Computer aided design techniques for network-on-chip architectures ...
Srinivasan, Krishnan
Dissertation Abstracts International; 2006; 67, 11; SciTech Premium Collection

COMPUTER AIDED DESIGN TECHNIQUES FOR NETWORK-ON-CHIP ARCHITECTURES AND SYSTEM-LEVEL LOW POWER OPTIMIZATION

by

Krishnan Srinivasan

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

ARIZONA STATE UNIVERSITY

December 2006

UMI Number: 3241349

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI®

UMI Microform 3241349

Copyright 2007 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

المنارة للاستشارات

COMPUTER AIDED DESIGN TECHNIQUES FOR NETWORK-ON-CHIP
ARCHITECTURES AND SYSTEM-LEVEL LOW POWER OPTIMIZATION

by

Krishnan Srinivasan

has been approved

October 2006

APPROVED:

, Chair

Sarma Vrudhula

Chaitali Chakrabarti

Supervisory Committee

ACCEPTED:

Department Chair

Dean, Division of Graduate Studies

ABSTRACT

System-on-Chip (SoC) integrates several processing cores, ASIC blocks, memory, and communication elements on a single chip. The International Technology Roadmap for Semiconductors (ITRS) predicts that future generations of the high end SoC architectures will be implemented in less than 50nm technology, and clocked in the multi GHz range. These architectures will be composed of tens to hundreds of cores communicating with each other at several Gigabits per second.

Along with performance, power minimization will be an important design goal in nanoscale SoC architectures. Aggressive performance optimization and power minimization techniques will be applied by dividing these architectures into voltage and clock islands. These islands will operate on local clocks. Communication between the voltage islands, also known as global communication, will take place asynchronously, thus obviating the need for a global clock. Such architectures are known as Globally Asynchronous and Locally Synchronous (GALS) based systems.

This thesis addresses two important problems in GALS based SoC design: i) Design and optimization of global or inter-core communication architecture, and ii) system-level power minimization. As a primary contribution, the thesis addresses the optimization problems of the communication architecture in the context of a new communication paradigm to meet the performance requirements in future SoCs. As a secondary contribution, it addresses the problem of low power mapping and scheduling in multiprocessor SoC architecture. We propose mathematical models, heuristic techniques, and provably good polynomial time algorithms to solve the optimization problems of future SoC architectures.

Network-on-Chip (NoC) has been proposed as a solution for the global communication challenges in GALS based architectures. NoC supports asynchronous transfer of data, and given a suitable topology, can provide extremely high bandwidth by pipelining signal transmission, and by supporting concurrent communication. This thesis presents techniques for the design of low power NoC architectures, under performance constraints.

Dynamic voltage scaling (DVS), and dynamic power management (DPM) are two well known system-level power minimization schemes that are employed at the board level. GALS based SoC architectures will also support these power minimization techniques. The thesis presents techniques that integrate DVS and DPM with algorithmic transformations namely, pipelining and loop unrolling to minimize the power consumption of a multiprocessor GALS based architecture, under performance constraints.

To my parents

ACKNOWLEDGMENTS

I vividly remember the day when I stopped by Dr Chatha's office and talked to him about a possibility of a PhD under his supervision. Having him as a supervisor would have required that I change my major from Electrical Engg. to Computer Science. So, here I was, knocking on the door of a Professor just out of college, in a department alien to me. If I were to take such a risk today, I would certainly think twice, if not reject it right away. However, I am glad I took the risk. Most of the times, it is these seemingly innocuous decisions that cause major changes in a person's life. In my case, I am on the threshold of getting a PhD.

I would like to express my regards and thanks to my advisor, Dr. Karamvir S. Chatha for his continued guidance and support during my time in graduate school. Over the past four and a half years, he has helped me mature as a researcher, as well as an individual.

Thanks to Prof. Goran Konjevod for serving in my committee and giving me invaluable advice, specially pertaining to linear programming based algorithms.

My sincere thanks to Profs. Sarma Vrudhula and Chaitali Chakrabarti for serving in my committee and providing insights into my research.

Over the years, there have been so many members in the CSRL lab that it is impossible to list all of them. I have seen a lot of them graduate, and have been part of their joy and relief. Now that its my turn, I thank them for all the support and company. Be it intellectual discussion on research topics, or grappling about where to lunch, or the recent business competition, every single one of them has been a wonderful lab mate. Thanks guys!!!

In the first year of my PhD, I was desperately trying to obtain a linear programming solver. And then one day, I went to Prof. Hans Mittelmann in the Department of Math. This thesis would not have been possible but for his help with linear programming solvers. Thanks Dr Mittelmann.

I have known Ms. Renate Mittelmann for over five years now. She let me use her lab space, and computers for my research. Her lab was particularly attractive due to the unlimited supply of chocolates and oranges (in winter). She is a wonderful human being, with whom I have always enjoyed talking. I thank her for all her support and encouragement during the course of my PhD.

My parents-in-law have waited patiently for more than a year for me to complete my PhD and get a job. I thank them for their support and prayers.

I would like to thank my brother Raman and sister-in-law Shantha for all their support. I owe everything I am today to my brother.

My sister Lakshmi and brother-in-law Ashutosh introduced me to the idea of a PhD at a very young age. In a way I got a head start, as I knew what to expect from a PhD at an age when most children did not know what PhD stood for. Thanks to you both for being the torch bearers of the family. You have been my role models.

My parents. Well, they have been with me all through. My brother summed it up when he said "They have not only given us our lives, but their lives as well".

Finally, my wife Rekha. She is the best thing that has happened to me.

