfers preceding S_i */
    if **not** $mark[DT_i]$ *then* $DFS_{topol}(DT_i, k)$
  /*visit precedding parallel tasks for the given Boolean assignment */
  for all tasks S_i // $S_j \mid \theta_i = \theta_j$ *and* S_i *is scheduled before* S_j
    if **not** $mark[S_i]$ *then* $DFS_{topol}(S_i, k)$
else if vertex u is a data transfer then
  Let $u = DT_j$
  $Topol[k] \longleftarrow v_{CE_j}$ /* end of data trasnfer DT_j */
  $Topol[k+1] \longleftarrow v_{CS_j}$ /* start of data trasnfer DT_j */
  $k \longleftarrow k+2$
  $S_i = source(DT_j)$
    /* visit tasks and data transfers precedding S_i */
    if **not** $mark[S_i]$ *then* $DFS_{topol}(S_i, k)$
  /*visit precedding parallel tasks for the given Boolean assignment */
  for all data transfers DT_i // $DT_j \mid \pi_i = \pi_j$ *and* DT_i *is scheduled before* DT_j
    if **not** $mark[DT_i]$ *then* $DFS_{topol}(DT_i, k)$

end

Theorem 6.1 *Algorithm 6.5 returns a topologic order for the vertices in the* timing constraint graph.

Proof. The algorithm inserts vertices according a depth first search order, where the children are inserted after the parents in *Topol*. As the *timing constraint graph* is a DAG, the list *Topol* corresponds to a topologic order. □.

6.11 Detecting Cycles

Let there be a directed acyclic graph $G = (V, E)$ and an edge $e \in E$. The following algorithm can be used to detect if a cycle is formed if the direction of e is reversed:

Algorithm 6.7 *Cycle detection*

Let be $e = u \longrightarrow v$
Let $G' = G - e$
if there is a path from u *to* v *in* G' *then*
  A cycle will be formed by reversing e *in* G.
else
  No cycle will be formed.

end

A *depth first search algorithm* [19] can be used to detect a path from u to v in G'. Algorithm 6.7 will be used in the *mutation* operator to detect if a *mutation* in one of the genes in the sets Gen_α or Gen_Φ leads to an infeasible solution.

6.12 ASAP-ALAP Based Scheduling Heuristics

The following heuristic is used for non-periodic scheduling of task-flow graphs during the generation of the initial population and after *catastrophic* changes in the value assignments given to the *discrete genes* such as during the application of a *crossover* operator. The same heuristic was adapted to handle the *macro-pipelined finite horizon* case by scheduling the correspondent *bounded unrolled* graph of a task-flow graph to be executed in a periodic mode of execution.

Definition 6.10

Given a task-flow graph $G = (V, E)$, *where* V *correspond to the set of tasks and* E *to the set of data transfers. The* generalized flow graph $G_{gf}(G) = (V', E')$ *is derived from* G *by applying the following transformations:*

i. $V' = V \cup E$ and $E' = \{\}$. In other words, each data transfer DT_i also becomes a vertex in $G_{gf}(G)$.

ii. For each data transfer $e \in E$, $e = DT_i = S_u \longrightarrow S_v$, include the following control-dependencies (zero data volume) as edges in E':

$$e_1(e) = S_u \longrightarrow DT_i$$
$$e_2(e) = DT_i \longrightarrow S_v \tag{6.18}$$

Lemma 6.5 *If G is a directed acyclic graph then the same is true for $G_{gf}(G)$.*

Proof: The edges $e_1(e)$ and $e_2(e)$, derived from the original edge e in G, have the same direction as e. Therefore no cycles are created during the transformation.

Definition 6.11 *The* time duration $\tau(v)$ *of a vertex v in $G_{gf}(G)$ is given by*

$\tau(v) = \omega_{p_j}(S_k)$ if $v = S_k$ for $\theta_k = i$ and $\vartheta_i = j$, where p_j is a processor type and $\omega_{p_j}(S_k)$ the computation time of S_k in p_j.

$\tau(v) = Delay_{b_j}(DT_k)$ if $v = DT_k$ for $\pi_k = i$ and $\varpi_i = j$, where b_j is a bus (data-link) type and $Delay_{b_j}(D_k)$ is the communication delay of DT_k in b_j.

Definition 6.12 *The* start time *(*end time*) $T_S(v)$ $(T_E(v))$ of a vertex v in $G_{gf}(G)$ corresponds to*

$T_{SS}(S_i)$ $(T_{SE}(S_i))$ if $v = S_i$.

$T_{CS}(DT_i)$ $(T_{CE}(DT_i))$ if $v = DT_i$.

Definition 6.13 *The set of predecessors of v is given by*

$$pred(v) = \{u \mid e \in E' \quad e = u \longrightarrow v\} \tag{6.19}$$

Definition 6.14 *The set of successors of v is given by*

$$succ(v) = \{u \mid e \in E' \quad e = v \longrightarrow u\} \tag{6.20}$$

The following algorithms are used to evaluate the *as soon as possible* and *as late as possible* start times for all vertices in $G_{gf}(G)$, where no *non-overlap constraints* among tasks or data transfers are taken into account.

Algorithm 6.8 As Soon As Possible *algorithm for $G_{gf}(G)$.*

For all v in V''

$T_S(v) \longleftarrow 0$

mark v as unscheduled.

repeat

Choose a vertex v which all predecessors are already scheduled.

*/*Schedule v just after all predecessors finish */*

$T_S(v) \longleftarrow max_{u \in pred(v)}(T_S(u)) + \tau(u)$

mark v as scheduled
if v is a data transfer with dummy-twins then
Let $v = DT_i$
For all dummy-twins $u = DT_j$ *of v such that* $\pi_i = \pi_j$
/* *Merge* dummy-twin *data transfers in the same bus* */
$T_S(u) \longleftarrow T_S(v)$
mark u as scheduled
until all vertices are scheduled.
$\Omega_{SYS} = max_{v \in V'}(T_S(v) + \tau(v))$

Algorithm 6.9 As Late As Possible *algorithm for* $G_{gf}(G)$.
Evaluate Ω_{SYS} *by* As Soon As Possible *algorithm.*
For all v in V'
$T_E(v) \longleftarrow \Omega_{SYS}$
mark v as unscheduled.
repeat
Choose a vertex v which all successors are already scheduled.
/**Schedule v just before all sucessors start* */
$T_E(v) \longleftarrow min_{u \in succ(v)}(T_E(u)) - \tau(u)$
mark v as scheduled
if v is data transfer with dummy-twins then
Let $v = DT_i$
For all dummy-twins $u = DT_j$ *of v such that* $\pi_i = \pi_j$
/* *Merge* dummy-twin *data transfers in the same bus* */
$T_E(u) \longleftarrow T_E(v)$
mark u as scheduled
until all vertices are scheduled.

end

Definition 6.15 *The* As Soon As Possible *start time* $T_{asap}(v)$ *for the vertex v is equal to* $T_S(v)$ *as evaluated by Algorithm 6.8.*

Definition 6.16 *The* As Late As Possible *start time* $T_{alap}(v)$ *for the vertex v is equal to* $T_E(v) - \tau(v)$*, where* $T_E(v)$ *is evaluated by Algorithm 6.9.*

Definition 6.17 *The* slack interval *of a vertex in* $G_{gf}(G)$ *is given by* $[T_{asap}(v), T_{alap}(v)]$.

Definition 6.18 *The following* selection criteria *are used for scheduling a vertex v of* $G_{gf}(G)$ *in decreasing order of importance, where a vertex v to be selected must have all predecessors already scheduled, i.e. v is* ready:

i. v has the earliest $T_{asap}(v)$ among ready *vertices.*

ii. v has the smallest slack interval.

iii. v has the highest number of successors.

Heuristic 6.1 *The following heuristic describes the scheduling of a task-flow graph G given its* generalized flow graph $G_{gf}(G) = (V', E')$ *and a valid assignment for gene types:* θ, ϑ, π *and* ϖ, *representing the allocation of tasks and data transfers.* Non-overlap *constraints are enforced.*

> *Evaluate the* slack interval *of all vertices in* $G_{gf}(G)$.
> *for all vertices v of* $G_{gf}(G)$
>   *mark v as* unscheduled
> **repeat**
>   *Choose a* ready *vertex v according to the* selection criteria *6.18*
>   *mark v as scheduled*
>   *if* $v = S_i$ *then /* v is a task */*
>   *for all* $u = S_j$ *such that* S_j // S_i *and* $\theta_i = \theta_j$
> $\alpha_{i,j} \longleftarrow 1$
> $T_S(u) \longleftarrow max(T_S(u), T_S(v) + \tau(v))$
>   *if* $v = DT_i$ *then /* v is a data transfer */*
>   *for all dummy-twin data transfers* $u = DT_j$
> $T_S(u) \longleftarrow T_S(v)$
>   *mark u as scheduled*
>   *for all* $u = DT_j$ *such that* DT_j // DT_i *and* $\pi_i = \pi_j$
> $\Phi_{i,j} \longleftarrow 1$
> $T_S(u) \longleftarrow max(T_S(u), T_S(v) + \tau(v))$
>   *if the* T_{asap} *of at least one unscheduled vertex changed*
>   *then /*a few* slack intervals *have to be updated */*
>   *Recalculate* slack intervals *of unscheduled vertices if necessary.*
> **until** *all vertices are scheduled*

end

6.12.1 Macro-Pipelined Finite Horizon Case

Given a a task-flow graph G, it is possible to apply Heuristic 6.1 to the correspondent *bounded unrolled graph* G_δ of G, as can be seen below.

Heuristic 6.2 *The following heuristic describes the scheduling of a task-flow graph G given a valid assignment for genes types: θ, ϑ, π and ϖ, representing the allocation of tasks and data transfers.* Non-overlap *constraints are enforced.*

Derive bounded unrolled graph G_δ *of* G

Create generalized flow graph $G_{gf}(G_\delta)$

Apply Heuristic 6.1 to $G_{gf}(G_\delta)$

end

6.13 Detailed Timing Information and Bellman-Ford Algorithm

Given the values of genes of types θ, ϑ, π and ϖ and genes in the sets Gen_α and Gen_{Phi}, as well as values of constants of types $r^i_{out_k}$ and $r^i_{in_k}$, as discussed in Section 1.2.1.15, the value of the timing variables is calculated by the following algorithms based on variations of the Bellman-Ford *single-source shortest paths* algorithm subject to the following constraint:

Restriction 6.1 *For each set of* dummy-twin data transfers $DTwins_{b_r}(DT_j)$ *such that*

$$DTwins_{b_r}(DT_j) = \{DT_k \mid DT_k \; is \; dummy-twin \; of \; DT_j \; with \; \pi_k = \pi_j = b_x\}$$

make vertices $v_{CS_i} = v_{CS_j}$ and $v_{CE_i} = v_{CE_j}$ if $DT_i \in DTwins_{b_r}(DT_j)$. In other words the vertices associated with start (end) times of dummy-twin data transfers allocated to the same bus become synonymous in the relaxed timing constraint graph *associated with a task-flow graph G.*

6.13.1 Non-Periodic Case

Algorithm 6.10 *Given a task-flow graph G a valid assignment for genes θ, ϑ, π and ϖ and genes in the sets Gen_α and Gen_{Phi}, the values of constants of types $r^i_{out_k}$ and $r^i_{in_k}$ and* timing constraints *discussed in Section 1.2.1:*

Find **relaxed timing constraint graph** *corresponding G^{relax}_{constr}.*

Apply Algorithm 4.4 (a variation of Bellman-Ford single-source shortest paths *algorithm discussed on Chapter 4) on G^{relax}_{constr}.*

6.13.2 Macro-Pipelined Finite Horizon Case

Similar to Algorithm 6.10, the following algorithm is an extension for the *macro-pipelined finite horizon case*:

Algorithm 6.11 *Given a task-flow graph G, an* overlapping factor n_{over}, *its corresponding* **bounded unrolled graph** G_δ, *a valid assignment for genes* θ, ϑ, π *and* ϖ *and genes in the sets* Gen_α *and* Gen_{Phi}, *the values of constants of types* $r^i_{out_k}$ *and* $r^i_{in_k}$ *and* timing constraints *discussed in Section 3.3.5:*

$T_I \longleftarrow max(T^0_{min}, LB(T_I))$, *where* T^0_{min} *is defined in Section 4.3.2.1.*

Find G^{relax}_{constr} **relaxed timing constraint graph** *of* G_δ, *where the following constraints among consecutive replications* S_a *and* S_{a+1} (DT_a *and* DT_{a+1}) *are inserted in* G^{relax}_{constr}.

$$T_{SE}(S_{a+1}) - T_{SE}(S_a) \geq T_I$$

$$T_{SS}(S_{a+1}) - T_{SS}(S_a) \geq T_I$$

$$T_{CE}(DT_{a+1}) - T_{CE}(DT_a) \geq T_I$$

$$T_{CS}(DT_{a+1}) - T_{CS}(DT_a) \geq T_I$$

repeat /**Relax till* T_I *converges and* $n_{over} * T_I \geq T_{SYS}$ */

Apply Algorithm 4.4 on G^{relax}_{constr}.

Find system latency T_{SYS} *for tasks in* G_δ *corresponding to task instances in the base iteration of* G_o

$T^{new}_I \longleftarrow max(T_I, \frac{T_{SYS}}{n_{over}})$

For each task $S_i \in G$ *and its corresponding* replications $S_a, S_{a+1}, \ldots$ G_δ, *as discussed in Section 6.9.1.*

For each replication S_a

$T^{new}_I \longleftarrow max(T^{new}_I, T_{SS}(S_{a+1}) - T_{SS}(S_a))$

$T^{new}_I \longleftarrow max(T^{new}_I, T_{SE}(S_{a+1}) - T_{SE}(S_a))$

For each data transfer $DT_i \in G$ *and its corresponding* replications $DT_a, DT_{a+1}, \ldots$ *in* G_δ, *as discussed in Section 6.9.1.*

For each replication DT_a

$T^{new}_I \longleftarrow max(T^{new}_I, T_{CS}(DT_{a+1}) - T_{CS}(DT_a))$

$T^{new}_I \longleftarrow max(T^{new}_I, T_{CE}(DT_{a+1}) - T_{CE}(DT_a))$

if $T^{new}_I > T_I$

/* *Replications are not equidistant yet**/

$T_I \longleftarrow T^{new}_I$

Update G^{relax}_{constr}

until **all replications are equidistant by** T_I

end

6.14 Profile Functions

Definition 6.19 *Given two numbers $a, b \in \mathcal{R}$, where R is the set of real numbers and $a \leq b$, and an integer number $m \geq 1$. The function $h_{prof}(i, a, b, m)$ is a* profile function *in the domain $1 \leq i \leq m$, if it has the following properties:*

i. $h_{prof}(1, a, b, m) = a$ and $h_{prof}(m, a, b, m) = b$

ii. if $i < j$ then $h_{prof}(i, a, b, m) \leq h_{prof}(j, a, b, m)$

Definition 6.20 *Given three numbers $a, b, m \in \mathcal{Z}$, where Z is the set of integer numbers, $a \leq b$ and $m \geq 1$. The function $h_{prof}(i, a, b, m)$ is a* integer profile function *in the domain $1 \leq i \leq m$, if it has the following properties:*

i. $h_{prof}(i, a, b, m)$ is a *profile function.*

ii.$h_{prof}(i, a, b, n) \in \mathcal{Z}$ for $1 \leq i \leq m$

The following function is an example of an integer profile function:

$$g(i, a, b, m) = \begin{cases} a & \text{if } i = 1 \\ \lfloor \frac{(b-a)*i}{m} \rfloor & \text{if } 1 < i < m \\ b & \text{if } i = m \end{cases} \tag{6.21}$$

6.15 Generating the Initial Population

A initial population $\mathcal{P}(0)$ with a high *genetic diversity* increases the chances of a genetic algorithm to converge to the *global optimum*. The same is true for MEGA, which adopts the following procedures and concepts when generating the solutions in $\mathcal{P}(0)$.

Definition 6.21 *The* processor (bus) selection criterion *used to decide the allocation of task S_i (data transfer DT_i) during the generation of $\mathcal{P}(0)$ is one of the following:*

i. The cheapest processor (bus) to which S_i (DT_i) can be allocated.

ii. The fastest processor (bus), i.e. one with lowest *computation (communication) time*, to which S_i (DT_i) can be allocated.

iii. The processor (bus) that maximizes a *trade-off cost versus performance function* for S_i (DT_i), w where a typical trade-off function g_{trade} is defined as:

$$g_{trade}(y,z) = \begin{cases} A * Cost(p_x) + B * \omega_{p_x}(S_i) \\ \text{if } y = S_i \text{ is a task and } z = p_x \text{ is a processor} \\ \\ A * Cost(b_x) + B * Delay_{b_x}(DT_i) \\ \text{if } y = DT_i \text{ is a data transfer and } z = b_x \text{ is a bus (data link)} \end{cases} \tag{6.22}$$

where $A, B, C, D \geq 0$

iv. Random selection of a processor (bus) to which the tasks can be allocated.

Definition 6.22 *The* probabilistic profile *of the initial population is given by the probabilities* $p_{cheapest}, p_{fastest}, p_{trade}$ *and* p_{random}, *that respectively indicate the probability that a task (bus) will be allocated with one of the four available* processor (bus) selection criteria:

$$\begin{aligned} 0 \leq p_{cheapest}, p_{fastest}, p_{trade}, p_{random} \leq 1.0 \\ p_{cheapest} + p_{fastest} + p_{trade} + p_{random} = 1.0 \end{aligned} \tag{6.23}$$

Algorithm 6.12 *Generate population P(0) with m initial solutions given a task-flow graph* $G = (V, E)$, *a probabilistic profile* ($p_{cheapest}, p_{fastest}, p_{trade}$ *and* p_{random}), *profile functions* [6.1] $f_{processor}$ *and* f_{bus}, *and a trade-off function* g_{trade}.

/* *Create m initial solutions* */
for $i \longleftarrow 1$ *to* m *do*
    $n_{proc} \longleftarrow f_{processor}(i, 1, |V|, m)$
/**Mapping tasks for initial solution i* */
    $np \longleftarrow 0$
    mark all tasks as unallocated
    Processor-set $\longleftarrow \{\}$
    for $j \longleftarrow 1$ *to* $|V|$ *do*
        Choose an unallocated *task* S_r *at random given*

[6.1] Profile functions are defined in Section 6.14

$p_{cheapest}, p_{fastest}, p_{trade}$ *and* p_{random}
mark S_r *as allocated.*
Choose a selection criteria *at random*
if $np < n_{proc}$ *then*
Allocate a new processor p_x
Choose the best type for p_x *according to*
selection criteria *and* S_r.
Allocate S_r *to* p_x
Processor-set $\longleftarrow$ *Processor-set* + p_x
$np \longleftarrow np + 1$
else /* $np \geq n_{proc}$ */
Choose a processor $p_x \in$ Processor-set *according to*
selection criteria *and* S_r.
Allocate S_r *to* p_x
/* *Mapping non-local (remote) data transfers for initial solution i* */
$ndt \longleftarrow$ *number of remote data-transfers for given task mapping.*
$j \longleftarrow$ **random integer** *number between* 1 *and* ndt
$n_{bus} \longleftarrow f_{bus}(j, 1, ndt, ndt)$
$nb \longleftarrow 0$
mark all remote data transfers as unallocated
Bus-set $\longleftarrow \{\}$
for $j \longleftarrow 1$ *to* $|E|$ *do*
Choose an unallocated *remote data transfer* DT_r *at random given*
$p_{cheapest}, p_{fastest}, p_{trade}$ *and* p_{random}
mark DT_r *as allocated.*
Choose a selection criteria *at random*
if $nb < n_{bus}$ *then*
Allocate a new bus b_x
Choose the best type for b_x *according to*
selection criteria *and* DT_r.
Allocate DT_r *to* b_x
Bus-set $\longleftarrow$ *Bus-set* + b_x
$nb \longleftarrow nb + 1$
else /* $nb \geq n_{bus}$ */
Choose a bus $b_x \in$ Bus-set *according to*
selection criteria *and* DT_r.
Allocate DT_r *to* b_x
Apply ASAP-ALAP heuristics[6.2] *to find values of genes in* Gen_α *and* Gen_Φ
Find detailed timing as discussed in Section 6.13.
/* *end of for* $i \longleftarrow 1$ *to* m */

[6.2] Discussed in Section 6.12

Procedure 6.1 Choose a selection criteria *at random*

$x \longleftarrow$ *random real number in the interval* $[0, 1]$
if $x \leq p_{cheapest}$
  return **the cheapest**
else if $x \leq p_{cheapest} + p_{fastest}$
  return **the fastest**
else if $x \leq p_{cheapest} + p_{fastest} + p_{trade}$
  return one with **the best trade-off**
else
  return **random-choice**

6.16 Genetic Operators

MEGA uses the *mutation* and *crossover* operators tailored to the system-level design problem. These operators were carefully designed in order to decrease the chances of creation of infeasible solutions.

6.16.1 Mutation

There are six types of genes in the chromosome representation used by MEGA. Therefore, a mutation corresponds to a random change of one of these genes.

6.16.1.1 θ and ϑ Genes

Let task S_i be assigned to processor p_j with type p_{t_k}, i.e. $\theta_i = p_j$ and $\vartheta_j = p_{t_k}$, for solution Sol_q.

Definition 6.23 *The set* $\mathcal{FA}(S_i, Sol_q)$ *is set of available processors in solution* Sol_q *to which* S_i *can be allocated without creating an* infeasibility, *where an infeasible choice can be one of the following:*

i. Processor p_y has a type to which S_i cannot be assigned, i.e. $\omega_{p_y}(S_i) = \infty$, where $\omega_{p_y}(S_i)$ is the computation time of task S_i on processor p_y, and $\omega_{p_y}(S_i) = \infty$ denotes that S_i cannot be executed on p_y.

ii. It is not possible to assign a bus (data-link) between p_y and other processors to (from) which S_i sends (receives) data transfers due to *topologic constraints* of the interconnection network as discussed in Section 6.4.5.

In both cases

$$p_y \notin \mathcal{FA}(S_i, Sol_q) \tag{6.24}$$

Definition 6.24 *The* mutation of the gene θ_i *is a random choice of a new assignment of* S_i *to one of the processors in* $\mathcal{FA}(S_i, Sol_k) - p_j$ *where* p_j *is the current processor to which* S_i *is assigned.*

Definition 6.25 *The set* $\mathcal{FT}(p_j, Sol_q)$ *is set of feasible types of processor* p_j *in solution* Sol_q *to which all tasks* $S_u, \ldots, S_v$ *assigned to* p_j *can execute, i.e.*

$$\mathcal{FT}(p_j, Sol_q) = \{ \; types \;\; p_{t_i} \mid \vartheta_j = p_{t_i} \Longrightarrow \omega_{p_j}(S_u), \ldots, \omega_{p_j}(S_v) < \infty \} \tag{6.25}$$

Definition 6.26 *The* mutation of the gene ϑ_j *is a random choice of a new type for* p_j *from one of the available types in* $\mathcal{FT}(p_j, Sol_q) - p_{t_k}$.

6.16.1.2 π and ϖ Genes

Let the remote data transfer DT_i be assigned to bus (data-link) b_j with type b_{t_k}, i.e. $\pi_i = b_j$ and $\varpi_j = b_{t_k}$. on solution Sol_q.

Definition 6.27 *The set* $\mathcal{FA}(DT_i, Sol_q)$, $DT_i = S_u \longrightarrow S_v$, *is the set of available buses in solution* Sol_q *to which* DT_i *can be allocated without creating an infeasible solution. A bus* b_s *that cannot connect the source and sink processors of the data transfer due to* topologic constraints, *as discussed in Section 6.4.5, cannot be a member of* $\mathcal{FA}(DT_i, Sol_q)$.

Definition 6.28 *The* source *(*sink*) processor of a data transfer* $DT_i = S_u \longrightarrow S_v$ *is one to which the source task* S_u *(sink task* S_v*) is assigned.*

Definition 6.29 *The* mutation of the gene π_i *is a random choice of a new assignment of* DT_i *to one of the buses (data-links) in* $\mathcal{FA}(DT_i, Sol_q) - b_j$.

Definition 6.30 *The set* $\mathcal{FT}(b_j, Sol_q)$ *is set of feasible types for bus (data-link)* b_j.

Definition 6.31 *The* mutation of the gene ϖ_j *is a random choice of a new type for* b_j *from one of the available in* $\mathcal{FT}(b_j, Sol_q) - b_{t_k}$.

6.16.1.3 Gen_α and Gen_Φ Sets

The genes in these sets, which are responsible for the encoding of *relative* scheduling information, are binary, i.e. their values are either 0 or 1.

Definition 6.32 *A valid mutation of a gene $\alpha_{ij} \in Gen_\alpha$ ($\Phi_{ij} \in Gen_\Phi$) is given by*

$$\alpha_{ij} \longleftarrow 1 - \alpha_{ij}$$
$$(\ \Phi_{ij} \longleftarrow 1 - \Phi_{ij} \ ) \tag{6.26}$$

if this does not introduce a cycle in the corresponding generalized flow graph[6.3] *of the task-flow graph G to be implemented.*

6.16.1.4 Probabilistic Profile in Mutation

A mutation in MEGA can be one of six possible types: θ, ϑ, π, ϖ, Gen_α and Gen_Φ. For each occurrence of a mutation, the probability that it will be of the one of these types is given respectively by: p_θ, p_ϑ, p_π, p_ϖ, p_α and p_Φ, where:

$$0 \le p_\theta, p_\vartheta, p_\pi, p_\varpi, p_\alpha, p_\Phi \le 1.0 \tag{6.27}$$

$$p_\theta + p_\vartheta + p_\pi + p_\varpi + p_\alpha + p_\Phi = 1.0 \tag{6.28}$$

A possible assignment for p_θ, p_ϑ, p_π, p_ϖ, p_α and p_Φ would be:

$$p_\theta = \frac{|S|}{N_{total}} \tag{6.29}$$

$$p_\vartheta = \frac{NPROC^{max}}{N_{total}} \tag{6.30}$$

$$p_\pi = \frac{|E|}{N_{total}} \tag{6.31}$$

$$p_\varpi = \frac{NBUS^{max}}{N_{total}} \tag{6.32}$$

[6.3] Discussed in Section 6.12

$$p_\alpha = \frac{|Par_S|}{N_{total}} \tag{6.33}$$

$$p_\Phi = \frac{|Par_{DT}|}{N_{total}} \tag{6.34}$$

$$N_{total} = |S| + |E| + NPROC^{max} + NBUS^{max} + |Par_S| + |Par_{DT}| \tag{6.35}$$

where

i. $NPROC^{max}$ ($NBUS^{max}$) is the maximum allowed number of processors (buses) as defined in Section 6.4.4.

ii. $Par_S = \{(S_i, S_j) \mid S_i, S_j \in TFG \;.\; S_i // S_j \;\; and \;\; i < j\}$, i.e. the set of pairs of parallel tasks in the TFG.

iii. $Par_{DT} = \{(DT_i, DT_j) \mid DT_i, DT_j \in TFG \;.\; DT_i // DT_j \;\; and \;\; i < j\}$, i.e. the set of pairs of parallel data transfers in the TFG.

iv. $|S|$ is the number of tasks in the task-flow graph.

v. $|E|$ is the number of data transfers in the task-flow graph.

6.16.1.5 Feasibility after Mutation

The following properties can be derived by inspection from the definitions of the six types of mutations.

Property 6.8 *A mutation in a gene of type ϑ or ϖ generates a solution with feasible allocation. However the values of the genes in sets Gen_α and Gen_Φ may correspond to an infeasible solution after mutation. There is need to reschedule some of the tasks and data transfers in the mutated solution.*

Property 6.9 *A mutation in a gene from set Gen_α or Gen_Φ generates a feasible solution as long as a cycle in the generalized flow graph is not created.*

6.16.2 Performing Crossover

The operator *crossover* in MEGA views a chromosome as a collection of *zones*, where each zone is associated with a subset of the Boolean variables in one the

MILP models for system-level design design in the previous chapters. The minimal amount of genetic information exchanged during a crossover is a *zone*, e.g.

$$chromosome \quad Z = z_0 z_1 \cdots z_m$$
$$chromosome \quad Y = y_0 y_1 \cdots y_m$$

After applying crossover to zones z_1 and y_1.

$$chromosome \quad Z = z_0 y_1 \cdots z_m$$
$$chromosome \quad Y = y_0 z_1 \cdots y_m$$

The gene set types $Alpha(i, j)$ and $Phi(i, j)$ for the macro-pipelined finite-horizon case are examples of possible zones, as discussed in Section 6.6.7.2.

Differently from the *mutation* operator, which is implemented as a random value change in one of the genes, crossover can be understood as a set of changes in each one of the parents that need to be propagated over their entire chromosomes of the offspring solutions in order to ensure feasibility.

6.16.2.1 Tasks Crossover

Definition 6.33 *Given two parent solutions (chromosomes) Sol_1 and Sol_2, a* task kernel $\mathcal{K}_S(p_i, p_j) = \{S_a, \ldots S_z\}$ *is a subset of the tasks in the TFG to be implemented, such that*

i. All tasks in $\mathcal{K}_S(p_i, p_j)$ are respectively assigned to processors p_i and p_j in solutions Sol_1 and Sol_2, i.e.

$$\begin{aligned} \theta_a = \ldots = \theta_z = p_i \quad in \quad Sol_1 \\ \theta_a = \ldots = \theta_z = p_j \quad in \quad Sol_2 \end{aligned} \tag{6.36}$$

ii. $\mathcal{K}_S(p_i, p_j)$ is maximal for the given pair of processors p_i and p_j, i.e. there is not other set $\mathcal{K}'_S(p_i, p_j)$ whose elements also respect restriction 6.36 and $\mathcal{K}_S(p_i, p_j) \subset \mathcal{K}'_S(p_i, p_j)$.

Definition 6.34 *Each task kernel $\mathcal{K}_S(p_i, p_j)$ defines a zone composed of genes $\theta_a = \ldots = \theta_z$ and the subset of α-genes $\mathcal{C}_\alpha(\mathcal{K}_S(p_i, p_j)) = \{\alpha_{ij} \mid S_i, S_j \in \mathcal{K}_S(p_i, p_j)$ and S_i is parallel $(//)$ to $S_j\}$.*

Definition 6.35 *A* task crossover *is defined by the exchange of the corresponding zones of one or more task kernels chosen at random between two solutions (chromosomes)* Sol_1 *and* Sol_2.

Property 6.10 *A task crossover generates offspring with feasible allocations. However their respective* relative schedules *may be infeasible.*

6.16.2.2 Data-Transfer Crossover

In a similar way, the concepts of a *data-transfer* crossover can be defined.

Definition 6.36 *Given two parent solutions (chromosomes)* Sol_1 *and* Sol_2, *a* data-transfer kernel $\mathcal{K}_{DT}(b_i, b_j) = \{DT_a, \ldots DT_z\}$ *is a subset of the tasks in the TFG to be implemented, such that:*

i. All data transfers in $\mathcal{K}_{DT}(b_i, b_j)$ are remote and they are respectively assigned to buses (data-links) b_i and b_j in solutions Sol_1 and Sol_2, i.e.

$$\begin{aligned} \pi_a = \ldots = \pi_z = b_i \;\; in \;\; Sol_1 \\ \pi_a = \ldots = \pi_z = b_j \;\; in \;\; Sol_2 \end{aligned} \tag{6.37}$$

ii. $\mathcal{K}_{DT}(b_i, b_j)$ is maximal for the given pair of buses b_i and b_j, i.e. there is not another set $\mathcal{K}'_{DT}(b_i, b_j)$ whose elements also respect restriction 6.37 and $\mathcal{K}_{DT}(b_i, b_j) \subset \mathcal{K}'_{DT}(b_i, b_j)$

Property 6.11 *Each data-transfer kernel* $\mathcal{K}_{DT}(b_i, b_j)$ *defines a zone composed of genes* $\pi_a = \ldots = \pi_z$ *and* $\bigcup Phi(i, j)$ *such that* $DT_i, DTj \in \mathcal{K}_{DT}(b_i, b_j)$, *where* DT_i *is parallel to* DTj.

Definition 6.37 *A* data-transfer crossover *is defined by the exchange of the corresponding zones of one or more data-transfer kernels chosen at random between two solutions (chromosomes)* Sol_1 *and* Sol_2.

6.16.2.3 Feasibility of Offsprings

In order to ensure feasible schedules, the ASAP-ALAP heuristics discussed in Section 6.12 are applied to the offspring subject to the following modified *selection criteria* of a *ready* task (data transfer) vertex $v = S_i$ ($v = DT_i$) of the generalized flow graph ($G_{gf}(G)$):

i. All other ready vertices $u = S_j$ ($u = DT_j$), members of the same kernel to which v is also member and parallel to v ($u//v$), have $\alpha_{ij} = 1$ ($\Phi_{ij} = 1$), i.e. u is supposed to be scheduled after v.

ii. v has the *earliest* $T_{asap}(v)$ among *ready* vertices.

iii. v has the smallest slack interval among ready vertices.

iv. v has the highest number of successors among ready vertices.

6.16.2.4 Probabilistic Profile

The probability that a *crossover* will be a task (data-transfer) crossover is given by p_c^{task} (p_c^{dt}), subject to

$$0 \leq p_c^{task}, p_c^{dt} \leq 1.0 \tag{6.38}$$

$$p_c^{task} + p_c^{dt} = 1.0 \tag{6.39}$$

A possible assignment for p_c^{task}, p_c^{dt} would be

$$\begin{aligned} p_c^{task} &= \frac{|V|}{|E|+|V|} \\ p_c^{dt} &= \frac{|E|}{|E|+|V|} \end{aligned} \tag{6.40}$$

where $G = (V, E)$ is the task-flow graph to be implemented.

6.16.2.5 Crossing Index

The maximum number of kernels considered, i.e. number of exchanged zones, in a crossover is a a design parameter called *crossing index* that can be decided by the user. A higher *crossing* index indicates a more aggressive crossover operator.

6.17 Managing Infeasible Solutions

As it would be computationally expensive to enforce feasibility for some constraints (e.g. very restrictive topologic constraints). MEGA allows the existence of infeasible solutions by setting them to a very low fitness value in order to discourage their propagation over future populations.

6.18 Managing Memory

Since the chromosomes in MEGA are regular structures of bounded size and the *population* is a unidimensional array (vector) of chromosomes, it is possible to reuse the space allocated to *dead* chromosomes, i.e. chromosomes that will not be part of population of the next iteration. Offspring of the mutation and crossover operations can be assigned the space of the *dead* chromosomes. This minimizes problems with dynamic allocation of memory while executing MEGA.

6.19 An Overview of the MEGA Algorithm

The following brief algorithmic description of MEGA gives an overview of its organization and how the algorithms discussed in this chapter are used within MEGA.

Algorithm 6.13 *MEGA - a genetic algorithm for system-level design of application specific multiprocessors.*

Read input data including the task-flow graph $G = (V, E)$
Transform multicasts in G to dummy-twin data transfers.
Detect parallelism among tasks (data transfers) using **transitive closure** *[19]*
Generate initial population $\mathcal{P}(0)$
$t \longleftarrow 0$
repeat
Perform **mutation** *in a few solutions of* $\mathcal{P}(t)$
Perform **crossover** *in a few solution pairs of* $\mathcal{P}(t)$
Find **detailed timing** *using Bellman-Ford based algorithms discussed in Section 6.13.*

Evaluate fitness *of population* $\mathcal{P}(t)$

Scale and normalize *fitness values*

Generate $\mathcal{P}(t+1)$ *from* $\mathcal{P}(t)$ *by a* **selection scheme**

$t \longleftarrow t+1$

until *convergence to a near-optimal or optimal design (solution)*

Generate output data (Gannt Chart)

6.20 Experimental Results

The tool MEGA was written in C. and it has approximately 6,800 line of code. A set of benchmarks was used to compare the performance of MEGA and SOS [100] on a SUN4-SPARC workstation for the *deterministic non-periodic* paradigm. Five designs are shown (Table 6.3). In all five designs MEGA was able to reach the optimal solution found by SOS.

Each design is optimal with respect to the following objective function:

$$min \;\; f \tag{6.41}$$

where

$$f = W_{Cost} * Cost + W_{T_{SYS}} * T_{SYS} \tag{6.42}$$

W_{Cost} is the *cost weight* and $W_{T_{SYS}}$ is the *system latency weight*. Different values of W_{Cost} and $W_{T_{SYS}}$ correspond to different points of the cost *versus* performance (*system-latency* T_{SYS}) trade-off curve.

Designs I and II (Table 6.3) are optimal solutions for *task flow graph A* (Figure 6.5). Designs III. IV and V are solutions for *task flow graph B* (Figure 6.5). The execution time of each task and the processor costs for the three available *processor types* are available for both *task flow graphs* (Tables 6.1 and 6.2). The processor allocation is shown (Table 6.3).

In all five designs. at most one shared bus is allowed. because the present implementation of the SOS tool set for shared bus design styles handles no more than one shared bus. allowing a fair comparison of MEGA against SOS. MEGA allows an unlimited number of shared buses. It is assumed that the data volume being transferred between tasks is equal to 1 (one) unit of data for all *data transfers*,

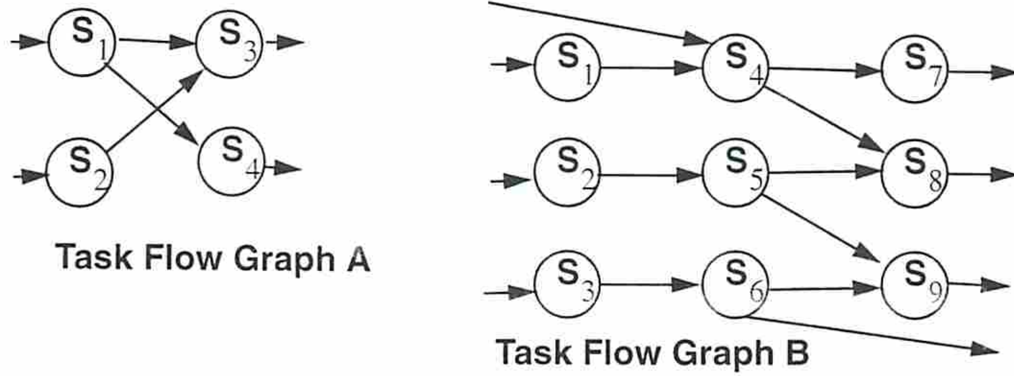


Figure 6.5: Task flow graphs

Processor types	Processor Cost	S_1	S_2	S_3	S_4
P_1	4	1	1	--	3
P_2	5	3	1	2	1
P_3	2	--	3	1	--

Table 6.1: Processor costs and task execution times - Task Flow Graph A

Processor types	Processor Cost	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
P_1	4	2	2	1	1	1	1	3	-	1
P_2	5	3	1	1	3	1	2	1	2	1
P_3	2	1	1	2	--	3	1	4	1	3

Table 6.2: Processor cost and task execution times - Task Flow Graph B

which implies a communication delay of 1 unit of time for all inter-processor *data transfers*, i.e. the *bus speed* is assumed to be equal to one. Intra-processor (local) data transfers have zero delay. These values, however, can be easily changed by the designer and they are used here only for demonstration purposes. The *selection scheme* used in MEGA in all experiments is *proportionate reproduction* [105]. The population size is fixed in the range of 100 to 200 chromosomes with mutation (p_m) and crossover (p_c) probabilities in the range of 0.1 to 0.2.

Design	Task Flow Graph	#Tasks	Weight T_{SYS}	Weight Cost	Run-time[a] SOS	Run-time[a] MEGA	Ratio of run-times	T_{SYS}	Cost	#P$_1$	#P$_2$	#P$_3$
I	A	4	100	1	13 s	0.8 s	16.25	4	6	1	0	1
II	A	4	1	100	4 s	1.5 s	2.67	7	5	0	1	0
III	B	9	100	1	107.3 min	18.0 s	357.7	6	10	2	0	1
IV	B	9	1	1	89.53 min	2.6 s	2066	7	6	1	0	1
V	B	9	1	100	61.52 min	2.8 s	1318	15	5	0	1	0

a. Run time on a SUN-SPARC4 workstation

Table 6.3: Solutions - Cost and Latency trade-off

The experimental results indicate that MEGA outperforms SOS in the five examples (Table 6.3). The final schedule and allocation for Design I is presented in Figure 6.6. The run-time of solving the MILP models generated by SOS is around one hour for a small task-flow graph of nine tasks (TFG B), using BOZO [50] as the MILP solver. MEGA is able to reach the same optimal solution in less than 20 seconds in all cases. The run-time of MEGA does not increase as fast with the complexity of the problem as SOS does. MEGA allows the MILP mathematical model outlined in Chapter 1 to be applied to examples to which exact tools such as SOS would not be able to find an optimal solution in reasonable time.

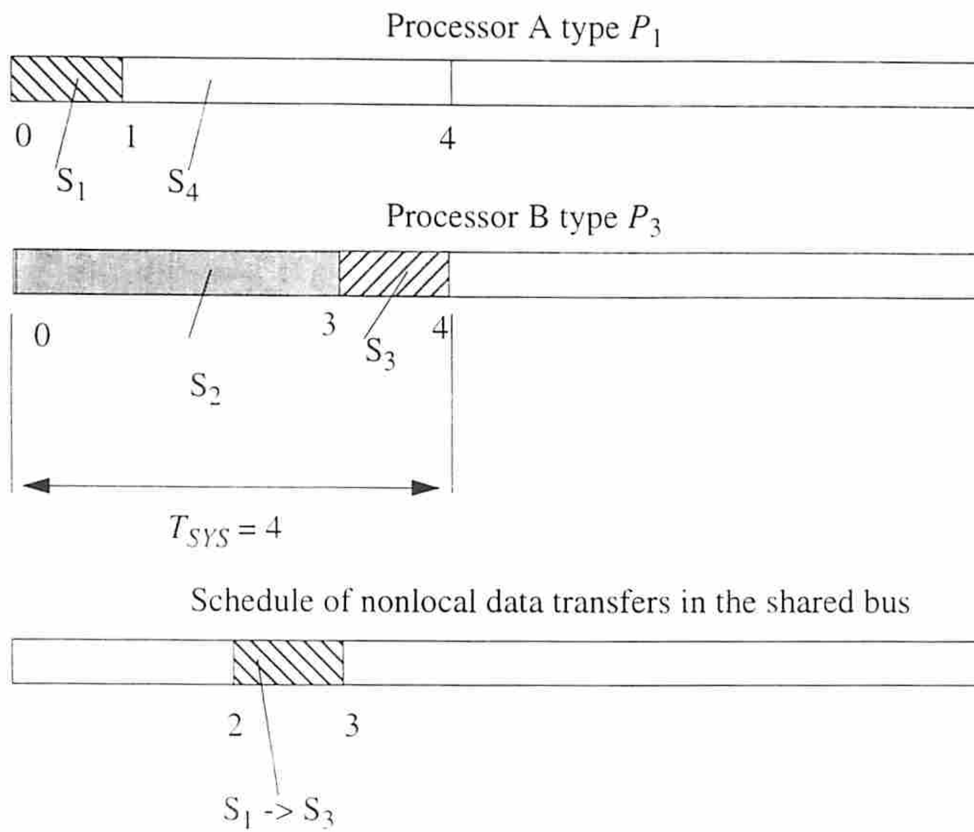


Figure 6.6: Gannt Chart for Design I

6.20.1 Comparing Different Selection Schemes

The following experiments confirm the rationale of using *proportionate reproduction* as the default selection scheme. Figures 6.7 and 6.8 compare the performance of the *proportionate reproduction* selection scheme (Section 5.2.7.1) against two other schemes: *tournament* of size 2 (Section 5.2.7.1) and *simulated annealing* (Section 5.2.7.1). The comparative experiment use a population size of 100 chromossomes with mutation (p_m) and crossover (p_c) probabilities equal to 0.1. The task flow graph to be optimized is TFG B (Design V - Table 6.3) described in Figure 6.5. The initial temperature for the simulated annealing selection scheme is $\theta_0 = 2000$ with an annealing factor $\kappa = 0.99$.

Figure 6.7, fitness of the best solution, indicates that *proportionate reproduction* has a faster initial convergence than *tournament of size 2*. Figure 6.8, fitness of the worst solution, indicates that *proportionate reproduction* has almost the same performance as *tournament of size 2*, however the former loses *diversity* at a slower pace than the latter, which may be helpful due the fact that the presence of solutions with lower fitness values decreases the possibility of a genetic algorithm being trapped in a local optimum. In both experiments (Figures 6.7 and 6.8), the *simulated annealing* has poor performance compared to the other two schemes.