TABLE OF CONTENTS

Page

LIST OF TABLES xiv

LIST OF FIGURES xvi

CHAPTER

1 INTRODUCTION 1

1.1. Motivation 1

1.2. Summary of research contributions 4

1.2.1. NoC design and optimization 4

1.2.2. System-level low power optimization(VLSI Design 04: [1], ISVLSI 04 [2], Integration, The VLSI Journal (Accepted)) 6

1.3. Thesis organization 6

2 APPLICATION SPECIFIC NOC DESIGN 8

2.1. Introduction 8

2.2. NoC design flow 8

2.3. Router Architecture characterization 11

2.3.1. Power consumption in input port of unit router 13

2.3.2. Power consumption in output port of unit router 14

2.3.3. Power consumption in physical links 15

2.3.4. Latency 15

2.4. Problem Definition 16

CHAPTER Page

2.5. Conclusion 17

3 LITERATURE SURVEY 19

3.1. Introduction 19

3.2. Seminal work 19

3.3. NoC architectures 20

3.3.1. Architectures for best effort traffic 20

3.3.2. Architectures for guaranteed throughput traffic 20

3.3.3. Architectures with error control schemes 21

3.3.4. Architectural optimizations for low power 21

3.4. Performance Evaluation 21

3.5. Automated design techniques 22

4 INTEGER LINEAR PROGRAMMING BASED TECHNIQUES FOR APPLICATION SPECIFIC NOC DESIGN 24

4.1. Introduction 24

4.2. ILP formulations 25

4.2.1. System-level floorplanning 26

4.2.2. Custom interconnection topology and route generation 30

4.3. Clustering based heuristic technique 40

4.4. Deadlock avoidance 42

4.5. Results 44

4.5.1. Benchmark applications 44

4.5.2. Experimental setup 44

4.5.3. Results and discussion . . . 47
4.6. Conclusion . . . 52
5 LOW COMPLEXITY HEURISTICS FOR DESIGN OF CUSTOM NOC ARCHITECTURES . . . 54
5.1. Introduction . . . 54
5.2. Heuristic for Interconnection network generation . . . 54
5.2.1. Link placement and trace mapping . . . 56
5.2.2. Core to router mapping . . . 59
5.2.3. Topology generation . . . 60
5.2.4. Router merging . . . 61
5.3. Complexity analysis . . . 63
5.4. Experimental results . . . 64
5.4.1. Benchmark applications . . . 65
5.4.2. Experimental setup and result analysis . . . 65
5.5. Conclusion . . . 71
6 APPROXIMATION ALGORITHMS FOR MAPPING AND ROUTING PROBLEMS DURING NOC DESIGN . . . 72
6.1. Introduction . . . 72
6.2. Core to router mapping . . . 73
6.2.1. Equivalence to max-flow min-cut problem . . . 74
6.3. Routing and topology generation . . . 76
6.3.1. Topology generation with least number of routers . . . 79

6.3.2. NoC topologies with minimum power consumption 87
6.3.3. Satisfying bandwidth constraints 89
6.3.4. Satisfying port constraints 90
6.3.5. Router merging 93
6.4. Results 94
6.4.1. Benchmark applications 95
6.4.2. Experimental setup 96
6.4.3. Analysis and discussion 97
6.5. Conclusion 101

7 GENETIC ALGORITHM BASED TECHNIQUE FOR NOC DESIGN 103
7.1. Introduction 103
7.2. Genetic algorithm for NoC topology design and route generation 104
7.2.1. Overview of GA 106
7.2.2. Data structures for representation of solution 107
7.2.3. Legality criteria for solution representation 110
7.2.4. Generation of initial population and modified shortest path algorithm (MSP) 112
7.2.5. Pareto points and fitness calculation 116
7.2.6. Overview of the optimization technique 117
7.2.7. Genetic operators 119
7.2.8. Post-synthesis floorplan adjustment 123
7.2.9. Deadlock avoidance 124
7.3. Results 127

7.3.1. Benchmark applications . . . 127
7.3.2. Experimental setup . . . 128
7.3.3. GA tuning parameters . . . 129
7.3.4. Results and discussion . . . 130
7.4. Conclusion . . . 145
8 INTEGER LINEAR PROGRAMMING AND HEURISTICS FOR LOW POWER MAPPING AND SCHEDULING FOR MULTIPROCESSOR ARCHITECTURES, UNDER THROUGHPUT CONSTRAINTS . . . 146
8.1. Introduction . . . 146
8.1.1. Embedded multiprocessor architecture . . . 148
8.1.2. Motivating Example . . . 150
8.1.3. Our Contributions . . . 152
8.2. Previous Work . . . 153
8.2.1. Power minimization in uni-processor systems . . . 153
8.2.2. Power minimization in multiprocessor systems . . . 154
8.2.3. Low power scheduling in high-level synthesis . . . 155
8.2.4. Algorithmic transformations for system-level low power design . . . 155
8.3. MILP Formulations for System Level Low Power Design . . . 156
8.3.1. Problem Definition . . . 156
8.3.2. The MILP formulation for optimal solution . . . 158
8.3.3. MILP formulation that assumes a constant switching overhead ($MILP_{so}$) . 173
8.3.4. MILP formulation that maximizes task run time ($MILP_{lat}$) . . . 173