6.21 Summary of the Chapter

The different concepts, algorithms and heuristics used in the MEGA system, a genetic algorithm and MILP-model-oriented tool set for system-level design, were shown. Emphasis was given to proving the correctness of the assumptions used in building the algorithms and heuristics used within MEGA.

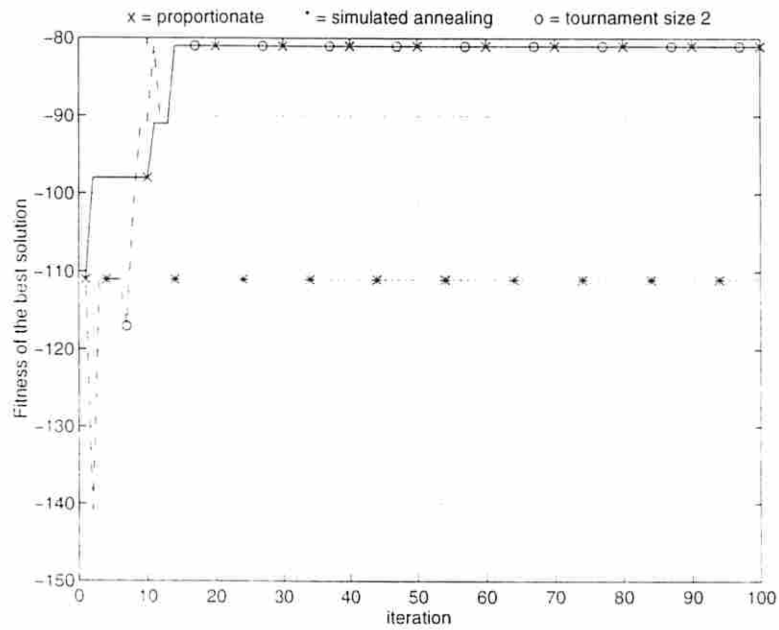


Figure 6.7: Best-solution fitness convergence for different selection schemes

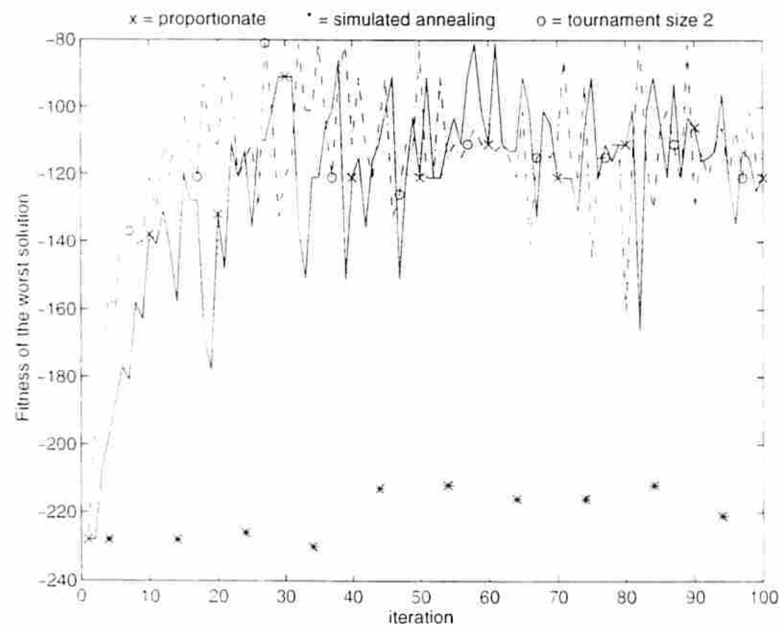


Figure 6.8: Worst-solution fitness convergence for different selection schemes

Chapter 7

Imprecise Computation

7.1 Motivation

The term *imprecise computation* was proposed by Liu et al. [79] to model real-time computations such as video processing, where it is possible to make a trade-off between quality of data (image) and computation time of particular video processing algorithms (e.g. video compression).

In the *non-preemptive imprecise computation* model assumed in this dissertation, each task S_i is divided in two parts: *mandatory* (S_i^m) and *optional* (S_i^o), as seen in Figure 7.1. These parts correspond to subtasks subject to the following constraints:

i. The *optional* part must follow the *mandatory* part without interruption.

ii. They are allocated to the same processor, i.e., they behave as *twin tasks*, as discussed in Chapter 5.

iii. The *mandatory* part must be allowed to execute until completion.

iv. The *output* and *input* data volumes for the task are independent of the amount of processing allowed for the *optional* part.

The result of a task is said to be a *precise* result if its optional part is allowed to compute until completion. By allowing *imprecise computation* of some tasks, it may be possible to meet hard time deadlines with a low implementation cost. The *imprecise execution* model assumes that the *quality* of the output data being produced by a task is a non-decreasing function of the amount of processing allowed

for the optional part. The *imprecise computation* paradigm allows an extra fine-grain trade-off between overall data *quality*, performance and cost.

Typical applications of imprecise computation would include *compression* algorithms such as *fractal compression*, digital signal processing transforms such as those used for finding the wavelet components of a non-periodic signal, or sub-band decomposition of video/audio signals. All these applications can model the *output data quality* as a function of the *processing time* expended by their respective algorithms.

Ergonomic studies have shown that human beings [11] are more sensitive to *audio quality* than to *video quality*. At the same time, *luminance quality* is more important than *chromatic quality* for video. These findings may be helpful when designing low-cost mass-production application-specific multiprocessor systems for real-time video/audio processing. Imprecise computation is able to represent these issues in a sound mathematical form.

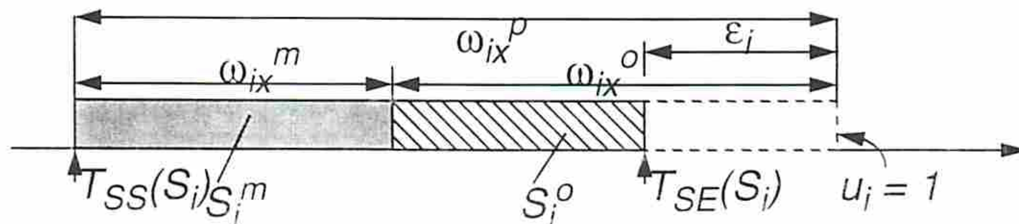


Figure 7.1: Modeling non-preemptive imprecise computation

7.2 Mathematical Formulation

As seen in Figure 7.1, the *imprecise computation* paradigm assumes that each task S_i in the task flow graph can be divided to two: *a mandatory* part S_i^m and an *optional*

one S_i^o, where the output data being generated by S_i has a *quality factor* Q_i, which is a non-decreasing function of the *utilization factor* u_i, $0 \leq u_i \leq 1$, where $u_i = 1$ means that S_i^o is executed till completion.

The *utilization factor* is the ratio of execution time S_i^o is allowed to execute over the time taken to completion. The key point is to trade the overall *quality factor* Q_{SYS} for a less expensive implementation.

$$Q_{SYS} = \sum_i Q_i(u_i) \tag{7.1}$$

where $Q_i(\bullet)$ is a non-decreasing function of the *utilization factor*. which might be nonlinear.

7.3 An MILP Formulation

Assuming $Q_i(u_i) = k_i u_i$. this leads to

$$Q_{SYS} = \sum_i k_i u_i \tag{7.2}$$

The normalized overall quality factor is

$$NQ_{SYS} = \frac{\sum_i k_i u_i}{\sum_i k_i} \tag{7.3}$$

$$NQ_{SYS} = 1 \quad for \quad u_i = 1 \quad \forall S_i \tag{7.4}$$

where k_i is a measure of the relative importance of the execution of S_i^o on the overall quality of output data to be produced by the multiprocessor system.

As it was said before, the imprecise computation model is useful in applications such as real-time video frame processing, where some task processing may be not be fully completed in order to meet some hard performance constraints when using a low-cost implementation. The quality of the image frames being generated by such a system will be affected by the choices of the values given to the utilization factors u_i.

An application-specific system designer might prefer not to have a preemption mechanism in exchange for a less-costly implementation using dedicated processors.

The proposed MILP model is for imprecise computation without preemption. The notation described used here is the same as in Chapters 1 and 3.

Let

$$\omega(S_i) = T_{SE}(S_i[k]) - T_{SS}(S_i[k]) \tag{7.5}$$

$T_{SE}(S_i[k])$ is the end of execution time for task S_i in iteration k
$T_{SS}(S_i[k])$ is the start of execution time for task S_i in iteration k

$$\omega^p_{S_i} = \sum_x \sigma_{i\,x}\omega^p_{i\,x} \tag{7.6}$$

Where $\omega^p_{i\,x}$ is the computation time of task S_i when allocated to processor x and allowed to execute till completion, i.e., the computation time of task S_i in the *precise computation* mode of execution.

$$\omega^p_{i\,x} = \omega^m_{i\,x} + \omega^o_{i\,x} \tag{7.7}$$

and $\omega^m_{i\,x}$ and $\omega^o_{i\,x}$ are the mandatory and optional computation times for task S_i when allocated to processor x, which are assumed to be constant parameters supplied by the designer.

$$u_i = \frac{\omega(S_i) - \sum_x \sigma_{i\,x}\omega^m_{i\,x}}{\sum_x \sigma_{i\,x}\omega^o_{i\,x}} \tag{7.8}$$

where $u_i = 1$ if $\sum_x \sigma_{i\,x}\omega^o_{i\,x} = 0$.
The utilization factor u_i will be a nonlinear function of the Boolean allocation variables $\sigma_{i\,x}$. In order to have a MILP model for the problem the quality factor of a task S_i can be reformulated as a function of the *processing error* ϵ_i.

$$\epsilon_i = \omega^p_{S_i} - \omega(S_i) \tag{7.9}$$

The quality factor will be some function $Q^e_i(\epsilon_i)$ of the error ϵ_i. The overall quality factor Q^e_{SYS} will be

$$Q^e_{SYS} = \sum_i Q^e_i(\epsilon_i) \tag{7.10}$$

Assuming

$$Q_i^e(\epsilon_i) = -k_i^\epsilon \, \epsilon_i \;\; ; k_i^\epsilon > 0 \tag{7.11}$$

The normalized value of Q_{SYS}^e will be

$$NQ_{SYS}^e = \frac{-\sum_i k_i^\epsilon \, \epsilon_i}{\sum_i k_i^\epsilon} = -\sum_i a_i \, \epsilon_i \tag{7.12}$$

$$where \quad a_i = \frac{k_i^\epsilon}{\sum_i k_i^\epsilon} \tag{7.13}$$

Therefore the MILP model for the case for imprecise computation of aperiodic non-preemptive tasks is given by

$$maximize \;\; NQ_{SYS}^e = -\sum_i \sum_x a_i \sigma_{i\,x} \omega_{i\,x}^p + \sum_i \sum_x a_i T_{SE}(S_i) - \sum_i \sum_x a_i T_{SS}(S_i) \tag{7.14}$$

subject to the other constraints of the MILP models for the *non-periodic* or *macro-pipelined with finite horizon cases.*

7.4 Allowing Imprecise Computation in MEGA

In order to allow the use of a genetic formulation for the problem of system-level design with *imprecise computation*, a set of a *utilization factor* genes is added to the chromosome representation used in MEGA for the *precise computation* paradigm.

7.4.1 Utilization Factors as Genes

Definition 7.1 *The* utilization factor u_i *gene takes real values in the range* $[0, 1]$ *and it is associated with the* utilization factor *of task* S_i.

A naive definition of the operator *mutation* for a real-valued gene in the range $[a, b]$ would be a random choice of a real number in the interval. In a similar way, the crossover of two real-valued genes can be defined as the exchange (swap) of their values.

7.4.1.1 Trap Function

Experimental results with an earlier version of MEGA using a naive definition of the *mutation* operator of showed that the chances to reach an optimal/near-optimal design (solution) with a particular utilization factor u_i equal to 1.0 or 0.0, are minimal. Instead u_i would assume fractional values of the form $1.0 - \epsilon$ or ϵ, where ϵ is a small number, e.g. $\epsilon = 0.05$. In order to avoid that, the *mutation* operator is redefined using the concept of *trap function.*

Definition 7.2 *For a given a random number x in the interval $[a, b]$, $b > a$, and z, the* trap length, *where $2 * z < b - a$. The trap function $f_{trap}(x, z, a, b)$ is defined as*

$$f_{trap}(x, z, a, b) = \begin{cases} a & \text{if } a \leq x < a + z \\ \frac{(x-z-a)*(b-a)}{b-a-2*z} + a & \text{if } a + z \leq x < b - z \\ b & \text{if } b - z \leq x \leq b \end{cases} \quad (7.15)$$

7.4.2 Redefining Mutation in MEGA

Definition 7.3 *The mutation of a* utilization factor gene *is defined as a change of its value to $f_{trap}(x, z, 0, 1)$, where x is a random real number in $[0, 1]$, and $0 \leq z < 0.5$.*

Property 7.1 *The mutation of a binary gene corresponds to a trap function $f_{trap}(x, 0.5, 0, 1)$.*

7.4.2.1 Probabilistic Profile

The introduction of the *utilization factor gene mutation* increases the total number of possible mutations to seven as discussed in Section 6.16.1.4: u, θ, ϑ, π, ϖ, Gen_α and Gen_Φ.

A similar set of constraints as in Section 6.16.1.4 would apply in this case.

$$0 \leq p_u, p_\theta, p_\vartheta, p_\pi, p_\varpi, p_\alpha, p_\Phi \leq 1.0 \quad (7.16)$$

$$p_u + p_\theta + p_\vartheta + p_\pi + p_\varpi + p_\alpha + p_\Phi = 1.0 \quad (7.17)$$

A possible assignment for p_u, p_θ, p_ϑ, p_π, p_ϖ, p_α and p_Φ would be

$$
\begin{aligned}
p_u = p_\theta &= \frac{|V|}{N_{total}} \\
p_\vartheta &= \frac{NPROC^{max}}{N_{total}} \\
p_\pi &= \frac{|E|}{N_{total}} \\
p_\varpi &= \frac{NBUS^{max}}{N_{total}} \\
p_\alpha &= \frac{|Par_S|}{N_{total}} \\
p_\Phi &= \frac{|Par_{DT}|}{N_{total}}
\end{aligned}
\tag{7.18}
$$

$$N_{total} = 2 * |S| + |E| + NPROC^{max} + NBUS^{max} + |Par_S| + |Par_{DT}|$$

7.4.3 Crossover

The same concepts of *task* and *data-transfer* crossovers, as defined in Section 6.16.2.1, apply to *imprecise computation*. However the associated zone of a task kernel is redefined as

Definition 7.4 *Each task kernel $\mathcal{K}_S(p_i, p_j) = \{S_a, \ldots S_z\}$ defines a zone composed of genes $\theta_a = \ldots = \theta_z$, $u_a, \ldots, u_z$ and $\{\alpha_{ij} \mid S_i, Sj \in \mathcal{K}_S(p_i, p_j) \; S_i//Sj\}$.*

7.4.4 Fitness Value

Fitness value is defined as a function of the cost, system latency (T_{SYS}) initiation interval T_I and overall data quality.

$$f_{fit}(x) = F_{max} - W_{cost} * Cost(x) - W_{T_{SYS}} * T_{SYS}(x) - W_{T_I} * T_I(x) + W_{Q_{SYS}} * Q_{SYS}(x) \tag{7.19}$$

$$0 \leq W_{cost}, W_{T_{SYS}}, W_{T_I}, W_{Q_{SYS}} \tag{7.20}$$

where F_{max} is a sufficiently large number to ensure that $f_{fit}(x) \geq 0$ for all possible solutions, $W_{T_l} = 0$ for the *non-periodic* case, $Cost(x)$ is the overall cost of the solution (design) x, and Q_{SYS} is a function of the values of utilization factor genes, not necessarily linear. Moreover, Q_{SYS} is a nondecreasing function of the quality factors.

Property 7.2 *Given two vectors of utilization factor value assignments* $\mathbf{U}_i = (u_1^i, \ldots, u_{|S|}^i)$ *and* $\mathbf{U}_{ii} = (u_1^{ii}, \ldots, u_{|S|}^{ii})$ *for genes* $u_1, \ldots, u_{|S|}$. *If* $\mathbf{U}_i < \mathbf{U}_{ii}$ *then* $Q_{SYS}(u_1^i, \ldots, u_{|S|}^i) \leq Q_{SYS}(u_1^{ii}, \ldots, u_{|S|}^{ii})$, *where* S *is the set of tasks in the task flow graph.*

7.5 Improving the Quality by Postprocessing

Given a value assignment for the *utilization factor genes* of a design implementing a directed acyclic task flow graph $G = (V, E)$, it is possible to recalculate the start and end times of tasks and data transfers in such a way that the system latency is kept the same and the utilization factors of a few tasks are increased, leading to an improvement of the overall *data quality*, as illustrated by the following heuristic, where $\tau(v)$ denotes the computation (communication) time of the task (data transfer) v.

Heuristic 7.1 *Increasing the utilization factors after performing* **detailed timing.**
All timing variables are assumed to be positive.

- *For the given value assignment to genes in sets* Gen_α *and* Gen_Φ.
  - *Let* $G_{gf}(G) = (V', E')$ *the generalized flow graph of* G *taking into account* Gen_α *and* Gen_Φ.
  - **Find** *the topologic order of* $G_{gf}(G) = (V', E')$
- *For all* v *in* V'
  - **mark** v *as not visited.*
- $i \longleftarrow 0$
- **repeat** *in topologic order*
  - $i \longleftarrow i + 1$
  - *Choose i-vertex* v *of the topologic order.*
  - $T_S(v) \longleftarrow min(T_S(v), max_{u \in pred(v)}(T_E(u)))$
  - $T_E(v) \longleftarrow min(T_E(v), T_S(v) + \tau(v))$
  - *mark* v *as visited*
  - *if* v *is a data-transfer with twins then*
    - *Let* $v = DT_i$
    - *For all twins* $u = DT_j$ *of* v *such that* $\pi_i = \pi_j$

$T_S(u) \longleftarrow T_S(v)$
$T_E(u) \longleftarrow T_E(v)$
mark *u as visited*

until *all vertices are visited.*
Update utilization factors *of all tasks in G.*

Heuristic 7.1 takes advantage of the fact that the algorithms for finding *detailed timing* are based on variations of the Bellman-Ford single-source shortest paths algorithm, which generates a timing assignment corresponding to an ALAP schedule as discussed in Chapter 4.

7.5.1 An Overview of MEGA with Imprecise Computation

The following algorithm illustrates the extension of MEGA to the *imprecise computation* paradigm. The initial population is generated with all utilization factors equal to 1.0, due to the fact that previous experiments showed that it is difficult to reach an optimal solution in this region of the design space if MEGA allows utilization factors with values less than 1.0 in the initial population $\mathcal{P}(0)$.

Algorithm 7.1 *MEGA-ϵ - a genetic algorithm for system-level design of application specific multiprocessors allowing imprecise computation.*

Read input data including the task flow graph $G = (V, E)$
Transform multicasts in G to twin data-transfers.
Detect parallelism among tasks (data-transfers) using **transitive closure**
Generate initial population $\mathcal{P}(0)$ with all utilization factor gene values set to 1.0
$t \longleftarrow 0$
repeat
  Perform **mutation** *of a few solutions of $\mathcal{P}(t)$*
  Perform **crossover** *of a few solution pairs of $\mathcal{P}(t)$*
  Find **detailed timing** *using Bellman-Ford based algorithms discussed in Section 6.13.*
  if G is a DAG then
    Improve utilization factor *by applying Heuristic 7.1.*
  Evaluate fitness *of population $\mathcal{P}(t)$*
  Scale and normalize *fitness values*
  Generate $\mathcal{P}(t+1)$ from $\mathcal{P}(t)$ by a **selection scheme**
  $t \longleftarrow t + 1$
until *near-optimal or optimal design (solution) is found*
Generate output data (Gannt Chart)

7.6 Experimental Results

A set of benchmarks [99] (Figure 7.2) was adapted to evaluate the performance of MEGA with *imprecise computation*. Tables 7.1 , 7.2 and 7.3 provide detailed information about the tasks of the two task-flow graphs used as benchmarks as well as information regarding available processor and bus types. For the sake of simplicity, all data transfers are assumed to have a data volume equal to 1, with a negligible communication delay when *local*, where the overall delay of a non-local data transfer is given by

$$Delay_{remote}(DT, bus) = \tau_{bus} + \frac{Volume(DT)}{s_{bus}} \tag{7.21}$$

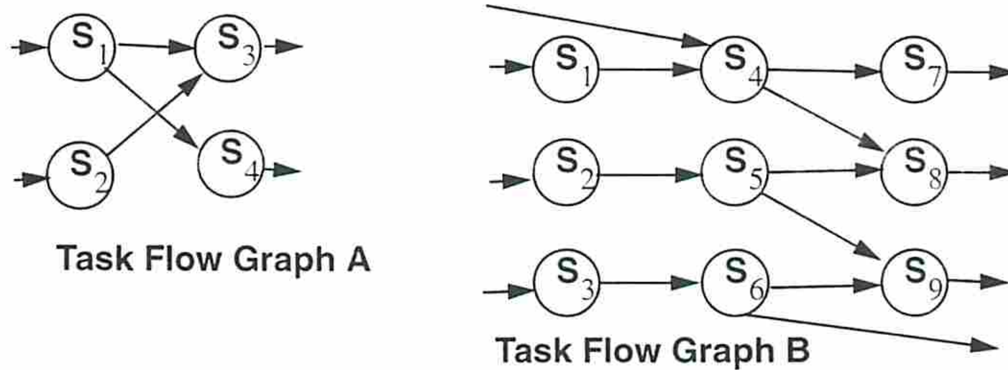


Figure 7.2: Task flow graphs

In all experiments, an *elitist* selection scheme [89] was used with mutation (p_m) and crossover (p_c) probabilities equal to 0.2 and a fixed size population of 100 chromosomes. Equation 7.4 is used in evaluating the overall *output-data quality*.

It can be seen from Figures 7.3 and 7.4 that the use of a small trap length z is helpful in speeding up the convergence to an optimal/near-optimal solution. This effect is more pronunciated for larger task-flow graphs (TFG B). Figure 7.5 and Table 7.4 give an idea of the design space for task-flow graphs A and B. Low quality solutions have associated low costs and latencies. An increase in quality requires more hardware resources. For a fixed cost. It is possible to have a lower latency by sacrificing the output-data quality. The run-time of MEGA was around a few minutes (< 5 min) on a SUN-SPARC4 to generate all points in Figure 6 compared

Processor		Tasks (S_i)	Computation time			
Type	.Cost		S_1	S_2	S_3	S_4
P_1	4	mandatory part	1.0	1.0	--	1.5
		optional part	0.0	0.0		1.5
P_2	5	mandatory part	1.5	1.0	1.3	0.5
		optional part	1.5	0.0	0.7	0.5
P_3	2	mandatory part	--	3.0	0.7	--
		optional part		0.0	0.3	
		Quality coefficients (k_i)	1.0	1.0	1.0	1.0

Table 7.1: Processor types - cost and performance information for TFG A

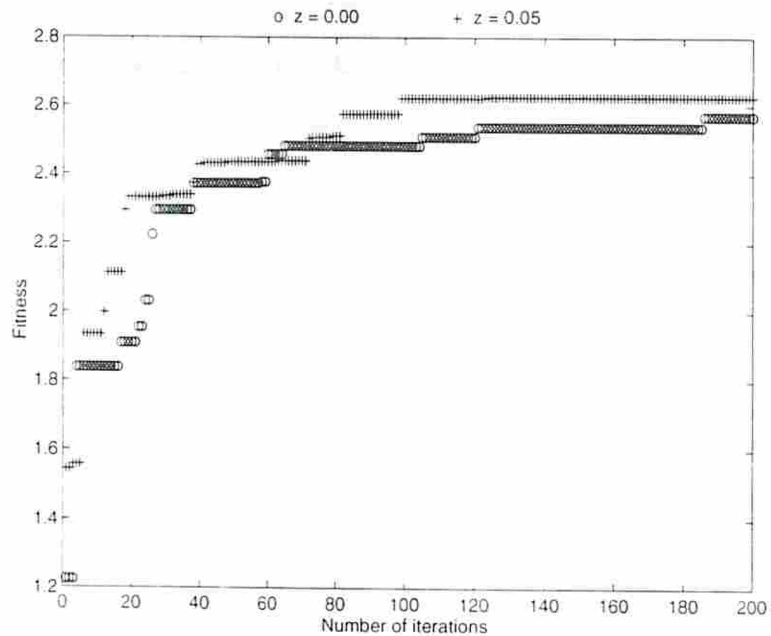


Figure 7.3: Effect of the trap length z (TFG A)

Processor		Tasks (S_i)	Computation time								
Type	Cost		S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
P_1	4	mandatory part	1.0	1.0	0.5	1.0	1.0	1.0	2.0	-	0.7
		optional part	1.0	1.0	0.5	0.0	0.0	0.0	1.0		0.3
P_2	5	mandatory part	1.5	0.5	0.5	3.0	1.0	2.0	0.7	1.3	0.7
		optional part	1.5	0.5	0.5	0.0	0.0	0.0	0.3	0.7	0.3
P_3	2	mandatory part	0.5	0.5	1.0	--	3.0	1.0	2.7	0.7	2.0
		optional part	0.5	0.5	1.0		0.0	0.0	1.3	0.3	1.0
		Quality coefficients (k_i)	2.0	2.0	2.0	1.0	1.0	1.0	0.5	0.5	0.5

Table 7.2: Processor types - cost and performance information for TFG B

Bus			
Type	Cost	Speed	Latency
B_1	10	1.0	0.0
B_2	20	3.0	0.0

Table 7.3: Buses types - cost and performance information

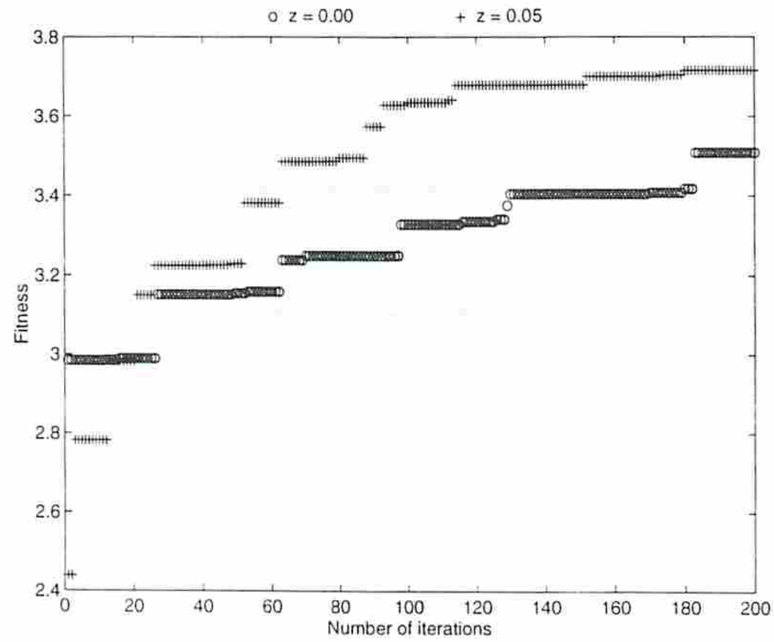


Figure 7.4: Effect of the trap length z (TFG B)

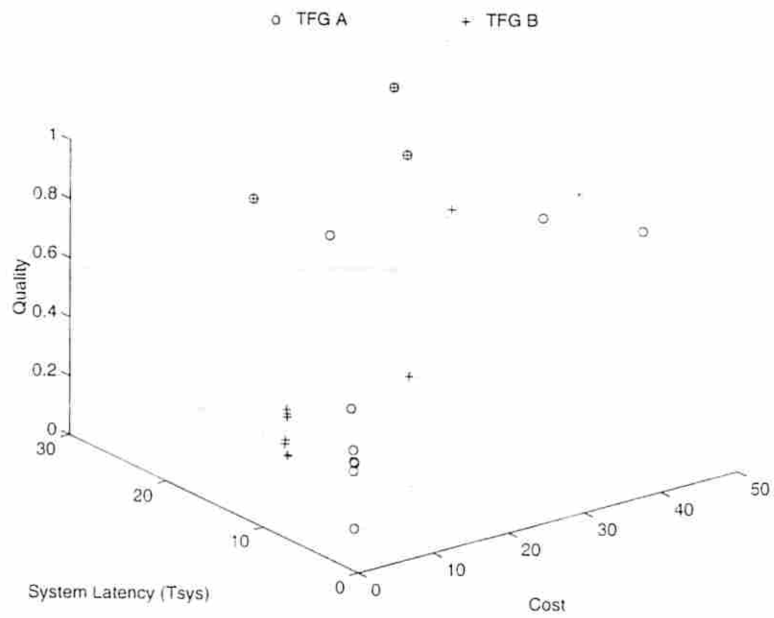


Figure 7.5: Trade-off points for the task flow graphs

		Number of Processors			Number of Buses		Trade-off points		
Design	TFG	#P1	#P2	#P3	#B1	#B2	Latency	Cost	Quality
I	A	0	1	0	0	0	4.33	5.0	0.05
II	A	0	1	0	0	0	4.46	5.0	0.31
III	A	0	1	0	0	0	4.71	5.0	0.45
IV	A	0	1	0	0	0	7.0	5.0	1.0
V	A	1	1	1	1	1	2.67	41	0.83
VI	B	0	1	0	0	0	11.5	5.0	0.33
VII	B	1	0	1	1	0	7.31	16.0	0.45
VIII	B	1	0	1	1	1	5.33	36.0	0.95
IX	B	1	0	1	1	0	15.33	26.0	1.0
X	B	2	0	1	1	0	6.0	20.0	1.0

Table 7.4: Selected Designs

hours using the SOS [99] approach, i.e optimization of MILP models restricted to the *precise computation paradigm.*

7.7 Summary of the Chapter

An MILP formulation was introduced to model the system-level design problem when imprecise computation is incorporated. The extensions of MEGA based on this MILP approach were presented. A new gene set was introduced: the set of *utilization factor genes*, along with their respective mutation and crossover operators. The objective function was redefined by adding data quality to the set of possible trade-offs. A postprocessing heuristic to improve (increase) the utilization factors after performing detailed timing was proposed. The overall structure of MEGA for the imprecise computation paradigm was shown.

Chapter 8

Probabilistic Approaches for System-Level Design

8.1 Motivation

A key issue in system-level design of ASHMs is that many times the design parameters such as cost of implementation of a processor, task execution time in a given processor and data transfer delays are not deterministic numbers. They can be either guesses provided by the designer, estimations made by prediction tools (e.g. area/performance predictors) [69] or benchmark evaluations. In such a situation, the designer may be not only interested in the worst-case scenario, but may prefer a design that meets the constraints with a certain probability. A practical example is a system for image compression following the MPEG standard [7]. Some types of frames, demanding large numbers of computations, are rather infrequent in a sequence of video-frames compressed according to MPEG. A worst-case scenario implementation would assume that they would arrive all the time, which would lead to an expensive implementation. A probabilistic approach would trade-off a non-zero probability p of not meeting some performance constraints for a less expensive design. A practical system-level design tool should be able to give different designs for different values of p, where $p = 0$ would correspond to the worst-case scenario.

Other possible examples of applications would include task-flow graphs with conditional branches, where different probabilities are associated with each one of

the possible branch outcomes. The worst-case approach would consider the path with the longest execution time instead of taking advantage of the probabilities associated with each branch. [8.1]

There are many formal methods to allow modeling of system-level design with probabilistically defined performance/cost parameters, among them queuing theory [124] and Petri nets [112]. These methods, however, lack the ability to easily express data dependencies among tasks in a task-flow graph as well as to evaluate possible *cost/performance* trade-offs. In this dissertation, an alternative approach was developed instead, where a formal method based on multivariable probability theory and MILP modeling is the basis for a probabilistic optimization method integrated into the tool set MEGA [127].

8.2 A Precise Mathematical Model for Handling Uncertainty

The analysis in this section is aimed at the case where the design parameters (processor costs, task execution times and data transfer delay times) are modeled as either random variables or predefined constants, where the set of performance parameters with non-zero variance defines a random column vector $X^{perf} = (x_1^{perf}, \ldots, x_{\dim(X^{perf})}^{perf})^t$ with a positive definite *covariance matrix* K^{perf}. Some emphasis is given to the situation where all random variables have a Gaussian distribution. A similar analysis can be done for other distributions. Both *pipelined* and *non-pipelined* cases are considered by applying *multivariable probability theory.* [8.2]

The sub-optimal approach discussed in Chapter 4 is used, i.e., the set of *design constraints* is reduced to a *system of difference constraints* once a particular value assignment is chosen for the Boolean variables of the MILP models used in *non-periodic* and *macro-pipelined* (periodic) cases. This approach allows the system-level problem in the presence of uncertainty to be become treatable, i.e. by avoiding the mix of the *discrete* genes value assignment problem with the evaluation of probability

[8.1]The BUD system [86], developed for data-path design by McFarland, is an example of this approach in high level synthesis by using the probability of execution of each operation to derive an *average clock cycle*.

[8.2]See the Appendix for a brief overview of multivariable probability theory.

distributions of the *timing* genes, which are embedded in the mathematical models solving the problem of simultaneous multiprocessor scheduling and allocation that has NP-complete time complexity even for the deterministic case.

8.2.1 Non-Pipelined Case

The Bellman-Ford *all single-source shortest paths algorithm* [19] cannot be directly applied when the performance parameters are non-deterministic, because the algorithm uses comparisons among the weights of paths arriving in each of the vertices v of the *constraint graph* $G = (V, E)$. The effect of these comparisons is highly nonlinear and hard to model when dealing with edge weights modeled as random variables.

The use of a Bellman-Ford based algorithm for the nondeterministic case could be replaced by taking advantage of the fact that the variable T_{SYS} (latency) corresponds to the *source node* v_0 of the *constraint graph* $G = (V, E)$, by using the following formalism:

Definition 8.1 *The set $\mathcal{P}_{v_0}^{v_{sink}}$ is defined to be the set of all possible paths from the* source node v_0 *to a sink node v_{sink}, which is added to the graph, where the weight of an edge connecting a node v to v_{sink} is zero.*

The weight $w(p)$ of a path in this set is given by the following equation:

$$w(p) = \sum_{i=0}^{dim(X^{perf})-1} u_i^p x_i^{perf} + h(p) \quad \forall p \;\; T_{SYS} \geq w(p) \tag{8.1}$$

where $u_i^p = 1$ *or* 0 is a constant indicating whether the performance parameter modeled by the random variable x_i^{perf} is counted or not in the path. The term $h(p)$ is a constant representing the sum of weights of edges that are not random variables. A minimum bound for T_{SYS} can be given by $T_{SYS}^{min} = h_{max}^p$.
where

$$h_{max}^p = max(h(p_0), \ldots, h(p_l)) \;\; , \;\; l = |\mathcal{P}_{v_0}^{v_{sink}}| \tag{8.2}$$

All paths that are *dominated* by other paths in $\mathcal{P}_{v_0}^{v_{sink}}$ are subtracted from the set, where p_i is dominated by p_j if $w(p_i) < w(p_j)$ for all possible values of the random variables x_k^{perf}, $0 \leq k \leq dim(X^{perf}) - 1$.

Definition 8.2 *$\mathcal{CP}_{v_0}^{v_{sink}}$ is the set of dominant paths with constant weight, i.e. if $p \in \mathcal{CP}_{v_0}^{v_{sink}}$ then $u_i^p = 0$ for $0 \leq i \leq dim(X^{perf}) - 1$.*

All paths p with constant weight, i.e., $p \in \mathcal{CP}_{v_0}^{v_{sink}}$, are also subtracted from $\mathcal{P}_{v_0}^{v_{sink}}$. The final set is named $\mathcal{DP}_{v_0}^{v_{sink}}$. The weight of the paths in $\mathcal{DP}_{v_0}^{v_{sink}}$ can be expressed by the matrix equation below.

$$\mathcal{W}_p = \mathcal{A}X^{perf} + \mathcal{H}_P \tag{8.3}$$

Where $\mathcal{W}_p = (w(p_0), \ldots, w(p_l))^T$, $\mathcal{H}_p = (h(p_0), \ldots, h(p_l))^T$, $l = |\mathcal{DP}_{v_0}^{v_{sink}}|$, and

$$\mathcal{A} = \{a_{i\,j} = u_j^{p_i} \mid 0 \leq i \leq l \text{ and } 0 \leq j \leq dim(X^{perf} - 1)\} \quad \mathcal{Y}_p = \mathcal{A}X^{perf} \tag{8.4}$$

The random vector $\mathcal{Y}_p = \mathcal{W}_p - \mathcal{H}_P$ will therefore have a *joint probability density function* given by the following expression, when all the non-deterministic performance parameters are modeled by Gaussian variables:

$$f_{\mathcal{Y}_p}(\mathcal{Y}_p) = \frac{1}{(2\pi)^{n/2} det \mathcal{Q}^{\frac{1}{2}}} exp[-\frac{1}{2}(\mathcal{Y}_p - E(\mathcal{Y}_p))^T \mathcal{Q}^{-1}(\mathcal{Y}_p - E(\mathcal{Y}_p))] \tag{8.5}$$

where $E(V)$ is the expected (average) value of vector V, $\mathcal{Q} = \mathcal{A}\mathcal{K}^{perf}\mathcal{A}^T$, and $\mathcal{K}^{perf}$ $= diag(\sigma^2(x_1^{perf}), \ldots, \sigma^2(x_{dim(X^{perf})}^{perf}))$ if $x_1^{perf}, \ldots, x_{dim(X^{perf})}^{perf}$ are independent. Both $\mathcal{K}^{perf}$ and $\mathcal{Q}$ are *positive definite*.

The *joint cumulative probability function* $F_{\mathcal{Y}_p}(\mathcal{Y}_p \leq \mathcal{T}(v))$ of the random vector $\mathcal{Y}_p$ is useful in determining the *cumulative probability function* $F_{T_{SYS}}(T_{SYS} \leq T_z)$ of the *latency* T_{SYS} for a maximum allowed latency T_z, as shown below.

$$T_{SYS} = max(h_{max}^p, T_{max}) \tag{8.6}$$

where

$$h_{max}^p = max_{p_i \in \mathcal{DP}_{v_0}^{v_{sink}} \cup \mathcal{CP}_{v_0}^{v_{sink}}}(h(p_i)) \tag{8.7}$$

and

$$T_{max} = max(y_0 + h(p_0), \ldots, y_l + h(p_l)) \tag{8.8}$$

Let the vectorial function $\mathcal{T}$ be defined as $\mathcal{T}(v) = (v - h(p_0), \ldots, v - h(p_l))^T$, where v is a real number.

$$F_{T_{SYS}}(T_{SYS} < T_z) = \begin{cases} 0 & \text{if } T_z < h^p_{max} \\ F_{Y_p}(\mathcal{Y}_p \leq \mathcal{T}(T_z)) & \text{otherwise} \end{cases} \tag{8.9}$$

where

$$F_{Y_p}(\mathcal{Y}_p \leq \mathcal{T}(T_z)) = F_{Y_p}(y_0 \leq T_z - h(p_0), \ldots, y_l \leq T_z - h(p_l)) \tag{8.10}$$

This result can be used in evaluating the *feasibility probabilities* of a design.

Definition 8.3 *Given the maximum allowed values for cost ($Cost^{max}$) and system latency (T^{max}_{SYS}) for an acceptable design, the* cost *($Fes(Cost)$) and* system latency *($Fes(T_{SYS})$) and feasibility probabilities for a design x are respectively defined as*

$$Fes(Cost(x)) = \mathcal{P}(Cost(x) \leq Cost^{max}) \tag{8.11}$$

$$Fes(T_{SYS}(x)) = \mathcal{P}(T_{SYS}(x) \leq T^{max}_{SYS}) \tag{8.12}$$

8.2.2 Pipelined Case

The *pipelined* case is very similar in mathematical formalism to the *non-pipelined case*. However, the point of interest is in the cycles of the *constraint graph* $G = (V, E)$. Let the set of cycles of the graphs be $\mathcal{C}(G)$. Let $c \in \mathcal{C}(G)$, where $w(c)$ is the sum of the edge weights of the cycle.

$$w(c) = \sum_{i=0}^{dim(X^{perf})-1} u^c_i x^{perf}_i + h(c, T_I) \tag{8.13}$$

where $h(c, T_I)$ is the sum of deterministic weights in the cycle and $u^c_i = 0\, or\, 1$. $h(c, T_I)$ is also a linear function of the *initiation interval* T_I.
Let

$$Y_C = W_C - H_C(T_I) \tag{8.14}$$

Therefore, the following expression can be derived for the case when the non-deterministic performance parameters are modeled by Gaussian random variables.

$$f_{Y_C}(\mathcal{Y}_C) = \frac{1}{(2\pi)^{n/2} det \mathcal{Q}^{\frac{1}{2}}} exp[-\frac{1}{2}(Y_C - E(Y_C))^T \mathcal{Q}^{-1}(Y_C - E(Y_C))] \tag{8.15}$$

$$Y = \mathcal{A}X^{perf} \tag{8.16}$$

$$\mathcal{Q} = \mathcal{A}\mathcal{K}^{perf}\mathcal{A}^T \tag{8.17}$$

where $\mathcal{W}_C = (w(c_0), \ldots, w(c_l))^T$, $\mathcal{H}_C(T_I) = (h(c_0, T_I), \ldots, h(c_l, T_I))^T$, $l = |C(G)|$.

$$\mathcal{A} = \{a_{i\,j} = u_j^{c_i} \quad | \quad 0 \leq i \leq l \quad and \quad 0 \leq j \leq dim(X^{perf} - 1)\} \tag{8.18}$$

The *feasibility* $Fes(T_I(x))$ of the system for a given T_I^{min} can be evaluated as the probability that all cycles in G will not have negative weights.

$$Fes(T_I(x)) = F(Y_C \geq H_C(T_I^{min})) \tag{8.19}$$

8.2.3 Difficulties of Using an Exact Formulation

The calculation of the values of the feasibility probabilities cannot be made in an analytical way for some important types of random variables, e.g., ones with a Gaussian (normal) distribution. I.e., even for the unidimensional case, the c.d.f. of a Gaussian random variable involves an integration that is not analytic [124]. [8.3] When Gaussian distributions are present, the evaluation of $Fes(T_{SYS})$, for example, requires the use of computationally expensive numerical methods to approximate a multidimensional integration over the space defined by the random variables x_k^{perf}, $0 \leq k \leq dim(X^{perf}) - 1$. Such methods become prohibitive when used along with a genetic algorithm framework. There is need for more computationally efficient ways to estimate the feasibility of a solution, as it is discussed in next section.

[8.3] An analytic integration is one that can be expressed as a sum/product of analytic functions, where an analytic function is one that be calculated without need of differentiation or integration of other functions

8.3 A Stochastic Simulation Based Approach

MEGA avoids the use of numerical integration techniques for finding $Fes(T_{SYS})$ and $Fes(T_I)$ by using Monte Carlo simulation methods. This approach allows MEGA to keep most of the data-structures and algorithms developed for the deterministic paradigms. This Section outlines how Monte Carlo simulation is integrated in MEGA to support design in the presence of uncertainty.

8.3.1 Definitions

8.3.1.1 Generator of a Random Variable

Most present day workstations have software that provides *random number generators* simulating random variables with a uniform probability density function. In the same way, the simulation of other types of distributions, e.g. Gaussian, can be done by means of generalized *generator* functions, built on top of the basic uniform distribution generator.

Definition 8.4 *Given a random variable x with a uniform probability density distribution in the interval $[0,1]$, and a general random variable y with domain in $\mathcal{R}$, the set of real numbers. A function $g: [0,1] \longrightarrow \mathcal{R}$ is a* generator *of y if $g(x)$ corresponds to a random variable with the same probability density distribution (p.d.f.) of y.*

As a consequence, the cumulative probability distribution $\mathcal{F}_y(z)$ of a random variable y can be used to define a *generator* of y, i.e. $\mathcal{F}_y(z) = P(y \leq z)$, making F_y^{-1} a generator of y.

$$g_y(x) = F_y^{-1} \tag{8.20}$$

For a given random variable y there many other ways to define its generator or possible approximations. Ripley [110] discusses many computationally inexpensive approximate methods to evaluate g_y for a variety of commonly used distributions, as well as for random variables with non-zero correlation.

8.3.1.2 Multidimensional Sampling

Given the set $\mathcal{S}_{rv}$ of cost and performance parameters modeled as random variables $y_1, \ldots, y_m$, where m is a dimension of the *sampling space*, i.e., $m = |\mathcal{S}_{rv}|$, we can define a multidimensional sample.