8.3.5. MILP formulation combining $MILP_{lat}$ and $MILP_{so}$ ($MILP_{ls}$) 174

8.4. Heuristic Techniques for System-level Low Power Design 174

8.4.1. Overview of the loop transformation algorithms 174

8.4.2. Overall heuristic for system-level low power design 175

8.4.3. Deterministic optimization for system-level low power design 179

8.4.4. Simulated annealing based optimization for Low Power Design 182

8.5. Results 184

8.5.1. Experimental set-up 185

8.5.2. Evaluation of MILP formulations 188

8.5.3. Evaluation of heuristic techniques 191

8.6. Conclusion 194

9 CONCLUSION 195

9.1. Application specific NoC design 195

9.1.1. Comparison of the different NoC design techniques 195

9.1.2. Discussion and future directions 196

9.2. System-level power minimization 199

Appendix A 200

REFERENCES 209

LIST OF TABLES

Table Page

1 Graph Characteristics . . . 44

2 Node descriptions . . . 45

3 Results for Floorplanning . . . 45

4 Comparison of ILP, and Clustering . . . 46

5 Comparison of Clustering, and Mesh . . . 46

6 Comparison of Clustering, and QNoC . . . 47

7 Comparison of Clustering final solution with ILP and Clustering lower bounds . . 47

8 Benchmarks . . . 64

9 Techniques . . . 64

10 Node descriptions for MPEG and MWD . . . 69

11 Node description for Set-top box . . . 69

12 Benchmarks . . . 95

13 Average power and router consumption . . . 97

14 Runtimes . . . 97

15 Node description for Set-top box, and NP . . . 99

16 Benchmarks . . . 129

17 Technique nomenclature . . . 130

18 Average power and router consumption . . . 134

19 Runtimes . . . 134

20 Nomenclature of variables . . . 159

21 Summary of results for *MILP* . . . 188

Table Page

22 Summary of results for $MILP_{so}$ 188

23 Summary of results for $MILP_{lat}$ 188

24 Summary of results for $MILP_{ls}$ 188

25 Comparison of deterministic technique with other heuristic techniques 192

26 Comparison of simulated annealing based technique with other heuristic techniques 192

27 Comparison between heuristic and optimum MILP based techniques 192

28 Comparison of all techniques for multimedia applications 192

29 Average power and router consumption 196

30 Runtimes 196

LIST OF FIGURES

Figure Page

1.1 Application specific SoC architecture design 3

2.1 NoC architecture design 9

2.2 Power consumption of NoC 10

2.3 NoC router architecture 12

2.4 Input port power consumption set-up 13

2.5 Output port power consumption set-up 13

2.6 Input port power consumption 14

2.7 Output port power consumption 14

2.8 Link power versus injection rate 14

2.9 Link power versus length 15

2.10 Latency for 2 routers 15

2.11 Latency for 4x4 mesh 15

4.1 ILP based NoC generation 25

4.2 Example mesh based floorplan 29

4.3 Illegal layout in mesh based topology 29

4.4 Example of router allocation for custom topology 32

4.5 Algorithm for removing redundant routers 32

4.6 Adding router to increase number of traffic 39

4.7 Clustering based approach 41

4.8 Breaking deadlocks with additional virtual channels 42

4.9 263 dec mp3 dec: Communication Trace Graph 49

Figure Page

4.10 Custom layout for 263 dec mp3 dec 49

4.11 Mesh layout for 263 dec mp3 dec 49

4.12 5 port topology for 263 dec mp3 dec 49

4.13 4 port topology for 263 dec mp3 dec 49

4.14 263 enc mp3 dec:Communication Trace Graph 50

4.15 Custom layout for 263 enc mp3 dec 50

4.16 Mesh layout for 263 enc mp3 dec 50

4.17 5 port topology for 263 enc mp3 dec 50

4.18 4 port topology for 263 enc mp3 dec 50

4.19 mp3 enc mp3 dec:Communication Trace Graph 51

4.20 Custom layout for mp3 enc mp3 dec 51

4.21 Mesh layout for mp3 enc mp3 dec 51

4.22 5 port topology for mp3 enc mp3 dec 51

4.23 4 port topology for mp3 enc mp3 dec 51

5.1 ANOC design flow 55

5.2 CTG, floorplan and channel intersection graph 55

5.3 Recursive partitioning link placement and trace mapping 57

5.4 Recursive cuts on the CTG 57

5.5 Topology generation and router merging 60

5.6 Router merging 61

5.7 Interconnection network generation 63

5.8 Power comparison between MM and MA_m 66

5.9 Router comparison between MM and MA_m 66

5.10 Power comparison between PM and PA_m 66

5.11 Router comparison between PM and PA_m 66

5.12 Power comparison between MA_m and MA_{nm} 67

5.13 Router comparison between MA_m and MA_{nm} 67

5.14 Power comparison between PA_m and PA_{nm} 67

5.15 Router comparison between PA_m and PA_{nm} 67

5.16 Power comparison between ANOC, Mesh and QNoC for MILP floorplan 68

5.17 Router comparison between ANOC, Mesh and QNoC for MILP floorplan 68

5.18 Power comparison between ANOC, Mesh and QNoC for Parquet floorplan 68

5.19 Router comparison between ANOC, Mesh and QNoC for Parquet floorplan 68

5.20 MPEG4 CTG 70

5.21 MA_m MPEG4 topology 70

5.22 PA_m MPEG4 topology 70

5.23 MWD CTG 70

5.24 MA_m MWD topology 70

5.25 PA_m MWD topology 70

5.26 CTG for set-top box application 70

5.27 Floorplan and interconnection network architecture 70

6.1 Core alignments and flow graphs 74

6.2 Topology design 79

6.3 Proof of supermodularity 82

Figure Page

6.4 Routing with shortest path and minimum routers 84

6.5 Construction of SPG 87

6.6 Logical versus physical routing 87

6.7 Resolving infeasible solutions due to port constraints 92

6.8 Router merging 92

6.9 Power and router comparisons with no port and no link constraints 98

6.10 Power and router comparisons with no port constraint but with link constraints . . 98

6.11 Power and router comparisons with port constraints but no link constraints. ANOC violated port constraints for all benchmarks and is not plotted 98

6.12 Power and router comparisons with power and router constraints. ANOC violated port constraints for all benchmarks and is not plotted 98

6.13 Minimizing power consumption by introducing additional routers 99

6.14 Floorplan and NoC topology for Set-top box 100

6.15 Floorplan and NoC topology for NP 101

7.1 Application specific NoC design flow 104

7.2 Example NoC design flow 105

7.3 Hierarchical representation 107

7.4 Array based data-structure 108

7.5 Communication trace mapping 113

7.6 Pseudo code for MSP algorithm 114

7.7 Pareto points for NoC generation 115

7.8 GA for custom NoC design 118

7.9 Trace level crossover 119

7.10 Node level crossover 120

7.11 Router level crossover 120

7.12 Router architecture 125

7.13 Breaking deadlocks by adding virtual channels 126

7.14 Comparison between $GA_{6,5}$, $ILP_{6,5}$ and $ANOC_{6,5}$ corresponding to minimum power consumption. ANOC caused port violations for all benchmarks 131

7.15 Comparison between $GA_{6,5}$, $ILP_{6,5}$ and $ANOC_{6,5}$ corresponding to solution with minimum router resource consumption. ANOC caused port violations for all benchmarks 131

7.16 Comparison between $GA_{6,\infty}$, $ILP_{6,\infty}$ and $ANOC_{6,\infty}$ corresponding to minimum power consumption 132

7.17 Comparison between $GA_{6,\infty}$, $ILP_{6,\infty}$ and $ANOC_{6,\infty}$ corresponding to solution with minimum router resource consumption 132

7.18 Comparison between $GA_{\infty,5}$, $ILP_{\infty,5}$ and $ANOC_{\infty,5}$ corresponding to minimum power consumption. ANOC caused port violations for all benchmarks 133

7.19 Comparison between $GA_{\infty,5}$, $ILP_{\infty,5}$ and $ANOC_{\infty,5}$ corresponding to solution with minimum router resource consumption. ANOC caused port violations for all benchmarks 133

7.20 Comparison between $GA_{\infty,\infty}$, $ILP_{\infty,\infty}$ and $ANOC_{\infty,\infty}$ corresponding to minimum power consumption 133

Figure Page

7.21 Comparison between $GA_{\infty,\infty}$, $ILP_{\infty,\infty}$ and $ANOC_{\infty,\infty}$ corresponding to solution with minimum router resource consumption 133

7.22 Minimizing power consumption by introducing additional routers 134

7.23 Our approach versus existing approach 136

7.24 Comparing GA_{pre} and GA_{post} for $GA_{6,5}$. Existing approach caused link length violations for benchmarks 1, 5 and 10. 136

7.25 Comparing GA_{pre} and GA_{post} for $GA_{6,\infty}$. Existing approach caused link length violations for benchmark 2. 136

7.26 Comparing GA_{pre} and GA_{post} for $GA_{\infty,5}$ 137

7.27 Comparing GA_{pre} and GA_{post} for $GA_{\infty,\infty}$ 137

7.28 CTG for JPEG encoder 139

7.29 Node description for set-top box, NP, and JPEG 139

7.30 Floorplan and NoC topology for set-top box 140

7.31 Floorplan and NoC topology for NP 141

7.32 Floorplan and NoC topology for JPEG encoder 142

7.33 Pareto curve for the JPEG, NP and set-top box applications 142

7.34 Power consumption across generations 142

7.35 Number of illegal solutions across generations 142

7.36 JPEG process graph 143

7.37 Latency for JPEG encoder 143

8.1 Motivating Example 149

8.2 Comparison of DVS policies: period constraint greater than task run time 165

8.3 Comparison of DVS policies: period constraint close to task run time 166

8.4 Heuristic Technique for System-level Low Power Design 176

8.5 Deterministic Optimization for System-level Low Power Design 180

8.6 Simulated annealing based optimization for low power design 183

8.7 MPEG decoder task flow 185

8.8 JPEG decoder task flow 185

8.9 MP3 encoder task flow 186

8.10 Runtimes of the MILP formulations 191

8.11 Relative power consumption of the MILP formulations 191

8.12 Runtimes of the heuristic techniques 193

A.1 JPEG: Results for $MILP$ 200

A.2 JPEG: Results for $MILP_{so}$ 201

A.3 JPEG: Results for the $MILP_{lat}$ 201

A.4 JPEG: Results for $MILP_{ls}$ 202

A.5 MPEG: Results for $MILP$ 202

A.6 MPEG: Results for the $MILP_{so}$ 203

A.7 MPEG: Results for $MILP_{lat}$ 203

A.8 MPEG: Results for $MILP_{ls}$ 203

A.9 MP3: Results for $MILP$ 204

A.10 MP3: Results for the $MILP_{so}$ 204

A.11 MP3: Results for $MILP_{lat}$ 205

A.12 MP3: Results for $MILP_{ls}$ 205

A.13 Heuristics: Results for JPEG decoding 206
A.14 Heuristics: Results for MPEG-1 decoding 206
A.15 Heuristics: Results for MP3 encoding 207
A.16 Single synthetic task graphs 207
A.17 Multiple synthetic task graphs 208

CHAPTER 1

INTRODUCTION

1.1. Motivation

Increased power densities, high performance requirements, and short design turn-around times have prompted designers to integrate multiple processor cores and memory elements on the same chip, known as multiprocessor System-on-Chip (MPSoC). Many of these architectures are targeted toward custom applications such as multimedia set-top box and have come to be known as application specific SoC. In the future, these architectures will be composed of hundreds of cores, will be implemented in $50nm$ technology or lower, and will be clocked in multi-GHz range.