Property 8.1 *Given the random vector* $\mathbf{U} = (u_1, \ldots, u_m)$, *where* u_i *is independent of* u_j, $i \neq j$, $1 \leq i, j \leq m$, *and* u_i *is a sample of a random variable with uniform probability density distribution in the interval* $[0, 1]$: *if the variables* $y_1, \ldots, y_m$ *are independent,* $\mathbf{U}$ *corresponds to a sample* $\mathbf{Y}$ *in the multidimensional domain defined by* $\mathcal{S}_{rv}$:

$$\mathrm{Y} = (g_{y_1}(u_1), \ldots, g_{y_m}(u_m)) \tag{8.21}$$

where g_{y_i} *is a generator for random variable* y_i, $1 \leq i \leq m$.

A similar approach can be applied to the case where the random variables are not independent.

8.3.1.3 Probabilistic Fitness Values in MEGA

MEGA adopts, as a fitness value function $f(x)$ of a design x, a linear combination of the average values ($\mu(\bullet)$), variations ($\sigma^2(\bullet)$) and *feasibility probabilities* ($Fes(\bullet)$) for cost, latency, and initiation interval (*macro-pipelined case* only).
Therefore the overall *fitness value function* $f(x)$ of a design x is defined in MEGA as

$$f(x) = w^f_{Cost} * Fes(Cost) + w^f_{T_{SYS}} * Fes(T_{SYS}) + w^f_{T_I} * Fes(T_I) + w^{\mu}_{Cost} * \mu(Cost) +$$
$$w^{\mu}_{T_{SYS}} * \mu(T_{SYS}) + w^{\mu}_{T_I} * \mu(T_I) + w^{\sigma}_{Cost} * \sigma(Cost)$$
$$+ w^{\sigma}_{T_{SYS}} * \sigma(T_{SYS}) + w^{\sigma}_{T_I} * \sigma(T_I) \tag{8.22}$$

where the most straightforward way to evaluate the *feasibility probabilities* is to generate N multidimensional samples (trials) of the cost and performance parameters modeled as random variables, and estimate the *feasibility probabilities* by the percentage of designs that meet the cost and performance constraints.

8.3.2 A Basic Template for Monte Carlo Simulation in MEGA

The following algorithm outlines how Monte Carlo simulation was incorporated in MEGA.

Algorithm 8.1 *Probabilistic MEGA*

*/**A Monte Carlo based approach for design in presence of uncertainty.**/*
Read input data including the task flow graph $G = (V, E)$.
Detect parallelism among tasks (data-transfers) using **transitive closure.**
Generate initial population $\mathcal{P}(0)$.
$t \longleftarrow 0$
repeat
  Perform **mutation** *in a few solutions of* $\mathcal{P}(t)$ *with probability* p_m
  Perform **crossover** *in a few solution pairs of* $\mathcal{P}(t)$ *with probability* p_c
  For $i = 1$ *to* N */* Stochastic Simulation */*
    Generate a multidimensional sample $\mathbf{Y} = (g_{y_1}(u_1), \ldots, g_{y_m}(u_m))$ *of the performance and cost parameters in* $\mathcal{S}_{rv}$.
    Find **detailed timing** *by using the Bellman-Ford algorithm.*
  end */*For i = 1 to N */*
  For *each solution (design)* x *in* $\mathcal{P}(t)$.
    From the N *samples of* x, *estimate statistics* $Fes(Cost)$, $Fes(T_{SYS})$, μ_{Cost}, $\mu(T_{SYS})$, $\sigma^2(Cost)$ *and* $\sigma^2(T_{SYS})$.
  end
  Evaluate fitness *of solutions in* $\mathcal{P}(t)$
  Scale and normalize *fitness values*
  Generate $\mathcal{P}(t+1)$ *from* $\mathcal{P}(t)$ *by a predefined* **selection scheme** *[89]*
  $t \longleftarrow t + 1$
until *near-optimal or optimal design (solution) is found*

8.3.3 Applying Variance Reduction Techniques

Given a stochastic simulation involving N trials and a statistic Υ measured from the result of these trials, e.g. a feasibility probability. It is expected that the variance ($\sigma^2(\Upsilon)$) of Υ [110] for N probabilistic independent trials be a function of N

$$\sigma^2(\Upsilon) = \frac{C}{N^{-\kappa}} \tag{8.23}$$

Where $\kappa \geq 1.0$ and C is a constant.

The *degree of certainty* of a simulation increases if the value $\sigma^2(\mathcal{Y})$ decreases. There are two possible approaches to accomplish this: either increase N or design a simulation method that has a high value for κ. The second approach is called *variance reduction* [110]. The use of variance reduction techniques might allow computationally inexpensive estimation of the different statistics of a design x , such as *feasibility probabilities*, and improve the quality of the results generated by a stochastic simulation-based fitness evaluation method, such as one used in MEGA for the *probabilistic* paradigm. This section exemplifies the use of two variance reduction techniques in MEGA: *stratified sampling* and *antithetic sampling*.

8.3.3.1 A Stratified Sampling Heuristic

The sampling method used in Algorithm 8.1 generates many samples that eventually do not violate the cost and performance constraints which leads to high imprecision of the values of the estimates of the *feasibility probabilities*, i.e. it is hard to estimate the feasibility if only a very small percentage of the samples do not meet the design constraints.

The rationale of *stratified or importance sampling* [110] is to have more multidimensional samples of the cost and performance parameters in the set $\mathcal{S}_{rv}$, violating at least one of the design constraints, allowing a decrease in the number of trials N needed to evaluate the *feasibility probabilities* with a high level of *certainty*, i.e. small variance. The following heuristic approach applies stratified sampling in the framework of MEGA. In order to maximize the chances that a solution will violate a cost and/or performance in T_{SYS}), the random vector $\mathbf{U}$ is redefined as containing elements that are random with a non-decreasing p.d.f. instead of a uniform p.d.f.

Definition 8.5 $\mathbf{U}' = (u_1', \ldots, u_m')$, *where* u_i' *is independent of* u_j', $i \neq j$, $1 \leq i, j \leq m$, *and* u_i' *is a sample of a random variable with a non-decreasing probability density distribution* f_i *in the interval* $[0, 1]$, *where* $f_i(x_a) \leq f_i(x_b)$ *if* $x_a \leq x_b$ *and* $0 \leq x_a, x_b \leq 1$.

A similar development can be made for the case of dependent random variables:

Property 8.2 *The probability of occurrence* $p(\mathbf{U}')$ *of* $\mathbf{U}'$ *is given by the following expression for the case of independent random variables:*

$$p(\mathbf{U}') = \prod_i f_i(u_i') \tag{8.24}$$

As a corresponding performance/cost multidimensional sample is given by $\mathbf{Y} = (g_{y_1}(u_1'), \ldots, g_{y_m}(u_m'))$, if $y_1, \ldots, y_m$ are independent, the use of non-decreasing probability density functions f_i will favor samples with high processor costs, high communication delays and high computational times, which increases the probability of occurrence of design constraint violations.

Let there be a function $\Psi(\bullet)$, used in the evaluation of one of the possible statistics used in MEGA. $\Psi(\bullet)$ has one of the following forms:

i. Feasibility probability

$$\Psi(x, x_{max}) = \begin{cases} 0 & \text{if } x \geq x_{max} \\ 1 & \text{otherwise} \end{cases} \tag{8.25}$$

ii. Average

$$\Psi(x, 0) = x \tag{8.26}$$

iii. Variance

$$\Psi(x, \mu(x)) = (x - \mu(x))^2 \tag{8.27}$$

where x is either the cost ($Cost$) or latency (T_{SYS}) of a design.

The estimation of the stratified average value $\mu_{str}(\Psi_{str}(\bullet))$ of the statistics corresponding to $\Psi(\bullet)$ is given by

$$\mu_{str}(\Psi_{str}(x, z)) = \frac{\sum_{k=1}^{N} \frac{\Psi(x(U_k), z)}{p(U_k)}}{N * \sum_{k=1}^{N} \frac{1}{p(U_k)}} \tag{8.28}$$

where N is the number of trials, $x(U_k)$ is a sample with a probability of occurrence $p(U_k)$, and z is a constant used in the evaluation of the statistics.

Intuitively, the fact that some multidimensional samples are expected to be less frequent is compensated for by using a harmonic average in terms of $p(\mathbf{U_k})$, probability of occurrence of $\mathbf{U_k}$. If all probability density functions f_i are uniform in the interval $[0, 1]$, Equation 8.28 is reduced to the general Monte Carlo case (non-stratified sampling).

8.3.3.2 Using Antithetic Sampling in MEGA

Given two random variables X_1 and X_2 with the same probability density distribution, and by consequence:

$$\mu = \mu(X_1) = \mu(X_2) \quad and \quad \sigma^2 = \sigma^2(X_1) = \sigma^2(X_2) \tag{8.29}$$

where $\sigma^2(X_i)$ is the variance $(var(X_i))$ of X_i, $i = 1, 2$.

Definition 8.6 *X_1 and X_2 are said to be* antithetic *[110] if they have a non-zero covariance and their correlation is negative, i.e. $corr(X_1, X_2) < 0$*

Property 8.3 *The average $Z_{1,2}$ of two antithetic variables X_1 and X_2 is a random variable with a variance $\sigma^2(Z_{1,2})$ smaller than σ^2, i.e.*

$$Z_{1,2} = \frac{X_1 + X_2}{2} \tag{8.30}$$

$$\sigma^2(Z_{1,2}) = \frac{\sigma^2 - covar(X_1, X_2)}{2} = \frac{\sigma^2}{2}(1 + corr(X_1, X_2)) \tag{8.31}$$

Property 8.4 *Given a generator function g_{y_i} and a random variable u with uniform probability distribution in the real interval $[0, 1]$, the variables $X_1 = g_{y_i}(u)$ and $X_2 = g_{y_i}(1 - u)$ are antithetic.*

8.3.3.2.1 Allowing Antithetic Sampling in MEGA

Antithetic variables can be incorporated in MEGA by changing the the inner loop of the genetic Algorithm 8.1.

Heuristic 8.1

For $i = 1$ *to* $N/2$ */*Stochastic Simulation*/*
Generate a multidimensional sample of the performance and cost parameters in $\mathcal{S}_{rv}$.
and its corresponding antithetic version
$\mathbf{Y}^* = (g_{y_1}(1 - u_1), \ldots, g_{y_m}(1 - u_m))$
Find **detailed timing** *using Bellman-Ford algorithm for* $\mathbf{Y}$ *and* $\mathbf{Y}^*$
Average results for detailed timing and cost.

end

8.4 Experimental Results

A set of benchmarks [99] (Figure 8.1) was adapted to evaluate the performance of MEGA for design in the presence of uncertainty. Tables 8.4, 8.4 and 8.4 provide detailed information about the tasks of the two task-flow graphs used as benchmarks as well as information regarding available processor and bus types. For the sake of simplicity, all probabilistic design parameters in Tables 8.4, 8.4 and 8.4 are assumed to have a Gaussian distribution. All data transfers are assumed to have a data volume equal to 1, with a negligible communication delay when *local*, where the overall delay of a non-local data transfer is given by

$$Delay_{remote}(DT, bus) = \tau_{bus} + \frac{Volume(DT)}{s_{bus}} \tag{8.32}$$

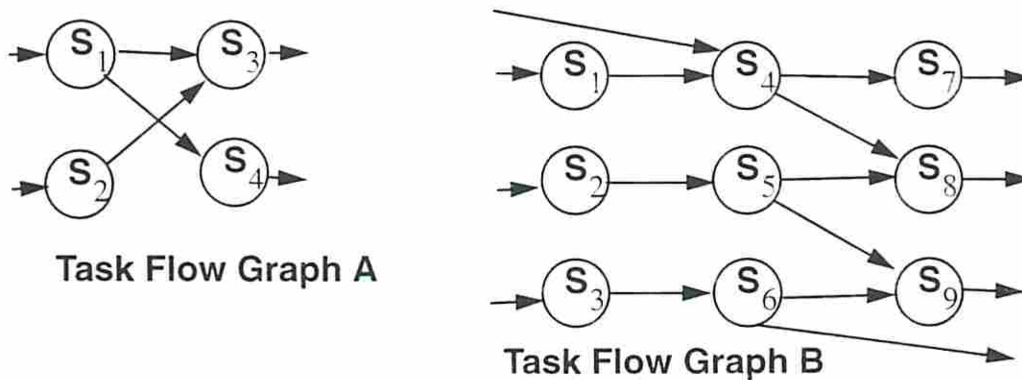


Figure 8.1: Task flow graphs

Tasks (S_i)	Computation Time S_1	S_2	S_3	S_4	Processor Type	Cost
average (μ)	1.0	1.0	--	3.0	P_1	4.0
standard deviation (σ)	0.1	0.1		0.3		0.4
average (μ)	3.0	1.0	2.0	1.0	P_2	5.0
standard deviation (σ)	0.3	0.1	0.2	0.1		0.5
average (μ)	--	3.0	1.0	--	P_3	2.0
standard deviation (σ)		0.3	0.1			0.2

Table 8.1: Processor types - cost and performance information for TFG A

Tasks (S_i)	Computation Time									Processor	
	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	Type	Cost
average (μ)	1.0	1.0	0.5	1.0	1.0	1.0	2.0	--	0.7	P_1	4.0
standard deviation (σ)	0.1	0.1	0.05	0.0	0.0	0.0	0.5		0.07		0.4
average (μ)	1.5	0.5	0.5	3.0	1.0	2.0	0.7	1.3	0.7	P_2	5.0
standard deviation (σ)	0.15	0.05	0.05	0.0	0.0	0.0	0.15	0.33	0.03		1.0
average (μ)	0.5	0.5	1.0	--	3.0	1.0	2.7	0.7	2.0	P_3	2.0
standard deviation (σ)	0.05	0.05	0.1		0.0	0.0	0.68	0.18	1.0		0.6

Table 8.2: Processor types - cost and performance information for TFG B

Bus Types		Design Parameters		
		Cost	Speed	Latency
B_1	average (μ)	10.0	1.0	0.01
	standard deviation (σ)	1.0	0.1	0.001
B_2	average (μ)	20.0	3.0	0.01
	standard deviation (σ)	2.0	0.1	0.001

Table 8.3: Buses types - cost and performance information

In all experiments, an *elitist* selection scheme [89] was used with mutation (p_m) and crossover (p_c) probabilities equal to 0.1, a fixed size population in the range of 100 to 150 chromossomes, 70 trials per stochastic simulation and absence of variance reduction heuristics.

Table 8.4 gives an idea of the design space for task-flow graphs A and B (Figure 8.1. Different designs (trade-off points) were found by changing the fitness weights in Equation 8.22. As expected, designs with a higher *feasibility probability* for the latency constraint ($Fes(T_{SYS})$) demand more hardware resources. On the other hand, designs with a higher *feasibility probability* for the cost constraint ($Fes(Cost)$) have longer latencies (T_{SYS}) and lower values for $Fes(T_{SYS})$. The typical shape of the probabilistic distribution of the system latency is shown in Figure 8.2 for Design VII in Table 8.4, whose Gannt chart is presented in Figure 8.4.

The run-time of MEGA was around a few minutes in a SUN-SPARC4 to generate all points in Table 8.4 compared to hours using the SOS [99] approach, that is restricted to optimal design of ASHM without the presence of uncertainty in design parameters, i.e. the deterministic case.

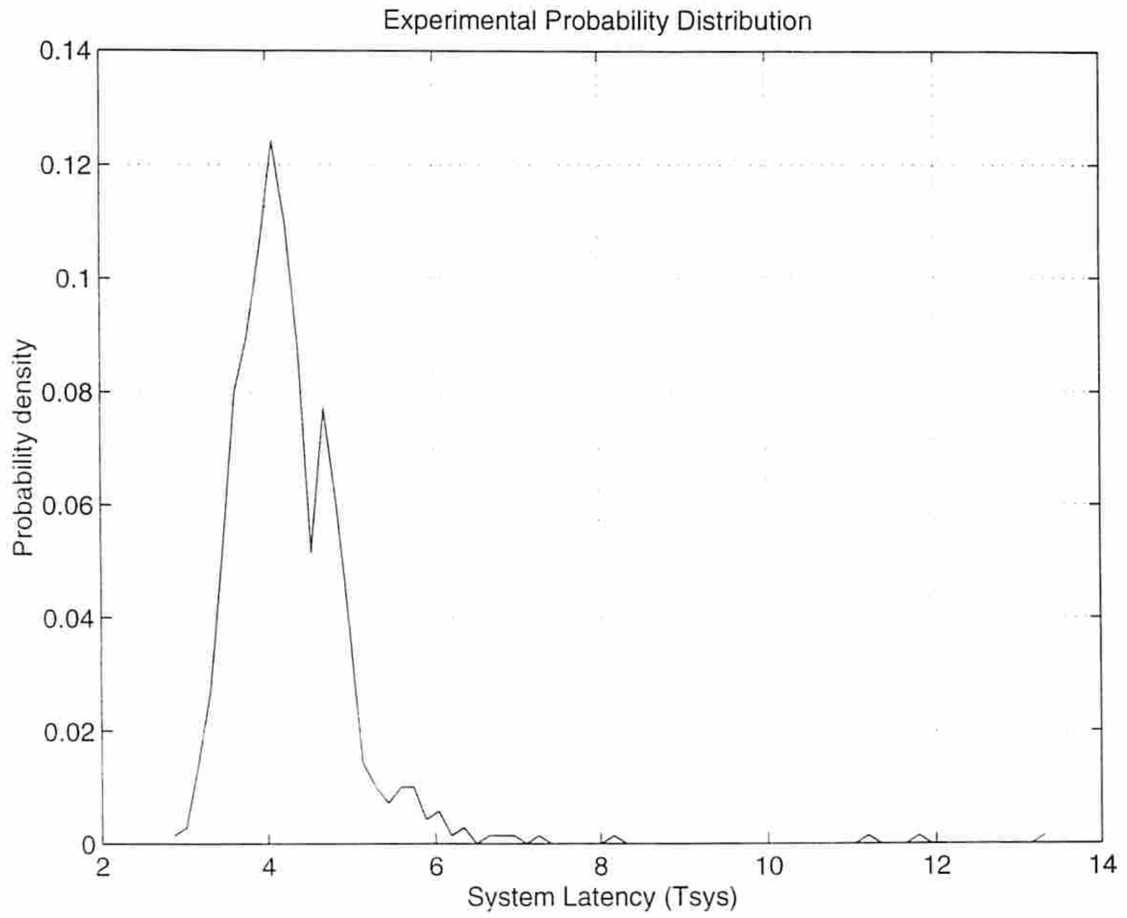


Figure 8.2: System latency (T_{SYS}) of Design VII

#	TFG	Design Constraints		Number of Processors			Number of Buses		Feasibility Probabilities			Trade-off Points		Run Time
		C_{ost}^{max}	T_{SYS}^{max}	P_1	P_2	P_3	B_1	B_2	*Cost*	T_{SYS}		*Cost*	T_{SYS}	Sun4
I	A	25.0	7.0	2	1	1	1	1	0.03	1.0	μ	43.0	4.28	16s
											σ	2.37	0.65	
II	A	25.0	7.0	1	0	1	1	0	1.0	1.0	μ	16.0	4.40	22s
											σ	1.41	0.62	
III	A	25.0	7.0	0	1	0	0	0	1.0	0.47	μ	5.0	7.02	20s
											σ	0.71	0.91	
IV	A	25.0	14.0	0	1	0	0	0	1.0	1.0	μ	5.0	7.02	20s
											σ	0.71	0.91	
V	A	25.0	14.0	2	0	1	1	0	1.0	0.98	μ	16.0	4.62	22s
											σ	1.41	1.40	
VI	A	50.0	14.0	0	1	0	0	0	1.0	1.0	μ	5.0	6.78	21s
											σ	0.71	0.85	
VII	B	25.0	7.0	1	1	2	1	1	0.03	1.0	μ	43.0	4.28	41s
											σ	2.37	0.65	
VIII	B	25.0	7.0	1	1	1	1	1	0.24	0.97	μ	41.0	4.50	42s
											σ	2.24	0.93	
IX	B	25.0	14.0	0	1	0	0	0	1.0	1.0	μ	5.0	11.27	42s
											σ	1.0	0.80	
X	B	15.0	14.0	1	1	1	1	1	0.24	1.0	μ	41.0	4.5	37s
											σ	2.23	1.05	

Table 8.4: Selected Designs - Probabilistic Trade offs for Latency (T_{SYS}) and Cost

The effectiveness of variance reduction techniques in speeding up the convergence of the proposed genetic algorithm is exemplified in Figure 8.3. *Antithetic sampling* allows a faster convergence rate in the beginning but is not competive compared to the case where no variance reduction techniques are applied at all. On the other hand, *stratified sampling* allows a faster rate of convergence coupled with increased chances to find a solution closer to the optimal, i.e. a design with a higher fitness value, being the best approach of all three.