High performance requirements, and low power consumption constraints will be two important design challenges in future SoC architectures. High clock speeds and increasing clock skews will make synchronous communication undesirable, or even infeasible. SoC designers will aggressively apply system-level power minimization techniques namely, dynamic voltage scaling (DVS), and dynamic power management (DPM) for power minimization. The requirement for timing closure, and the need for optimizing the architecture for power and performance will move design techniques toward partitioning the system into multiple voltage and clock islands. These islands will be locally clocked, and communicate with each other in an asynchronous manner. Architectures supporting this type of communication are known as globally asynchronous and locally synchronous (GALS) architectures [3]. GALS based communication provides the flexibility required by the designer to optimize the architecture for power and performance. The voltage and frequency of each island can be locally optimized to satisfy performance requirements, and minimize power consumption.

GALS based SoC designs in nanoscale technologies will also be faced with several communication challenges. Global signal delays will span multiple clock cycles [4] [5]. Signal integrity would also suffer due to increased RLC effects. The poor scalability of bus based communica-

المنارة للاستشارات

tion will make them undesirable for future multi-core architectures. As mentioned before, low power consumption under performance constraints will be a major design goal. Network-on-Chip (NoC) has been proposed as a solution to the global communication challenges for SoC design in nanoscale technologies [6] [7]. NoC supports asynchronous transfer of packets, and is a suitable communication paradigm in GALS based systems. Given a suitable topology, it can provide extremely high bandwidth by distributing the propagation delay across multiple switches, and thus pipelining the signal transmission. NoCs also offer advantages of supporting error control techniques, and supporting quality of service by distinguishing between different classes of traffic.

Figure 1.1 depicts the various steps involved in the design of an application specific SoC architecture. The SoC design problem can be broadly classified into two categories: i) computation architecture design, and ii) communication architecture design integrated with system-level physical design. The computation architecture design stage takes the application specification and performance constraints as input, and performs i) computation architecture synthesis, and ii) mapping and scheduling of the application on the processing cores. At the end of the computation architecture design stage, the application can be characterized, and communication patterns between the various cores of the architecture can be generated. The second step takes the communication traces as input, and generates a communication architecture that is optimized for a given design goal (for example, low power under performance constraints).

In this thesis, we address the power and performance optimization problems in the realm of GALS based application specific SoC architectures. We address the problems from computation, as well as the communication architecture design standpoints. At the computation architecture design stage, the architecture synthesis problem has been extensively investigated by several researchers [8] [9] [10]. On the other hand, low power mapping and scheduling is still an open problem. As mentioned before, SoC architecture design will move toward partitioning the system into voltage and clock islands. Optimized mapping and scheduling of the application to the

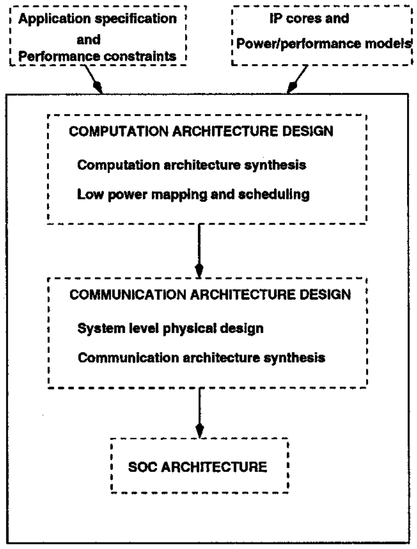

Application specification
and
Performance constraints
IP cores and
Power/performance models
COMPUTATION ARCHITECTURE DESIGN
Computation architecture synthesis
Low power mapping and scheduling
COMMUNICATION ARCHITECTURE DESIGN
System level physical design
Communication architecture synthesis
SOC ARCHITECTURE

Fig. 1.1: Application specific SoC architecture design

different voltage islands is an important step in minimizing the computation architecture power.

A NoC designer has to choose between utilizing a regular NoC architecture like a mesh or torus, or a custom architecture optimized for a given application. Regular architectures offer lower design time, and are useful when implemented in a generic multiprocessor environment such as the MIT RAW [11]. On the other hand, custom NoC architectures are designed with the given application in mind. For example, a NoC may be optimized for the MPEG-4 decoder application. In this thesis, we demonstrate that application specific architectures are superior to regular architectures in terms of power, performance and area.

In nanoscale technologies, link power will consume a considerable part of the total communication power [12]. In order to account for link power, system-level physical design must be integrated in the NoC design flow. NoCs designed for future SoC architectures will have to support the communication requirements between hundreds of cores. The complexity of designing a NoC that integrates system-level physical design, topology of the network, mapping of hundreds of cores and routing of communication traffic calls for efficient automated techniques for its optimization.

1.2. Summary of research contributions

The thesis addresses the NoC design and optimization, and system-level power minimization problems in the context of GALS based SoC architectures. The following sections summarize the contributions of the thesis toward each of the two problems.

1.2.1. NoC design and optimization

In this thesis, we defined the problem of designing custom NoCs that are optimized for a given application domain. This is in contrast to the existing approach by other researchers who assume that the NoC topology has a regular structure (such as a mesh). Through extensive experiments on multimedia and network processing benchmarks, we have demonstrated that the NoCs that are optimized for the given application domain are better than regular NoCs in terms of power, performance and area.