8.5 Summary

This chapter introduces a formal and exact method to evaluate the feasibility of a design subject to uncertainty in the performance/cost parameters. A practical method for estimation of the performance/cost probabilistic distribution of a design,

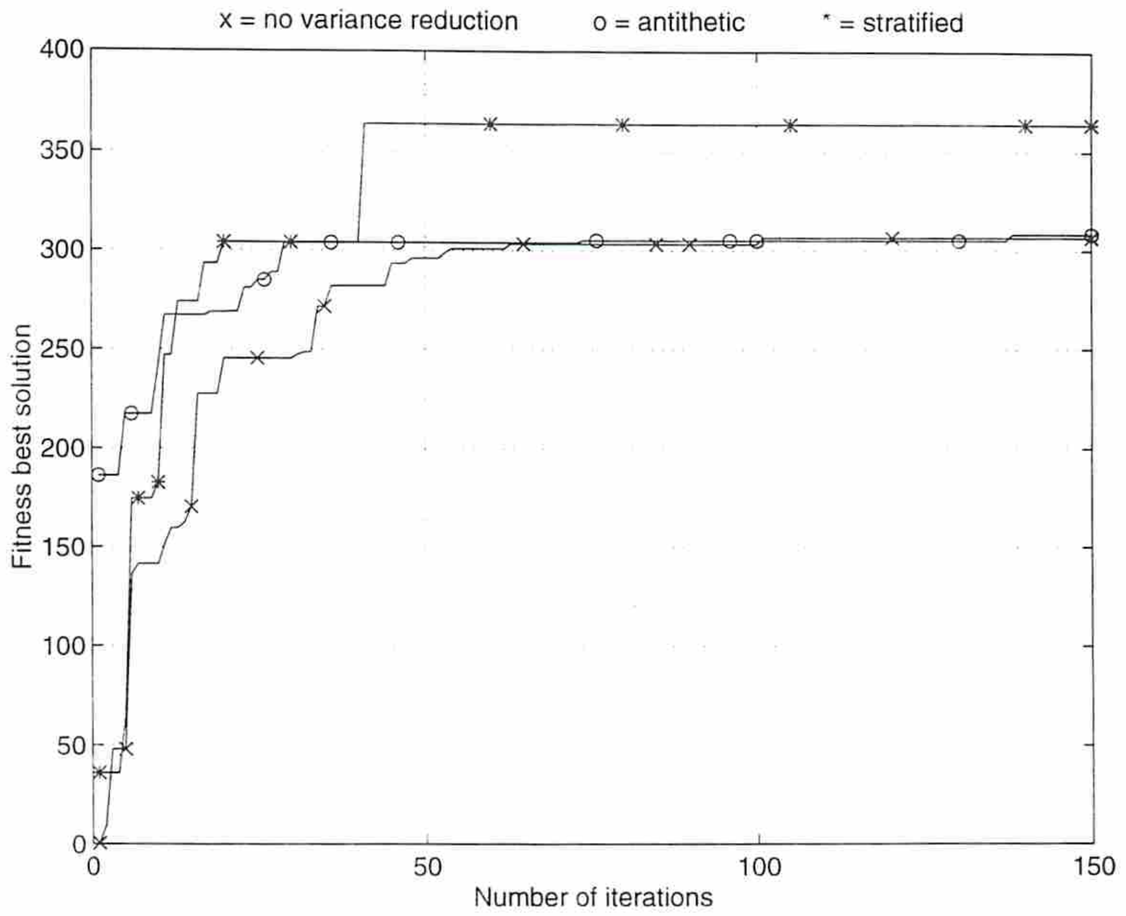


Figure 8.3: Effect of variance reduction techniques (Design VII)

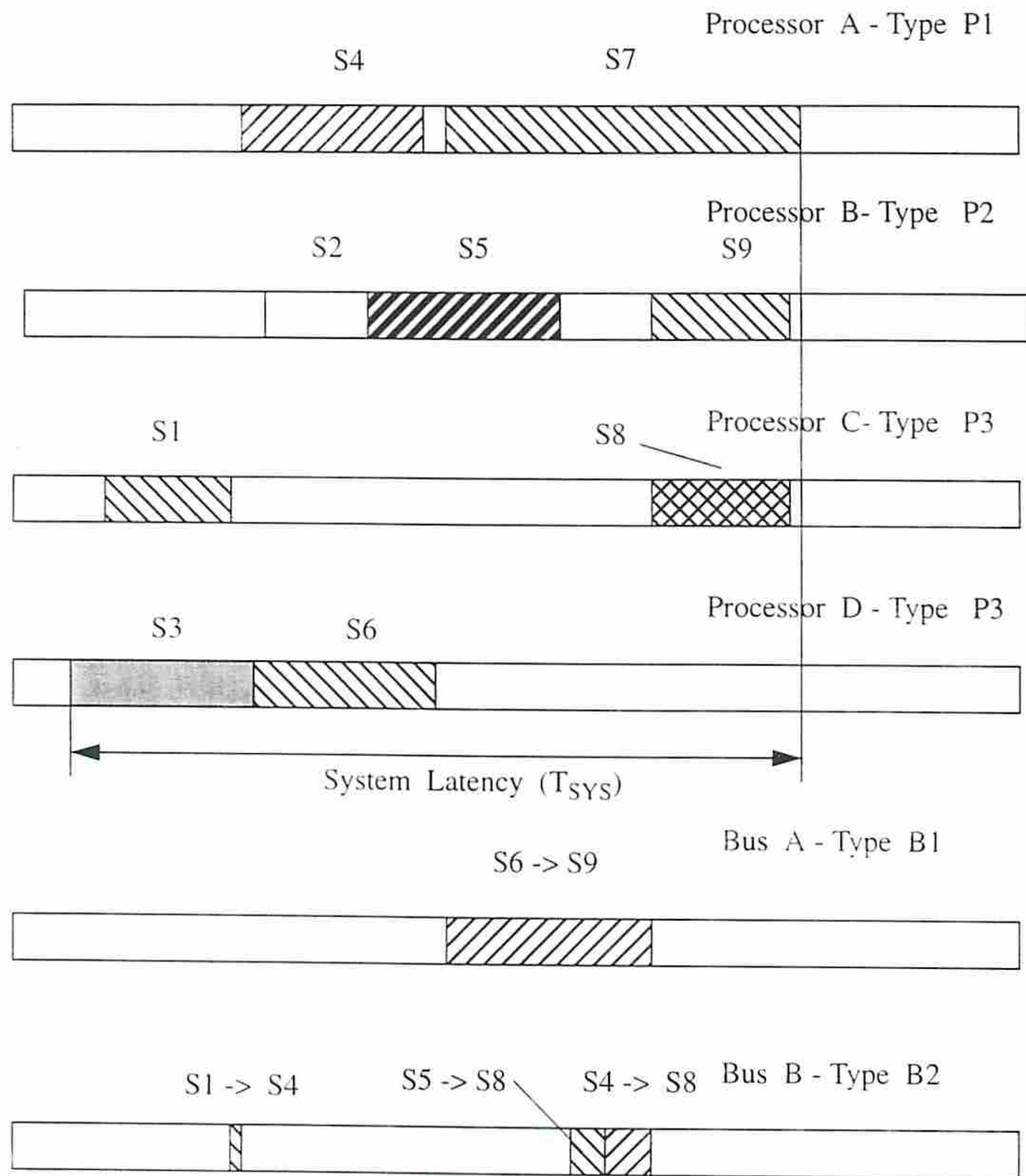


Figure 8.4: Gannt chart for Design VII

corresponding to a particular value assignment of the Boolean variables, is proposed by applying stochastic simulation. The *fitness value* of a solution is made a function of the feasibility values and average and variance values for the *system latency*, and *cost*. Variance reduction methods that are suitable for use with the MEGA tool set are outlined. Comparative experiments among different approaches discussed here are presented showing the soundness of the proposed methods used in handling system-level design with uncertainty.

Chapter 9

Fast prediction in system-level design

9.1 Motivation

Performance bounds play an important role for synthesis of application-specific heterogeneous multiprocessors. They allow evaluation of the expected quality of the designs found by a synthesis algorithm or synthesis heuristic. They can be used to predict the performance of the optimal design.

9.2 Fundamentals

Definition 9.1 *Given a design M by an algorithm or heuristic A, the ratio R_M is defined as the ratio of the performance of the optimal design M_{OPT} to the performance of the design M. The upper bound $UB(R_M)$ of the ratio R_M gives an idea of how far the design M is from the optimal design in terms of performance in the worst case.*

Non-pipelined case

$$Perf(M) = \frac{1}{\omega_M} \tag{9.1}$$

$$R_M = \frac{\omega_{OPT}}{\omega_M} \tag{9.2}$$

Pipelined case

$$Perf(M) = \frac{1}{T_I} \tag{9.3}$$

$$R_M = \frac{T_I}{T_{I_{OPT}}} \tag{9.4}$$

Where $Perf(M)$ is a measure of the *performance* of a design M.

Given the performance of the design $M, UB(R_M)$ allows the evaluation of an *upper bound* for the performance of the optimal design. The design M and $UB(R_M)$ can be used as a *predictor* of the performance of the optimal design for the designer or for the software system helping the designer to synthesize a feasible optimal or near-optimal design. The use of a *performance predictor* is highly valuable whenever the computational cost to find the optimal design is prohibitive, which is the case in system-level synthesis, where most important problems, e.g. scheduling of tasks in a multiprocessor, are at least *NP-complete*. The performance bound $UB(R_M)$ can be dependent or not on the algorithm or heuristic that is being used. The key point is to try to design algorithms or heuristics of low computational cost having a $UB(R_M)$ close to 1. Below some important findings on this research subject are highlighted for the case of non-preemptive task execution, since the target of this dissertation is the synthesis of low-cost application-specific implementations.

There are many possible ways to define the *performance* of an application-specific multiprocessor M. The most basic one is to define performance in terms of the *processing speed.* Two possible execution modes of the task flow graph: *non-periodic and macro-pipelined (periodic)*, are assumed.

The processing speed is usually defined with the following terms for each case. The speed is the inverse of the *system delay* ω for the non-periodic case. The *system delay* is defined as the minimal time to execute the task-flow graph in the multiprocessor being evaluated. The pipelined case performance depends on the *initiation interval* T_I, which is basically the minimal allowed period for the start of a new iteration of the task-flow graph for the multiprocessor being considered. The speed is defined as the inverse of the initiation interval for the pipelined case. Known bounds for the non-periodic case are discussed below. This survey was unable to find published performance bounds for the macro-pipelined case, which indicates a possible new frontier of research.

9.3 A survey in performance bounds theory

Let the task flow graph be defined as $TFG = (V, <)$, where $V = \{S_1, S_2, ...\}$ is the set of tasks and $S_i < S_j$ means that there is a data transfer from task S_i to task S_j. The binary relation $<$ is *empty* when the tasks are independent from each other.

The problems of both optimal allocation and optimal scheduling of the set of tasks V in a multiprocessor for maximization of the performance, as defined above, are known to be at least NP-complete even when the relation $<$ is empty, i.e., no data dependencies are present.

A *homogeneous* multiprocessor is one where all processors are identical. A *heterogeneous* multiprocessor in one where the processors can differ in *time of execution* and *ability to execute* each one of the tasks in V. A *uniform* multiprocessor is one where the *ratio of execution times* of a task S_i in two processors P_r and P_s is equal to the ratio of *relative speeds* s_r and s_s. A *non-uniform* multiprocessor is one where this property does not need to be true.

Uniform case

$$\frac{ExecutionTime(S_i, P_r)}{ExecutionTime(S_i, P_s)} = \frac{s_r}{s_s} \; : \forall S_i \in V \tag{9.5}$$

Note that a *homogeneous* multiprocessor is by definition *uniform*, as the *ratio of execution-times* is 1 for all tasks and processor pairs. A *heterogeneous* multiprocessor can be *uniform* or not. The most general case is a *non-uniform* heterogeneous multiprocessor. The algorithm or heuristic used to synthesize the application-specific multiprocessor is said to *count communication costs* if the non-local data transfers are assumed to not be instantaneous in the final design, i.e., transfers between tasks allocated to different processors have non-zero delays. It should be noted that the works covered in this survey assume that a communication link allows only one data transfer at any instant of time, i.e., there is no concurrence of data transfers allocated to the same communication link.

The set $\Re = \{Rs_1, Rs_2, ...\}$ is the set of resources to be shared by the processors in the multiprocessor. A resource in $\Re$ can only be used by one processor at any instant of time (*mutual-exclusive usage constraint*). A shared-bus multiprocessor can be modeled as one with $\{bus\} \subset \Re$, i.e., the *bus* is one of the resources in $\Re$. The integer s is equal to $|\Re|$, the number of resources in $\Re$ (which is assumed to be empty in the cases below unless otherwise mentioned). The set $P = \{P_1, P_2, ..., P_n\}$ is the set of processors in the multiprocessor and n is the number of processors. The integer n is always less than or equal to the number of tasks $|V|$: $n = |V|$ allows the assignment of one task per processor.

Graham [40] proved the bounds below for the case of a *priority-list-based* algorithm for scheduling a task-flow graph on a homogeneous multiprocessor where there are no communication costs or resources constraints:

Homogeneous Case

$$\frac{\omega_M}{\omega_{OPT}} \leq 2 - \frac{1}{n} = UB(R_M) \tag{9.6}$$

$$\frac{\omega_M}{\omega_{OPT}} \leq \frac{4}{3} - \frac{1}{3n} = UB(R_M)\ ;\ when < empty \tag{9.7}$$

For the case where tasks are data independent ($<$ *is empty*) an allocation algorithm (MULTIFIT) using the *bin-packing* technique [19] will have $UB(R_M)$ probably less than 1.22 [42]. The MULTIFIT algorithm provides a tighter upper bound than pure list scheduling.

Garey [41] extended the above results for list scheduling algorithms with resource constraints:

Homogeneous Case

$$\frac{\omega_M}{\omega_{OPT}} \leq n = UB(R_M)\ ;\ for\ s = 1(one\ resource) \tag{9.8}$$

$$\frac{\omega_M}{\omega_{OPT}} \leq s + 1 = UB(R_M)\ ;\ for\ n = |V|\ and < empty \tag{9.9}$$

$$\frac{\omega_M}{\omega_{OPT}} \leq min(\tfrac{n+1}{2}, s + 2 - \tfrac{2s+1}{n}) = UB(R_M)\ ; \quad for\ n \geq 2\ and < empty \tag{9.10}$$

Hwang et al. [40] extended the results for the case where there are communication costs. Their analysis considered a polynomial-time *priority-driven* scheduling algorithm called *Earliest Task First (ETF)* [59] [74] . Their bound uses the value C defined as the sum of communication costs (delays) over some of chain of data transfers in the task flow graph. Hwang et al. provide an algorithm to find C.

Homogeneous Case with communication costs

$$\omega_M \leq (2 - \frac{1}{n})\omega_{OPT} + C \tag{9.11}$$

Tighter lower bounds on the execution time for the case of homogeneous processors and presence of communication costs were developed by Al-Mouhamed [90]. His work allows the rough evaluation of trade-offs between bounds on the *system*

delay and the number of processors and communication links. His work is based on a earlier result [31] which was the first to introduce this kind of trade-off analysis for the case of homogeneous multiprocessors without communication costs.

Results for the *heterogeneous uniform* multiprocessor with no communication costs case include performance bounds for the designs found by the MULTIFIT (MF) algorithm [35], LPT (longest processing time task first) list scheduling algorithm [36] and a general performance bound for list scheduling algorithms (LS) [61] for the typed-tasks case assuming k types of processors ($k \leq |V|$). In a typed-task scheduling problem, a task of a given type is only able to execute on a processor of the same type. Processors of the same type can differ in speed, but they are *uniform* among themselves. The bound for this more general case is given as the sum of the number of types k and the *maximum ratio between speeds of any processor of the same type* . These results are summarized below.

Heterogeneous Uniform Case

$$1.341 \leq UB(R_M(MF)) \leq 1.4 \,;\, for\ < empty\ [35] \tag{9.12}$$

$$1.52 \leq UB(R_M(LPT)) \leq 1.67 \,;\, for\ < empty\ [36] \tag{9.13}$$

$$\frac{\omega_M}{\omega_{OPT}} \leq k + max_i(\tfrac{s_p}{s_q}) = UB(R_M(MF))\ ; \quad Processors P_p\ and\ P_q\ are\ of\ type\ i\ [61] \tag{9.14}$$

Liu & Liu [78] and Jaffe [61] independently developed the same performance bound for the typed-tasks case assuming a heterogeneous multiprocessor with processors of the same type being identical (same speed).

Heterogeneous Case

$$\sum_i m_i = n\ \ ; m_i\ is\ the\ number\ of\ processors\ of\ type\ i \tag{9.15}$$

$$\frac{\omega_M}{\omega_{OPT}} \leq k + 1 - max_i(\frac{1}{m_i}) = UB(R_M(LS)) \tag{9.16}$$

The overview of published findings in performance bounds theory indicates the following new directions of research. There is need to develop performance bounds for multiprocessors where communication costs are not negligible. The approach of Hwang et al. [59] which evaluated the effect of the communication cost in the design

found by a particular list scheduling algorithm can give hints on how to achieve a similar result for the heterogeneous case.

The work of Al-Mouhamed [90], which considers communication costs, allows us to evaluate trade-offs on performance by giving tight performance bounds as a function of constraints on the number of processors and communication links in a homogeneous multiprocessor.

9.4 Generating a lower bound surface

The available bounds discussed in the survey are not very tight. However the development of better analytic performance bounds for heterogeneous multiprocessors is a difficult problem. Analytic bounds allowing study of trade-offs between performance and cost are even harder to achieve [90]. A more practical approach would be to generate a tight *lower-bound* surface composed of *non-inferior solutions*. This can be easily done by applying algorithms and heuristics discussed in the previous chapters for the automatic generation of a *trade-off* surface composed of different non-inferior solutions from all regions of the design space.

9.4.1 Non-inferior solutions

Given a solution (design) x, the set of *evaluation criteria* $\{c_1(x), \ldots, c_k(x)\}$ used to measure the overall quality or optimality of x can be used to define a evaluation vector $U(x) = (c_1(x), \ldots, c_k(x))$, where in the MILP models studied in Chapters 1 and 3, the objective function f_{obj} is generally given as

$$f_{obj} = \mathbf{WU}(\mathbf{x}) = \sum_{j=1}^{k} w_{c_j} * c_j(x) \tag{9.17}$$

Definition 9.2 *Given two vectors* $\mathrm{U_i} = (c_1^i(x), \ldots, c_k^i(x))$ *and* $\mathrm{U_{ii}} = (c_1^{ii}(x), \ldots, c_k^{ii}(x))$. $\mathrm{U_i} \geq \mathrm{U_{ii}}$ *if and only if* $c_1^i(x) \geq c_1^{ii}(x), \ldots, c_k^i(x) \geq c_k^{ii}(x))$.

Definition 9.3 *Given two vectors* $\mathrm{U_i} = (c_1^i(x), \ldots, c_k^i(x))$ *and* $\mathrm{U_{ii}} = (c_1^{ii}(x), \ldots, c_k^{ii}(x))$. $\mathrm{U_i}$ *is inferior to* $\mathrm{U_{ii}}$ *if and only if* $\mathrm{U_i} \leq \mathrm{U_{ii}}$ *and for at least one* $j \in \{1, \ldots, k\}$ $c_j^i(x) < c_j^{ii}(x)$.

Definition 9.4 *Given a set of solutions $\mathcal{S}$. $\mathcal{S}$ is a set of non-inferior solutions if and only if there does not exist a pair of solutions $x, y \in \mathcal{S}$ such that* $\mathbf{U}(\mathbf{x})$ *is inferior to* $\mathbf{U}(\mathbf{y})$.

Property 9.1 *A $\mathcal{S}$ set of non-inferior solutions defines a surface in a k-dimensional space of coordinates $(c_1(x), \ldots, c_k(x))$.*

Moreover $\mathcal{S}$ is both a lower-bound surface for different optimal points of design space, i.e. each one corresponding to a different weight vector $\mathbf{W}$ in the objective function, and a trade-off surface among different *evaluation criteria* in the k-dimensional space.

9.4.2 Non-periodic case

The evaluation vector of a solution x can be defined as $\mathbf{U} = (-T_{SYS}(x), -Cost(x))$. In the case of imprecise computation being allowed, $\mathbf{U} = (-T_{SYS}(x), -Cost(x), Q_{SYS}(x))$.

For the probabilistic paradigm, a possible evaluation vector would be $\mathbf{U} = (-\mu(T_{SYS}(x)), -\mu(Cost(x)), -\sigma^2(T_{SYS}(x)), -\sigma^2(Cost(x)))$, where $\mu(z)$ and $\sigma^2(z)$ are the expected value and variance of the random variable z.

9.4.2.1 Macro-pipelined case

The macro-pipelined case is similar to the non-periodic case. However the *initiation interval* is added to the *evaluation criteria*.

$\mathbf{U} = (-T_{SYS}(x), -Cost(x), -T_I(x))$ is a possible evaluation vector for the deterministic paradigm. $\mathbf{U} = (-T_{SYS}(x), -Cost(x), -T_I(x), Q_{SYS}(x))$ for the *imprecise computation* paradigm, and $\mathbf{U} = (-\mu(T_{SYS}(x)), -\mu(Cost(x)), -\mu(T_I(x)), -\sigma^2(T_{SYS}(x)), -\sigma^2(Cost(x)), -\sigma^2(T_I(x)))$ is a possible choice for the probabilistic paradigm.

9.4.3 Genetic generation of trade-off surfaces

The following algorithm can be used to generate progressively tighter lower-bound surfaces. Each iteration has an associated trade-off surface $\mathcal{S}(t)$ that is updated from

the new non-inferior solutions in population $\mathcal{P}(t)$. A key point of this algorithm is the need for a high *genetic diversity* in order to assure that different regions of the design space will be represented on the surface.