In the context of application specific NoC design techniques, we have proposed four techniques, each of which trade-off the quality of the solution with the time required to arrive at the final solution. The proposed techniques are:

1. *Integer Linear Programming (ILP) formulation, and clustering based heuristics (ICCD 04 [13], ISQED 06 [14], IEEE TVLSI 06 [12])*: We formulated the NoC design problem as an ILP formulation. The ILP formulation is extremely useful as it has the ability to generate optimal solutions. The optimal solution also serves as an excellent benchmark to which the solutions generated by other heuristics can be compared. However, the complexity of the ILP is known to be exponential in the number of inputs. Therefore, the time taken to arrive at the final solution explodes with increasing size of the problem. In order to alleviate the large solution times, we proposed a clustering based heuristic that divides the problem into

smaller clusters, solves the clusters optimally, and generates the final solution by merging the clusters together.

2. *Low complexity heuristic technique for NoC design (DATE 06 [15], ISPLED 05 [16], ISSS-CODES 06 [17], under review at ACM TODAES)*: We proposed a low complexity heuristic technique for custom NoC design that generates solutions for large problem sizes in negligible time. Due to its low complexity, the technique can be utilized as part of an iterative design space exploration for the design of application specific SoC architectures. We compared our heuristic with the optimal ILP based technique, and observed that the heuristic is able to generate solutions that are on average, within 1.25 of the optimal.

3. *Approximation algorithms for NoC design (ICCAD 05 [18], ASPDAC 07 [19])*: An approximation algorithm has the ability to generate solutions with known quality bound in polynomial time. Since the algorithm is polynomial time, it scales well with problem size. The approximation bound guarantees that the solution generated by the technique will never be worse than the bound. Approximation algorithms are immensely useful not only in the generation of good solutions, but also in serving as a benchmark for other techniques when the problem size is so large that the ILP based technique cannot be utilized. Our technique divides the NoC design problem into two phases, and generates solutions with quality bounds for each of the two phases.

4. *Genetic algorithm based heuristic for NoC design (VLSI Design 05 [20] ASPDAC 05 [21], Under second round review at IEEE TVLSI)*: A Genetic Algorithm (GA) models the problem as a set of genetic strings that undergo genetic operations namely, crossover and mutation and evolve over successive generations. We model the NoC design problem as a GA in three levels of hierarchy. The GA serves as a trade-off between the high runtimes of the ILP based technique, and lower solution quality of the low complexity heuristics. Moreover,

since the GA maintains several solutions at each generation, it has the ability to generate a set of Pareto points for multi-objective optimization problems. In the context of custom NoC design, the Pareto points provide the designer with an opportunity to select a solution that corresponds to the best trade-off between low power consumption, and corresponding number of routers in the NoC.

1.2.2. System-level low power optimization(VLSI Design 04: [1], ISVLSI 04 [2], Integration, The VLSI Journal (Accepted))

In this thesis, we present two techniques for low power integrated mapping and scheduling in multiprocessor architectures. The techniques are aimed at multimedia and network processing domains that are periodic, and allow the application of loop transformations such as pipelining and unrolling. Our technique applies these loop transformations to increase the throughput of the application, and trades-off the increase for power minimization by applying system-level low power techniques namely, Dynamic Voltage Scaling (DVS), and Dynamic Power Management (DPM).

We present two techniques for system-level low power minimization. The first technique is an ILP based technique that generates optimal solutions at high runtimes. We also propose several speedup techniques to reduce the runtime of the of the formulations, with a corresponding trade-off in quality. The second technique is an iterative search based heuristic technique that generates high quality solutions at a lower runtime.

1.3. Thesis organization

The remaining part of the thesis is broadly categorized into two parts. The first part, which is composed of chapters 2 through 7 defines the custom NoC design problem, and presents our solution techniques in detail. Chapter 8 defines and presents our proposed techniques for system-level low power optimization. Finally, Chapter 9 summarizes the thesis, and discusses future work in this field.

CHAPTER 2

APPLICATION SPECIFIC NOC DESIGN

2.1. Introduction

This chapter develops the necessary background for the NoC design problem. Automated techniques for NoC design require that a design flow is specified. The design flow in turn is governed by the application domain and interconnection architecture elements. The chapter is organized as follows. First, it presents a NoC design flow for nanoscale SoC design. This is followed by a characterization of the NoC elements to generate the power and performance models that are utilized by the automated techniques. Finally, we define the NoC design problem, and conclude the chapter.

2.2. NoC design flow

NoC architecture design is shown in Figure 2.1. The input to the communication architecture design problem is the computation architecture specification, characterized library of interconnection network components, and performance constraints. The computation architecture consists of processing and memory elements shown by rectangular blocks labeled as “P/M” in the top of the figure. Each ”P/M” block is uniquely identified by a node number ”n *i*” as denoted within each rectangle. The physical dimensions of the blocks are also specified as part of the inputs.

The directed edges between any two blocks represent the communication traces. The communication traces are annotated as “C*m*(B,L)” where ‘*m*” represents the trace number, “B” represents the bandwidth requirement, and “L” is the latency constraint. The bandwidth and latency requirements of a communication trace can be obtained by profiling the system-level specification in the context of overall application performance requirements. Applications in multimedia and network processing domains demonstrate well defined periodic communication characteristics and hence, can be easily modeled in the trace graph.