Definition 9.5 *The fitness value of a solution x in $\mathcal{P}(t)$ is redefined as the number of other solutions in $\mathcal{P}(t)$ that are inferior to x.*

Algorithm 9.1 *MEGA - a genetic algorithm for system-level design of application specific multiprocessors.*

Read input data including the task flow graph $G = (V, E)$
Transform multicasts in G to twin data-transfers.
Detect parallelism among tasks (data-transfers) using **transitive closure**
Generate initial population $\mathcal{P}(0)$
Generate $\mathcal{S}(0)$ *from non-inferior solutions of* $\mathcal{P}(0)$.
$t \longleftarrow 0$
repeat
    Perform **mutation** *in a few solutions of* $\mathcal{P}(t)$
    Perform **crossover** *in a few solution pairs of* $\mathcal{P}(t)$
    Find **detailed timing** *using Bellman-Ford based algorithms discussed in Section 6.13.*
    Evaluate fitness *of population* $\mathcal{P}(t)$
    $t \longleftarrow t + 1$
    Generate $\mathcal{S}(t)$ *from non-inferior solutions of* $\mathcal{P}(t-1) \cup \mathcal{S}(t-1)$.
    Generate $\mathcal{P}(t+1)$ *from* $\mathcal{P}(t)$ *by a* **selection scheme**
until $\mathcal{S}(t)$ *converges or* $t > t_{max}$

9.5 Experimental Results

A set of benchmarks [99] (Figure 9.1) was used to evaluate the performance of MEGA as a *cost-performance trade-off curve* generator. Tables 9.1 . 9.2 and 9.2 give performance information about the tasks of the two task-flow graphs as well as performance and cost numbers for the available processor and bus types. All data transfers are assumed to have a data volume equal to 1. with a negligible

communication delay when *local*, where the overall delay of a non-local data-transfer is given by

$$Delay_{remote}(DT, bus) = \tau_{bus} + \frac{Volume(DT)}{s_{bus}} \tag{9.18}$$

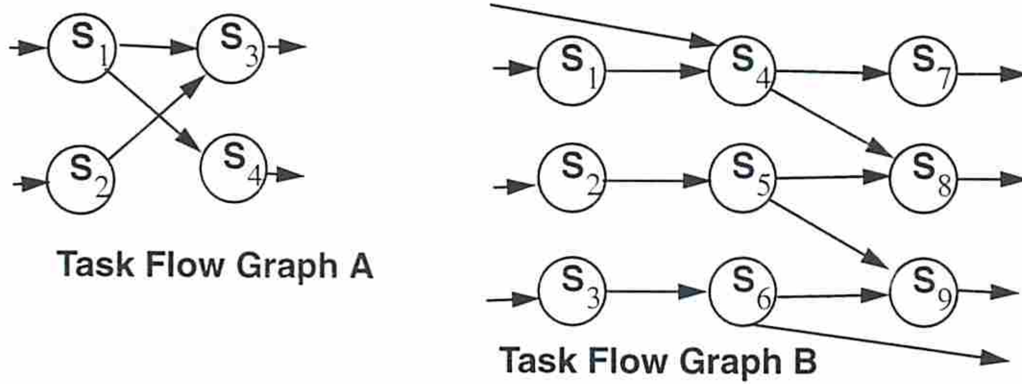


Figure 9.1: Task flow graphs

Processor types	Processor Cost	S_1	S_2	S_3	S_4
P_1	4	1	1	--	3
P_2	5	3	1	2	1
P_3	2	--	3	1	--

Table 9.1: Processor costs and task execution times - Task Flow Graph A

Processor types	Processor Cost	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
P_1	4	2	2	1	1	1	1	3	-	1
P_2	5	3	1	1	3	1	2	1	2	1
P_3	2	1	1	2	--	3	1	4	1	3

Table 9.2: Processor cost and task execution times - Task Flow Graph B

During the generation of the *trade-off curves*, the mutation (p_m) and crossover (p_c) probabilities were made equal to 0.15 and a fixed size population of 50 chromosome was assumed in order to better visualize the progressive convergence of the

Bus			
Type	Cost	Speed	Latency
B_1	10	1.0	0.0
B_2	20	3.0	0.0

Figure 9.2: Buses types - cost and performance information

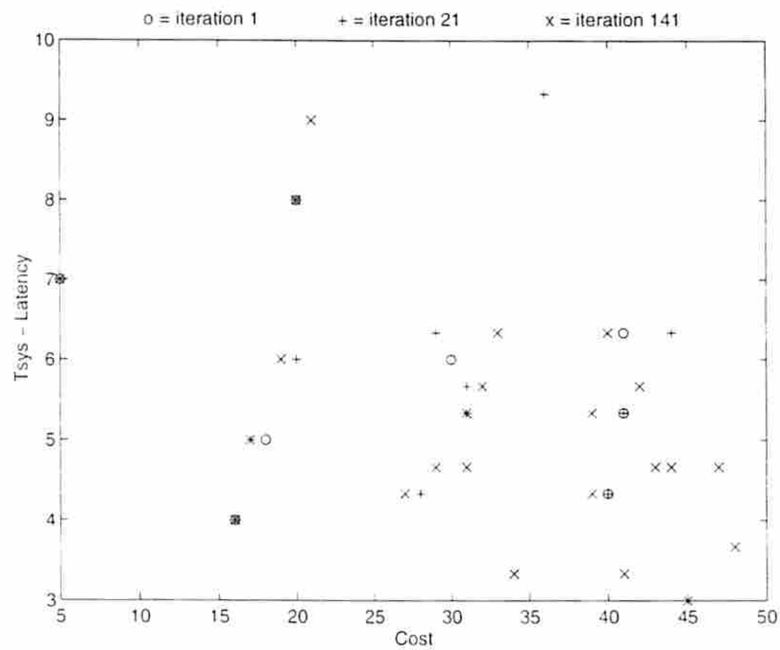


Figure 9.3: TFG A - Cost and Latency trade-off curve

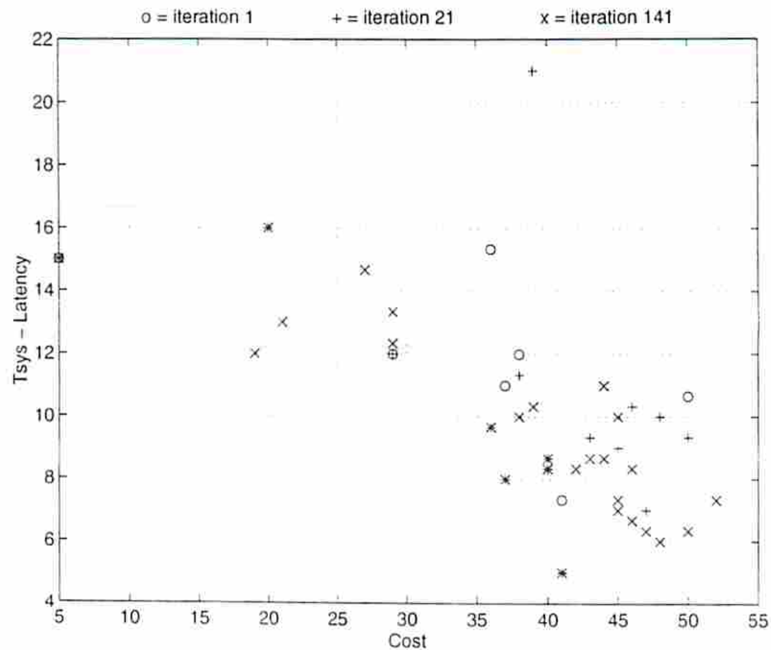


Figure 9.4: TFG B - Cost and Latency trade-off curve

trade-off curves, as shown in Figures 9.3 and 9.4. Faster results (with less iterations) can be achieved with a larger population and more agressive values for p_m and p_c. The run-time of MEGA was to 1.3 sec for *TFG A* and 4.4 sec for *TFG B* on a SUN-SPARC4 workstation in the present experiment.

9.6 Summary of the chapter

The previous work on analytic performance bounds for homogeneous and heterogeneous multiprocessors was revisited. A trade-off curve/surface generator based on the routines using in MEGA was proposed as an alternative method for fast system-level prediction.

Chapter 10

Conclusion and Future Research

10.1 Contribution

In this thesis, we have considered practical and theoretical issues of computer-aided *system-level* design of application-specific heterogeneous multiprocessors (ASHMs) with emphasis on non-preemptive scheduling. Our approach was based on two major techniques: exact optimization of mixed-integer linear programming (MILP) models and heuristic optimization using genetic algorithms.

This thesis extended previous MILP formulations [100] for representing different issues in system-level design of ASHMs for both the deterministic (*macro-pipelined* and *imprecise computation*) and non-deterministic cases (design in the presence of *uncertainty*). Key-points in MILP modeling applied to system-level design problems were discussed in Chapter 1.

The design of ASHMs is a very interdisciplinary problem involving techniques and approaches applied in many other fields of computer science and operations research. Chapter 2 reviewed relevant related work with particular interest in periodic (*macro-pipelined*) scheduling, retiming, genetic algorithms, stochastic optimization, imprecise computation and performance bounds theory.

MILP optimization was applied to the problem of system-level design of macro-pipelined AHSMs as discussed in Chapter 3. The proposed model address the case of *finite-horizon* static scheduling where processor sharing is allowed. The use of retiming as a task-flow graph transformation technique was incorporated in another MILP model allowing simultaneous optimization of the initiation interval and cost but without processor sharing.

Particular properties of the MILP models for system-level design of ASHMs were discussed in Chapter 4. These properties were the basis for the formulation of the genetic algorithms in the MEGA tool system. Chapter 5 explored different topics on the use of genetic algorithms in system-level design. The advantages of this *soft-computing* paradigm over exact MILP optimization were outlined. An overview of key concepts in genetic algorithms applied to system-level design, including relevant selection schemes and a new *variable size population* were presented on Chapter 5.

The MEGA system, which is a tool set of different genetic algorithms for different paradigms of ASHM design, was described in Chapter 6, that chapter presented two of the deterministic paradigms supported by MEGA: non-pipelined and macro-pipelined (periodic) design. The main genetic data structures and algorithms of MEGA were discussed along with a set of experimental results validating the proposed framework.

The concept of *imprecise computation* was incorporated in MEGA to allow design of low-implementation-cost ASHMs by doing a trade-off among *cost, performance, and quality of results.* This important contribution of this thesis is discussed in Chapter 7 along with a formal MILP formulation. Modifications of the basic genetic framework of MEGA were outlined, new genes were added, the *mutation* operator for the *deterministic* cases was redefined, and a postprocessing heuristic was proposed.

Chapter 8 described our work on the subject of design of ASHMs *in presence of uncertainty.* A precise mathematical for handling uncertainty was introduced for both non-pipelined and pipelined cases. The high computational cost of performing exact optimization was highlighted. Stochastic simulation was incorporated in MEGA to allow probabilistic design with a low computational cost. Experiments with *variance reduction* techniques (*stratified* and *antithetic* sampling) were made to evaluate how effectively they could accelerate the convergence rate of the proposed *Monte Carlo* genetic algorithm.

A genetic approach for faster generation of trade-off surfaces for the different design paradigms supported by MEGA, our final contribution, was presented in Chapter 9. Our method used a progressively improving sequence of trade-off surfaces made of non-inferior solutions generated by the genetic algorithms proposed in this thesis.

10.2 Future Research

This thesis uncovered many possible future areas of research. We highlight below some relevant research topics in system-level and high-level synthesis computer hardware, to which our thesis results can be applied.

10.2.1 Multidimensional Periodic Scheduling Problems

Key applications in multimedia, realtime computation, wireless communication and digital signal processing involve the use of periodic computations: unidimensional (e.g. audio processing) and multidimensional periodic (e.g. video signals). The design of low-cost computer systems for these applications can be formulated by the use of modular integer programming models, whose mathematical properties are not yet well understood. Further research on this field could also lead to improvement of compiler technology for general purpose parallel computation. MEGA, presently limited to unidimensional periodic scheduling, could be extended to handle to this case.

10.2.2 Evolutionary Hardware

There is an increasing interest in the theoretical aspects of evolutionary programming and genetic algorithms for solving different problems in computer science. We plan to work with the concept of evolutionary hardware, which is the application of the same concepts to design of hardware able to evolve during its lifetime, i.e. by trying to meet changing performance requirements from its environment along continuous or temporary degradation of the computer components.

Evolutionary hardware can be of particular importance in designing fault-tolerant computer systems. We are interested in studying the theoretical basis for developing software to be run in an evolving hardware platform. MEGA genetic algorithms could be adapted to be part of the operating system kernel of an evolutionary ASHM.

10.2.3 Logic Design, Computer Arithmetic and High-Level Design

The traditional model for logic design, where gate propagation delays dominate interconnection delays, cannot be applied in modern submicron integrated circuit technologies. Interconnection delays, which are usually circuit dependent, can be described in a probabilistic form. We plan to study how *retiming* and other gate-level transformations can be used in the presence of uncertainty in relation to gate and interconnection delays when optimizing a logic network. Possible mathematical frameworks would include variations of the genetic algorithms used in MEGA.

10.2.4 Performance Bounds for Application Specific Multiprocessors

We plan to develop tight analytic performance bounds for application-specific heterogeneous and homogeneous periodic multiprocessors taking into account communication costs. Another key issue is to analyze the effect of fixed interconnection structures on the performance bounds of a heterogeneous multiprocessor. Interconnection networks such as N-CUBE and MESH are examples of networks that future commercial application-specific multiprocessors will potentially use. MEGA could be used for fast generation of lower-bound surfaces in evaluating the quality of the proposed theoretical work.

Appendix A

Overview of multi-variable probability theory

We present here some of the results in multi-variable probability theory [54] [124] in order to allow us gain a better understanding of system-level design with a probabilistic approach. We assume in our explanation that the $S_X = \{x_0, \ldots, x_{n-1}\}$ is a set of n continuous random independent variables, which define the random column vector X[A.1].

$$X = \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \end{pmatrix} = (x_0, \ldots, x_{n-1})^T$$

The probability $F_X(A \leq X \leq B)$ that $a_0 \leq x_0 \leq b_0$, ..., $a_i \leq x_i \leq b_i$, ..., $a_{n-1} \leq x_{n-1} \leq b_{n-1}$ is given by the following expression.

$$F_X(A \leq X \leq B) = \int_{a_{n-1}}^{b_{n-1}} \cdots \int_{a_0}^{b_0} f_x(x_0, \ldots, x_{n-1})\, dx_0 \cdots dx_{n-1}$$

Where $f_x(x_0, \ldots, x_i, \ldots, x_{n-1}) = f_X(X)$ is the *joint probability density function* in the space R_n associated with the random variables in S_X. The vectors $A = (a_0, \ldots, a_{n-1})^T$ and $B = (b_0, \ldots, b_{n-1})^T$ are constants.

The *joint cumulative distribution* $F_X(X \leq B)$ is given by

[A.1] M^T is the *transpose* of matrix M. The transpose of a *row vector* is a *column vector* and vice-versa.

$$F_X(X \leq B) = \Phi_X(b_0, \ldots, b_{n-1}) = \int_{-\infty}^{b_{n-1}} \cdots \int_{-\infty}^{b_0} f_x(x_0, \ldots, x_{n-1})\, dx_0 \cdots dx_{n-1}$$

Where $f_x(x_0, \ldots, x_{n-1})$ is a positive function over the space R_n and the integral of the function over the space is equal to 1.

$$f(x_0, \ldots, x_{n-1}) \geq 0$$

$$\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f_x(x_0, \ldots, x_{n-1})\, dx_0 \cdots dx_{n-1} = 1$$

Whenever $\Phi_X(b_0, \ldots, b_{n-1})$ is continuous and differentiable

$$f(x_0 = b_0, \ldots, x_{n-1} = b_{n-1}) = \frac{\partial^n}{\partial b_0 \cdots \partial b_{n-1}} \Phi_X(b_0, \ldots, b_{n-1})$$

Given a set of n linear independent random variables $z_0, \ldots, z_{n-1}$, where

$$z_i = \sum_{j=0}^{n-1} a_{ij} x_j$$

We can define a non-singular matrix $\mathcal{A}$, i.e. $det(\mathcal{A}) \neq 0$, such that

$$\mathcal{A} = \{a_{ij}, 0 \leq i, j \leq n-1\}$$

$$Z = \mathcal{A}X \text{ , } where\ Z = (z_0, \ldots, z_{n-1})^T$$

The joint probability density function $f_x = f(z_0, \ldots, z_{n-1})$ can be expressed in the following way:

$$f_z(z_0, \ldots, z_{n-1}) = \frac{f_x(x_0, \ldots, x_{n-1})}{J_X^Z}$$

Where J_X^Z is the *Jacobian determinant* of Z in function of X.

$$J_X^Z = J\begin{pmatrix} z_0 \ldots z_{n-1} \\ x_0 \ldots x_{n-1} \end{pmatrix}$$

$$J_X^Z = |det(M)|$$

$$M = \{m_{ij} = \frac{\partial z_i}{\partial x_j}, 0 \leq i, j \leq n-1\}$$

As Z is a linear transformation of X, we have

$$J_Z^X = |det(\mathcal{A}|)$$

These results can now be applied to the special case when the random variables $x_0, \ldots, x_{n-1}$ are independent and they have the Gaussian distribution:

$$f_x(x_0, \ldots, x_{n-1}) = \prod_{i=0}^{n-1} f_{x_i}(x_i)$$

$$f_x(x_0, \ldots, x_{n-1}) = \frac{1}{(2\pi)^(n/2)\sigma_0 \ldots \sigma_{n-1}} exp[-\frac{1}{2}\sum_{i=0}^{n-1}(\frac{x_i - \mu_i}{\sigma_i})^2]$$

$$\mu_i = E(x_i)\ is\ the\ expected\ (average)\ value\ of\ x_i$$

The result above can be expressed in a matrix form, by means of the covariance matrix $\mathcal{K}$ for the more general case where $covar(x_i, x_j)$ does not need to be zero for all $i \neq j$, i.e. K does not need to be a diagonal matrix.

$$\mathcal{K} = \{k_{ij} = covar(x_i, x_j), 0 \leq i, j \leq n-1\}$$

Where $covar(x_i, x_i) = var(x_i)$ and $\mathcal{K}$ is positive definite, i.e., $\forall Y \in R_n \quad Y^T\mathcal{K}Y > 0$.

$$f_X(X) = \frac{1}{(2\pi)^(n/2)[det\mathcal{K}]^{\frac{1}{2}}} exp[-\frac{1}{2}(X - E(X))^T\mathcal{K}^{-1}(X - E(X)))]$$

As for the case where $Z = \mathcal{A}X$

$$f_Z(Z) = \frac{f_X(X)}{J_X^Z} = \frac{f_X(X)}{|det(\mathcal{A})|}$$

Some algebraic manipulation [124] leads to the following result:

$$f_Z(Z) = \frac{1}{(2\pi)^{n/2}[det\mathcal{Q}]^{\frac{1}{2}}} exp[-\frac{1}{2}(Z - E(Z))^T \mathcal{Q}^{-1}(Z - E(Z))]$$

Where

$$\mathcal{Q} = \mathcal{A}\mathcal{K}\mathcal{A}^T$$

and

$$E(Z) = \mathcal{A}E(X)$$

The matrix $\mathcal{Q}$ will be positive definite if $\mathcal{K}$ is positive definite.
This result can be generalized [124] for the case where $\mathcal{A}$ is a $m \times n$ matrix with m linear independent lines.

$$Z = \mathcal{A}X\,, \mathcal{A}\, is\, an \quad m \times n\, matrix$$

$$\mathcal{Q} = \mathcal{A}\mathcal{K}\mathcal{A}^T\, is\, an \quad m \times m\, matrix$$

Where Q will be positive definite if $\mathcal{K}$ is positive definite.

Another important finding [124] is that given a vector X of random variables with a non-diagonal but positive definite covariance matrix K we can always find a non-singular linear transformation C such that $Z = \mathcal{C}^{-1}X$ and the covariance matrix of Z is a positive definite diagonal matrix. The matrix C is given by

$$\mathcal{K} = \mathcal{C}\mathcal{C}^T$$

Reference List

[1] AGRAWAL, R. and JAGADISH, H. V., Partitioning Techniques for Large-Grained Parallelism, *IEEE Transactions on Computers*, vol. 37, no. 12, pp. 1627-1634, December 1988.

[2] ALI, H. H. and EL-REWINI, Task Allocation in Distributed Systems: A Split Graph Model, *Technical Report UNO-CS-TR-91-22*, Department of Math and Computer Science, University of Omaha, 1991.

[3] ARUNKUMAR, S. and CHOCKALINGAM, Genetic Search Algorithms and their Randomized Operators, *Computer Mathematical Applications*, vol. 25, no. 5, pp. 91-100, 1993.

[4] BARSKIY, A. B., Two Problems of Optimization of the Use of Non-Homogeneous Computational Systems, *Engineering Cybernetics*, vol. 4, no. 4, pp. 694-699, 1971

[5] BARSKIY, A. B., Minimization of Carrying Capacity of Exchange Lines in a Special-Purpose Computation System, *Eng. Cybernetics*, vol. 10, no. 6, pp. 1101-1106, 1972.

[6] BARTHOU, D., GASPERONI, F. and SCHWIEGELSHON, U., Allocating Communication Channels to Parallel Tasks, *in Environments and Tools for Parallel Scientific Computing*, Elsevier Science Publishers B.V., pp. 275-289, 1993.

[7] BARON, S. and WILSON, W. R., MPEG Overview, *SMPTE Journal - Society of Motion Picture and Television Engineers*, vol. 103, no. 6, pp. 391-394, June 1994.

[8] BATTLE, D. and VOSE, M., Isomorphisms of Genetic Algorithms,*Artificial Intelligence*, Elsevier Science Publications, vol. 60, pp. 155-165, 1993

[9] BERTONI, A. and DORIGO, M., Implicit parallelism in genetic algorithms,*Artificial Intelligence*, Elsevier Science Publications, vol. 61, pp. 307-314, 1993.

[10] BOKHARI, S. H., Assignment Problems in Parallel and Distributed Computing, Kluwer Academic Publishers, 1987.

[11] BRAND, S., The Media Lab : Inventing the Future at MIT, Kluwer Academic Pub., New York, 1988.

[12] BURT, J. M. and GARMAN, M. B., Conditional Monte Carlo: A Simulation Technique for Stochastic Network Analysis, *Management Science*, vol. 18, no. 3, pp. 207-217, November 1971.

[13] CHAO, L.F. and LaPAUGH, A., Rotation Schedule: A Loop Pipelining Algorithm, *30th ACM/IEEE Design Automation Conference*, pp. 566-572, 1993.

[14] CHEN, C. T., GUPTA, P., DeSouza-BATISTA, J. C. and PARKER, A. C., Rapid Prototyping of JPEG Image Compression System using Systems Tools, *Proceedings of the Data Compression Conference*, 1994.

[15] CHO, Y. and SHANI, S., Bounds for List Schedules on Uniform Processors, *SIAM Journal of Computing*, vol. 9, no. 1, pp. 91-103, February 1980.

[16] CHU, W. W., HOLLAWAY, L.J. and EFE, K., Task Allocation in Distributed Data Processing, *Computer*, vol. 13, no. 11, pp. 57-69, Nov 1980.

[17] CHU, W. W. and LAN, L. M., Task Allocation and Precedence Relations for Distributed Real-Time Systems, *IEEE Transactions on Computers*, vol. C-36, no. 6, pp. 667-679, June 1987.

[18] CHUNG, J.Y., LIU, J. W. S. and LIN, K-J., Scheduling Periodic Jobs that Allow Imprecise Results, *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1156-1174, September 1990.

[19] CORMEN, T. H., LEISERSON, C. E. and RIVEST, R. L., Introduction to Algorithms, McGraw-Hill Book Company, New York, 1989.

[20] CHRETIENNE, P., The Basic Cyclic Scheduling Problem with Deadlines, *Discrete Applied Mathematics*, vol. 30, pp. 109-123, 1991.

[21] DeSOUZA-BATISTA, J.C., ESHAGHIAN, M., PARKER, A. C., PRAKASH, S. and WU, Y. C., A Sub-Optimal Assignment of Application Tasks onto Heterogeneous Systems, *Workshop on Heterogeneous Processing, International Parallel Processing Symposium*, 1994.

[22] DeSOUZA-BATISTA, J. C. and PARKER, A. C., Optimal Synthesis of Application Specific Heterogeneous Pipelined Multiprocessors, *Proceedings of the International Conference on Application-Specific Array Processors*, August 1994.

[23] DeSOUZA-BATISTA, J. C., POTKONJAK, M. and PARKER, A. C., Optimal ILP-based Approach for Throughput Optimization Using Simultaneous Algorithm/Architecture Matching and Retiming, Proceedings of the 1995 Design Automation Conference, June 1995.