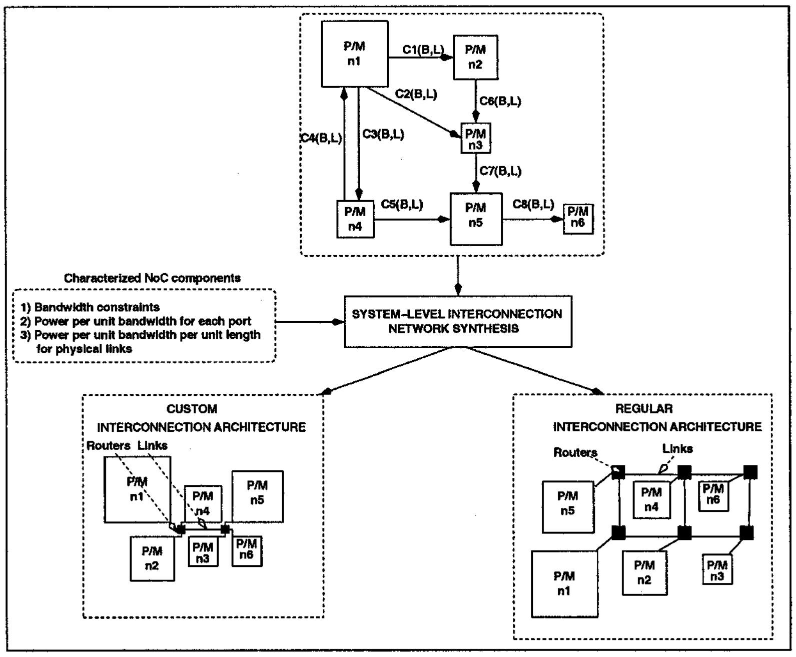

P/M
n1
C1(B,L)
P/M
n2
C2(B,L)
C6(B,L)
C4(B,L)
C3(B,L)
P/M
n3
C7(B,L)
P/M
n4
C5(B,L)
P/M
n5
C8(B,L)
P/M
n6
Characterized NoC components
1) Bandwidth constraints
2) Power per unit bandwidth for each port
3) Power per unit bandwidth per unit length for physical links
SYSTEM-LEVEL INTERCONNECTION NETWORK SYNTHESIS
CUSTOM INTERCONNECTION ARCHITECTURE
Routers
Links
P/M n1
P/M n4
P/M n5
P/M n2
P/M n3
P/M n6
REGULAR INTERCONNECTION ARCHITECTURE
Routers
Links
P/M n5
P/M n4
P/M n6
P/M n1
P/M n2
P/M n3

Fig. 2.1: NoC architecture design

The characterized library of interconnection architecture components is depicted on the left hand side of the figure. In nanoscale technologies, minimizing power consumption is a first order design goal along with performance maximization. Therefore, the components are characterized for both performance and power consumption. Each router architecture is characterized as follows:

- the peak bandwidth supported at input or output ports, and
- the power consumed to transfer data across the ports.

The power consumption in a port is a function of the total traffic flowing through it. Hence, the port is characterized by power consumed per unit bandwidth of traffic.

Figure 2.2 plots the power consumption of various components of the NoC for 100nm and 180nm technologies. In the figure, the X axis denotes the various components, and the Y axis denotes the corresponding dynamic energy consumption. We can infer from the plot that in nanoscale NoC architectures, the physical link will consume upwards of 30% of the total com-

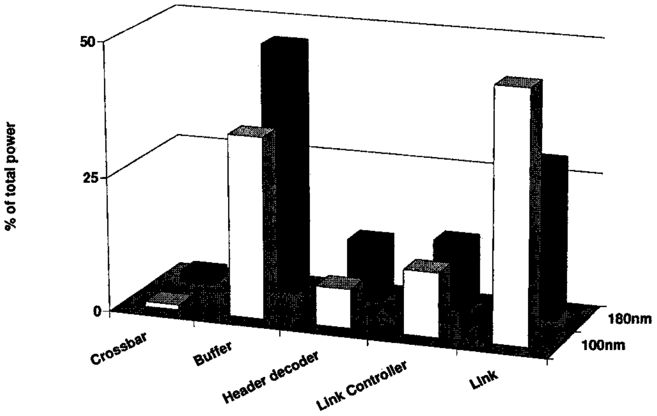


Fig. 2.2: Power consumption of NoC

munication energy. The energy consumption of the physical links is dependent on its length, and the bandwidth of traffic flowing through it. The length of the link in turn is determined by the system-level floorplan. Therefore, the design of the NoC architecture must jointly address system-level floorplan and interconnection network generation. The power consumption in the physical link is a function of the bandwidth of data flowing through the link, and the length of the link. Therefore, the physical links are characterized by power consumed per unit bandwidth per unit length.

The output of the communication architecture design problem is a system-level floorplan of the final design, topology of the network, and static routing of the communication traces on the network such that the performance constraints are satisfied, and the power consumption is minimized. As shown in the bottom part of Figure 2.1, the NoC topology can either be pre-defined as shown on the right side, or it can be optimized for the application, as shown on the left.

In this thesis, we address the design of a NoC in the context of application specific SoC architecture. Application specific SoC design offers the opportunity for incorporating custom NoC architectures that are optimized for the target problem domain, and do not necessarily conform to regular topologies. Regular topologies are suitable for general purpose architectures such as the MIT RAW [11] that include homogeneous cores. Application specific SoC architectures consist of heterogeneous cores and memory elements which have vastly different sizes. For such architectures we demonstrate that the custom NoC architecture is superior to regular architecture in terms of power and area consumption under identical performance requirements.

There are a number of limitations of the regular interconnection architecture. It assumes that all cores are of the same size, which is not the case for custom SoC. Therefore, even if the system-level topology is regular, it does not remain regular after the final floorplanning stage. The alternative option of regular layout results in large amount of area overhead. The regular topology assumes that every core has equal communication bandwidth with every other core. This does not hold in realistic applications. Regular topologies are known to support design reuse. In custom topologies, the router architecture itself is regular and can be easily parameterized (on number of ports, width of physical links, number of virtual channels and so on). In other words, custom NoC architectures also support design reuse. Therefore, this work concentrates on custom NoC topologies that are optimized for the target application.