[24] DODIN, B., Approximating the Distribution Functions in Stochastic Networks, *Computer & Operations Research*, vol. 12, no. 3, pp. 251-264, 1985.

[25] DODIN, B., Bounding the Project Completion Time Distribution in PERT Networks, *Operations Research*, vol. 33, no. 4, pp. 862-881, July 1985.

[26] EFE, K., Heuristic Models of Task Assignment Scheduling in Distributed Systems, *Computer*, vol. 15, no. 6, pp. 50-56, June 1982.

[27] ELMAGHRABY, S. E., The Theory of Networks and Management Science: Part II, *Management Science*, vol. 17, no. 2, pp. B.54-B.71, 1970.

[28] EL-REWINI, H. and LEWIS, T. G., Scheduling Parallel Programs Tasks onto Arbitrary Target Machines, *Journal of Parallel and Distributed Computing*, no. 9, pp. 138-153, 1990.

[29] ELSAYED, E. A. and ETTOUNEY, M., Pertubation Analysis of Linear Programming Problems with Random Parameters, *Computers and Operations Research*, Pergamon Press Ltd., vol. 21, pp. 211-224, 1994.

[30] FENG, D. T. and SHIN, K. G., Static Allocation of Periodic Tasks with Precedence Constraints in Distributed Real-Time Systems, *Proceedings of the 9th International Conference of Distributed Computing*, pp. 190-198, June 1989.

[31] FERNANDEZ, E. B. and BUSSEL, B., Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules, *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 745-751, August 1975.

[32] FISHER, D. L., SAISI, D. and GOLDSTEIN, W. M., Stochastic Pert Networks: OP Diagrams, Critical Paths and the Project Completion Time, *Computers & Operations Research*, vol. 12, no. 5, pp. 471-482, 1985.

[33] FOGEL, D., An Introduction to Simulated Evolutionary Optimization, *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 3-14, January 1994.

[34] FORREST, S., Genetic Algorithms: Principles of Natural Selection Applied to Computation, *Science*, vol. 261, pp. 872-878, August 1993.

[35] FRIESEN, D. K. and LANGSTON, M. A., Bounds for Multifit Scheduling on Uniform Processors, *SIAM Journal of Computing*, vol. 12, no. 1, pp. 60-70, February 1983.

[36] FRIESEN, D. K., Tighter Bounds for LPT Scheduling on Uniform Processors, *SIAM Journal of Computing*, vol. 16, no. 3, pp. 554-560, June 1987.

[37] FULKERSON, D. R., Expected Critical Path Lengths in Pert Networks, *Operations Research*, vol. 10, no. 5, pp. 808-817, November 1962.

[38] GEBOTYS, C. H., Optimal Scheduling and Allocation of Embedded VLSI Chips, *Proceedings of the 29th ACM/IEEE Design Automation Conference*, pp. 116-119, 1992.

[39] GELABERT, P. R., BARNWELL, T. P., Optimal Automatic Periodic Multiprocessor Scheduler for Fully Specified Flow Graphs, *IEEE Transactions on Signal Processing*, vol. 41, no. 2, pp. 858-888, February 1993.

[40] GRAHAM, R. L., Bounds on Multiprocessing Timing Anomalies, *SIAM Journal of Computing*, vol. 17, no. 2, pp. 416-429, 1969.

[41] GAREY, M. R. and GRAHAM, R. L., Bounds for Multiprocessor scheduling with Resource Constraints, *SIAM Journal of Computing*, vol. 4, no. 2, pp. 187-200, June 1975.

[42] GAREY, M. R., GRAHAM, R. L. and JOHNSON, D. S., Performance Guarantees for Scheduling Algorithms, *Operations Research*, vol. 26, no. 1, pp. 3-21, February 1978.

[43] GAREY, M. R., JOHNSON, D. S., Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

[44] GERASOULIS, A. and YANG, T., A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors, *Journal of Parallel and Distributed Computing*, no. 16, pp. 276-291, 1992.

[45] GAUDIOT, J-L. and BIC, L., Advanced Topics in Data-Flow Computing, Prentice Hall Inc., New Jersey, 1991.

[46] GUPTA, P., CHEN, C. T., DeSOUZA-BATISTA, J. C., PARKER, A. C., Experience with Image Compression Chip Design using Unified System Construction Tools, *Proceedings of the 31st Design Automation Conference*, June 1994.

[47] GUSFIELD, D., Parametric Combinatorial Computing and a Problem of Program Module Distribution, *Journal of the Association for Computing Machinery*, vol. 30, no. 3, pp. 551-563, July 1983.

[48] HADDAD, E. K., Optimal Load Allocation for Parallel and Distributed Processing, *Technical Report TR 89-12*, Department of Computer Science, Virginia Polytechnic Institute and State University, April 1989.

[49] HADDAD, E. K., Analysis, Modeling and Optimization of Multiprocessing Execution Time, *Technical Report TR89-11*, Department of Computer Science, Virginia Polytechnic Institute and State University, April 1989.

[50] HAFER, L.J. and HUTCHINGS, E., Bringing up Bozo, *Tech Rep. CMPT TR 90-2*, School of Computing Science, Simon Fraser University, Burnaby, B.C., V5A 1S6, March 1990.

[51] HAFER, L. AND PARKER, A., Automated Synthesis of Digital Hardware, *IEEE Transcations on Computers* , vol. C-31, no. 2, pp. 93-109, February 1981.

[52] HAGSTROM, J. N., Computing the Probability Distribution of Project Duration in a PERT Network, *Networks*, John Wiley & Sons, vol. 20, pp. 231-244, 1990.

[53] HANEN, C. and MUNIER, A., A Study of the Cyclic Scheduling Problem on Parallel Processors, Discrete Applied Mathematics, vol. 57, pp. 167-192, 1995.

[54] HELSTROM, C. W., Probability and Stochastic Processes for Engineers, Macmillan Publishing Company, New York, 1994.

[55] HO, K., LEUNG, J. Y-T. and WEI, W-D., Minimizing Maximum Weighted Error for Imprecise Computation Tasks, *Technical Report UNL-CSE-92-017*, Department of Computer Science and Engineering, University of Nebraska, Lincoln, 1992.

[56] HOMAIFAR, A., QI, C. X. and LAI, S. H., Constrained Optimization Via Genetic Algorithms, *Simulation*, vol. 62:4, pp. 242-254, April 1994.

[57] HOU, E.S.H. ANSARI, N. and REN, H., A Genetic Algorithm for Multiprocessor Scheduling, *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 2, pp. 113-120, February 1994.

[58] HWANG, K., Advanced Computer Architecture : Parallelism, Scalability, Programmability, McGraw-Hill, 1993.

[59] HWANG, J.J., CHOW, Y. C., AHNGER, F. D. and LEE, C. Y., Scheduling Precedence Graphs in Systems with Interprocessor Communication Times, *SIAM Journal of Computing*, vol. 18, no. 2, pp. 244-257, April 1989.

[60] INDURKHYA, B., STONE, H. S. and CHENG, L. X., Optimal Partitioning of Randomly Generated Distributed Programs, *IEEE Transactions on Software Engineering*, vol. SE-12, no. 3, pp. 483-495, March 1986.

[61] JAFFE, J. M., Bounds on the Scheduling of Typed Task Systems, *SIAM Journal of Computing*, vol. 9, no. 3, pp. 541-551, August 1991.

[62] KAMBUROWSKI, J., An Upper Bound on the Expected Completion Time of PERT Networks, *European Journal of Operational Research*, vol. 21, pp. 206-212, 1985.

[63] KAMBUROWSKI, J., A Note on the Stochastic Shortest Route Problem, *Operations Research*, vol. 33, no. 3, pp. 696-698, May 1985.

[64] KASAHARA, H. and NARITA, S., Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing, *IEEE Transactions on Computers*, vol. C-33, no. 11, pp. 1023-1029, November 1984.

[65] KLEINDORFER, G. B., Bounding Distributions for a Stochastic Acyclic Network, *Operations Research*, vol. 19, no. 7, pp. 1586-1601, November 1971.

[66] KORYACHKO, V. P., Efficient Organization of a System of Processors of a Specialized Multiprocessor System, *Engineering Cybernetics*, vol. 16, no. 4, pp. 83-89, 1978.

[67] KÜÇÜKÇAKAR, K., System-Level Synthesis Techniques with Emphasis on Partitioning and Design Planning, Ph.D. Thesis, Department of Electrical Engineering and Systems, September 1991.

[68] KÜÇÜKÇAKAR, K., CHOP: A Constraint-Driven System-Level Partitioner, *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pp. 514-519, 1991.

[69] KÜÇÜKÇAKAR, K. and PARKER, A. C., A Methodology and Design Tools to Support System-Level VLSI Design, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* , vol. 3, no. 3, pp. 355-369, September 1995.

[70] KURDAHI, F. J., Techniques for Area Estimation of VLSI Layouts, *IEEE Transaction on Computer-Aided Design*, vol. 8, no. 1, pp. 81-92, January 1989.

[71] KULKARNI, V. G. and ADLAKHA, V. G., Markov and Markov-Regenerative Pert Networks, *Operations Research*, vol. 34, no. 5, pp. 769-781, September 1986.

[72] KUNDE, M., Nonpreemptive LP-Scheduling on Homogeneous Multiprocessor Systems, *SIAM Journal of Computing*, vol. 10, no. 1, pp. 151-173, February 1981.

[73] Lamps User Guide - Linear and Mathematical Programming System, Version 1.68, Advanced Mathematical Software Ltd, London, 1994.

[74] LEE, C. Y., HWANG, J. J., CHOW, Y. C. and ANGER, F. D., Multiprocessor Scheduling with Interprocessor Communication Delays, *Operations Research Letters*, vol. 7, no. 3, pp. 141-147, June 1988.

[75] C. L. LEISERSON and J. B. SAXE, Retiming Synchronous Circuitry, *Algorithmica*, vol. 6, pp. 5-35, 1991.

[76] LINSKY, V.S. and KORNEV, M.D., Construction of Optimum Schedules for Parallel Processors, *Engineering Cybernetics*, vol. 10, no. 3, pp. 506-514, 1972.

[77] SCHRAGE, L. E., Linear, Integer, and Quadratic Programming with LINDO, Scientific Press, Palo Alto, CA, 1984.

[78] LIU, J.W.S. and LIU, C. L., Performance Analysis of Multiprocessor Systems Containing Functionally Dedicated Processors, *Acta Informatica 10*, pp. 95-104, 1978

[79] LIU, J.W.S. LIN, K.-J., SHIH, W.-K., YU, A. C.-S., CHUNG, J.-Y. and ZHAO, W., Algorithms for Scheduling Imprecise Computations, *IEEE Computer*, vol. 24, no. 5, pp. 58-68, May 1991.

[80] LUENBERGER, D. G., Introduction to linear and nonlinear programming, Addison-Wesley Pub. Co., 1974.

[81] LUKASZEWICS, J., On the Estimation of Errors Introduced by Standard Assumptions Concerning the Distribution of Activity Duration in Pert Calculations, *Operations Research*, vol. 13, no. 2, pp. 326-327, March 1965.

[82] MA, P. R., LEE, E. Y. and TSUCHIYA, M., A Task Allocation Model for Distributed Computing Systems, *IEEE Transactions on Computers*, vol. C-31, no. 1, pp. 41-47, January 1982.

[83] MADISETTI, V., VLSI Digital Signal Processors - An Introduction to Rapid Prototyping and Design Synthesis, Butterworth-Heinemann and IEEE Press, 1995.

[84] MALCOLM, D. G., ROSEBOOM, J.H., CLARK, C. E. and FAZAR, W., Application of a Technique for Research and Development Program Evaluation, *Operations Research*, vol. 7, no. 5, pp. 646-669, September 1959.

[85] MARTIN, J. J., Distribution of the Time Through a Directed, Acyclic Network, *Operations Research*, vol. 13, no. 1, pp. 46-66, January 1965.

[86] MCFARLAND, M. C. and KOWALSKI, T. J., Incorporating Bottom-Up Design into Hardware Synthesis, *IEEE Transactions on Computer-Aided Design*, vol. 9, no. 9, pp. 938-950, September 1990.

[87] MEHROTRA, R. and TALUKDAR, S. N., Scheduling of Tasks for Distributed Processors, *Technical Report DRC-18-68-84*, Design Research Center, Carnegie-Mellon University, December 1984.

[88] MEHROTRA, K., CHAI, J. and PILLUTLA, S., A Study of Approximating the Moments of the Job Completion Time in PERT Networks, School of Computer and Information Science, Syracure University, New York, 1991.

[89] MICHALEWICZ, Z., Genetic Algorithms + Data Structures = Evolution Programs , Springer-Verlag, Berlin, 1994.

[90] MOUHAMED, M., Lower Bound on the Number of Processors and Time for Scheduling Precedence Graphs with Communication Costs, *IEEE Transactions on Software Engineering*, vol. 16, no. 12, December 1990.

[91] MUTKA, M. W. and LI, J-P., A Tool for Allocating Periodic Real-Time Tasks to a Set of Processors, *Systems Software*, vol. 29, pp. 135-148, 1995.

[92] NEMHAUSER, G. L. and WOLSEY, L. A., Integer and Combinatorial Optimization, John Wiley & Sons, New York, 1988.

[93] NICOL, D. M., Optimal Partitioning of Random Programs across two Processors, *IEEE Transactions on Software Engineering*, vol. 15, no. 2, pp. 134-141, February 1989.

[94] PARK, N. and PARKER, A. C., Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications.*IEEE Transactions on Computer-Aided Design*, vol. 7, no.3, pp. 356-369, March 1988.

[95] PARK, N. and PARKER, A. C., Theory of Clocking for Maximum Execution Overlap of High-Speed Digital Systems, *IEEE Transactions on Computers*, vol. 37, no. 6, pp. 678-690, June 1988.

[96] PARHI, K. and MESSERSCHMITT, D. G., Static Rate-Optimal Scheduling of Iterative Data-Flow Programs via Optimum Unfolding, *IEEE Transactions on Computers*, vol. 40, no. 2, pp. 178-195, February 1991.

[97] PARHI, K. and LUCKE, L. E., Data-flow Transformations for Critical Path Time Reduction in High-Level DSP Synthesis, *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 7, pp. 1063-1068, July 1993.

[98] PASSOS, N. L., SHA, E. H. and BASS S. C., Optimizing DSP Flow-Graphs via Schedule-Based Multidimensional Retiming, *IEEE Transaction on Signal Processing*, vol. 44, no. 1, pp. 150-155, January 1996.

[99] PRAKASH, S. and PARKER, A. C., SOS: Synthesis of Application Specific Heterogeneous Multiprocessor Systems, *Journal of Parallel and Distributed Computing*, no. 16, pp. 338-351, December 1992.

[100] PRAKASH, S., Synthesis of Application-Specific Multiprocessor Systems, *Ph.D. Thesis*, Department of Electrical Engineering and Systems, University of Southern California, January 1994.

[101] PURUSHOTHAMAN, S. and SUBRAHMANYAM, P. A., Reasoning About Probabilistic Behavior in Concurrent Systems, *IEEE Transactions on Software Engineering*, vol. SE-13, no. 6, pp. 740-745, June 1987.

[102] RAMAMRITHAM, K., STANKOVIC, J. A. and SHIAH, P.F., Efficient Scheduling Algorithms for Real-Time Multiprocessors Systems, *IEEE Transaction on Parallel and Distributed Systems*, vol. 1, no. 2, pp. 184-194, April 1990.

[103] RAMAMRITHAM, K., Allocation and Scheduling of Precedence-Related Periodic Tasks, *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 4, April 1995.

[104] RAVIKUMAR, C.P. and GUPTA, A., Genetic Algorithm for Mapping Tasks onto a Reconfigurable Parallel Processor, *IEE Proceedings in Computing Digital Technology*, vol. 142, no. 2, pp. 81-86, March 1995.

[105] RAWLINS, G. J. E. (editor), Foundations of Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, California, 1991.

[106] REEVES, C. R., Modern Heuristic Techniques for Combinatorial Problems, John Wiley & Sons, Inc., New York, 1993

[107] RENFORS, M. and NEUVO, Y., The Maximum Sampling Rate of Digital Filters Under Hardware Speed Constraints, *IEEE Transcations on Circuits and Systems*, vol. CAS-28, no. 3, pp. 196-202, March 1981.

[108] RICH, E., Artificial Intelligence, McGraw-Hill Inc., 2nd ed., New York, 1991.

[109] RINGER, L. J., Numerical Operators for Statistical Pert Critical Path Analysis, *Management Science*, vol. 16, no. 2, pp. B.136-B.143, October 1969.

[110] RIPLEY, B. D., Stochastic Simulation, John Wiley & Sons, 1987.

[111] ROBILLARD, P. and TRAHAN, M., The Completion Time of PERT Networks, *Operations Research*, vol. 25, no. 1, pp. 15-29, January 1977.

[112] ROZEMBERG, G., Advances in Petri Nets, Springer-Verlag, 1992.

[113] RUDOLPH, G., Convergence Analysis of Canonical Genetic Algorithms, *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 96-101, January 1994.

[114] RUPPERT, D., REISH, R. L., DERISO, R. B. and CARROL, R. J., Optimization Using Stochastic Approximation and Monte Carlo Simulation (with Application to Harvesting of Atlantic Menhaden), *Biometrics*, vol. 40, pp. 535-545, June 1984.

[115] SASTRY, S., Wiring Space Estimation of Master Slice ICs, Technical Report DISC/83-5, Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, June 1983.

[116] SASTRY, S., Wiring Analysis of Integrated Circuits, Technical Report DISC/85-2, Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, October 1985.

[117] SASTRY, S. and PARKER, A. C., Stochastic Models for Wireability Analysis of Gate Arrays, *IEEE Transactions on Computer-Aided Design*, vol. CAD-5, no. 1, January 1986.

[118] SHEN, C. C. and TSAI, W. H., A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion, *IEEE Transactions on Computers* ,vol. C-34, no. 3, pp. 197-203, March 1985.

[119] SHIH, W. K., LIU, J. W. S. and CHUNG, J.-Y., Algorithms for Scheduling Imprecise Computations with Timing Constraints, *SIAM journal of Computing*, vol. 20, no. 3, pp. 537-552, June 1991.

[120] SHOGAN, A. W., Bounding Distributions for a Stochastic PERT Network, *Networks*, John Wiley & Sons Inc., vol. 7, pp. 359-381, 1977.

[121] SHROFF, P., WATSON, D. W., FLANN, N. S. and FREUND, R. F., Genetic Simulated Annealing for Scheduling Data-Dependent Tasks in Heterogeneous Environments, *Proceeding of Heterogeneous Computing Workshop, 1996 International Parallel Processing Symposium*, IEEE Computer Society Press, Honolulu, Hawaii, pp. 98-104, 1996.

[122] SIGAL, C. E., PRITSKER, A. A. B. and SOLBERG, J. J., The Stochastic Shortest Route Problem, *Operations Research*, vol. 28, no. 5, pp. 1122-1129, September 1980.

[123] SINGH, H. and YOUSSEF, A., Mapping and Scheduling Heterogeneous Task Graphs using Genetic Algorithms, *Proceeding of Heterogeneous Computing Workshop, 1996 International Parallel Processing Symposium*, Honolulu, Hawaii, pp. 86-97, 1996.

[124] STARK, H. and WOODS, J., Probability, Random Processes, and Estimation Theory for Engineers, Prentice-Hall, New Jersey, 1994.

[125] STONE, H. S. and BOKHARI, S. H., Control of Distributed Processes, *Computer*, no. 11, pp. 97-106, July 1978.

[126] TCHA, D. W., LEE, B.I. and LEE, Y. D., Processors Selection and Splitting in a Parallel Processors System, *Acta Informatica*, no. 28, pp. 415-423, 1992.

[127] TIRAT-GEFEN, Y.G. and PARKER, A. C., MEGA: An Approach to System-Level Design of Application-Specific Heterogeneous Multiprocessors, *Proceeding of Heterogeneous Computing Workshop, of the 1996 International Parallel Processing Symposium*, Honolulu, Hawaii, pp. 105-117, 1996.

[128] TIRAT-GEFEN, Y. G. and PARKER, A. C., Probabilistic approaches to system-level design. Technical Report *in preparation*, Department of Electrical Engineering and Systems, University of Southern California, September 1996.

[129] THOMASIAN, A., Analytic Queueing Network Models for Parallel Processing of Task Systems. *IEEE Transactions on Computers*, vol. C-35, no. 12, pp. 1045-1054, December 1986.

[130] VAN SLYKE, R. M., Monte Carlo Methods and the Pert Problem. *Operations Research*, vol. 11, no. 5, September 1963.

[131] VAYRADYAN, A. S., Planning of Parallel Computations in a Multiprocessor System of Real Time. *Engineering Cybernetics*, vol. 19, no. 2, pp. 120-129, 1981.

[132] VENKOVA, N. B., A Schedule that Minimizes the Number of Processors in a Homogeneous System. *Engineering Cybernetics*, vol. 9, no. 5, pp. 884-886, 1971.

[133] VERHAUGER, W.F., Multidimensional Periodic Scheduling, Ph.D. Thesis, Eindhoven University of Technology, Holland, 1995.

[134] WANG, D.J. and HU, Y. H., Fully Static Multiprocessor Array Realizability Criteria for Real-Time Recurrent DSP Applications', *IEEE Transcations on Signal Processing*, vol. 42, no. 5, pp. 1288-1292, May 1994.

[135] WANG, D. J. and HU, Y. H., Multiprocessor Implementation of Real-Time DSP Algorithms. *IEEE Transactions on Very large Scale Integration (VLSI) Systems*, vol. 3, no. 3, pp. 393-403, September 1995.

[136] WANG, L., SIEGEL, H. J. and ROYCHOWDHURY, V. P., A Genetic-Algorithm-Based Approach for Task Matching and Scheduling in Heterogeneous Computing Environments. *Proceeding of Heterogeneous Computing Workshop, 1996 International Parallel Processing Symposium*, Honolulu, Hawaii, pp. 72-85, 1996.

[137] WHITLEY, D., A Genetic Algorithm Tutorial. Technical Report CS-93-103, Colorado State University, November, 1993.

[138] WOJCIECHOWSKI, J. and VLACH, J., A Method for Statistical Design Centering. *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, pp. 33-36, 1992.

[139] WOLF, R. W., Stochastic Modeling and the Theory of Queues, Prentice-Hall, New Jersey, 1989.