2.3. Router Architecture characterization

Figure 2.3 shows the top-level (left hand side) and detailed (right hand side) architecture of a 5-port NoC router utilized in this work. The router consists of five unit routers that communicate with the neighboring routers, and with the processor. Unit routers inside a single router are connected through a 5x5 crossbar. Data is transferred across routers or between the processor and the corresponding router by a four phase asynchronous handshaking protocol. A single unit router, and corresponding input and output ports are highlighted in the right half of the figure.

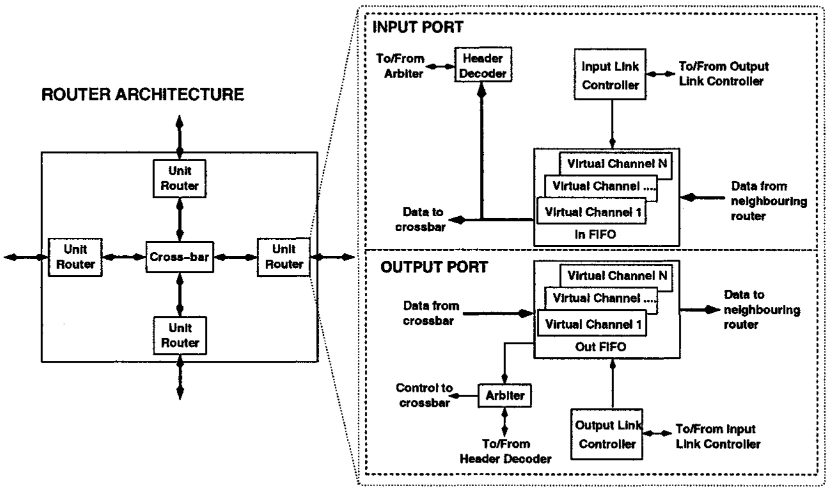

ROUTER ARCHITECTURE
Unit Router
Unit Router
Cross-bar
Unit Router
Unit Router
INPUT PORT
To/From Arbiter
Header Decoder
Input Link Controller
To/From Output Link Controller
Virtual Channel N
Virtual Channel ....
Virtual Channel 1
In FIFO
Data to crossbar
Data from neighbouring router
OUTPUT PORT
Data from crossbar
Virtual Channel N
Virtual Channel ....
Virtual Channel 1
Out FIFO
Data to neighbouring router
Control to crossbar
Arbiter
To/From Header Decoder
Output Link Controller
To/From Input Link Controller

Fig. 2.3: NoC router architecture

The unit router consists of input and output link controllers, input and output virtual channels, a header decoder, and an arbiter.

Data arrives at an input virtual channel of a unit router from either the previous router or the processor connected to the same router. The header decoder decodes the header flit of the packet after receiving data from the input virtual channel, decides the packet's destination direction, and sends a request to the arbiter of the unit router in the corresponding direction for access to the crossbar. In other words, the header decoder performs the task of routing the packet. We assume that the routing strategy is an application specific scheme and the header decoder decides the output port based on the communication trace identifier (ID). The communication trace ID uniquely identifies a particular data stream flowing from a source node to the destination. Once the grant is received, the header decoder starts sending data from the input to the output virtual channel through the crossbar.

We characterize the power consumption of the unit router in 100 nm technology with the help of a cycle accurate power and performance evaluator [22]. The cycle accurate simulator was constructed by characterizing the power and performance of each of the router sub-components and physical links through Hspice simulations. In the experiment the width of the physical links

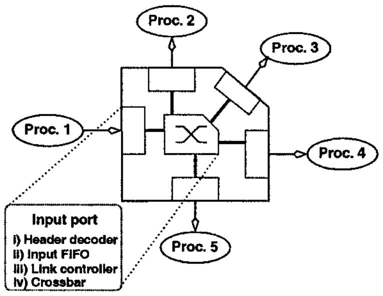

Proc. 2
Proc. 3
Proc. 1
Proc. 4
Proc. 5
Input port
i) Header decoder
ii) Input FIFO
iii) Link controller
iv) Crossbar

Fig. 2.4: Input port power consumption set-up

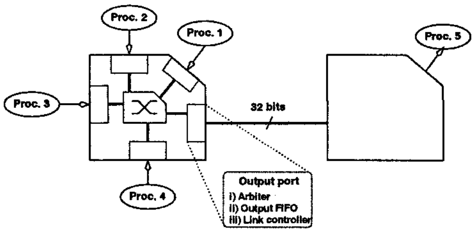

Proc. 2
Proc. 1
Proc. 5
Proc. 3
32 bits
Proc. 4
Output port
i) Arbiter
ii) Output FIFO
iii) Link controller

Fig. 2.5: Output port power consumption set-up

(consequently the width of input and output FIFO, and crossbar) is 32 bits, number of virtual channels is 2, depth of virtual channels is 4, and the number of flits in the packet is 8 (packet size 256 bits).

2.3.1. Power consumption in input port of unit router

The first experiment evaluated the power consumption of a port due to the traffic flowing into the unit router. The total power consumption due to traffic at a particular input port is the summation of the power consumed by the link controller, header decoder, input FIFO and crossbar. We considered a 5-port router with 5 processors attached to each port as shown in Figure 2.4. Processor 1 sends packets with random contents to the 4 other processors with a uniformly random distribution. Figure 2.6 plots the power consumption in the input port for the 4 components (link controller, header decoder, input FIFO, crossbar) for varying injection rate at processor 1. The delay for a particular injection rate was uniformly distributed within the mean delay interval. As can be observed from the plot the power consumption in the input port varies linearly with the injection rate.

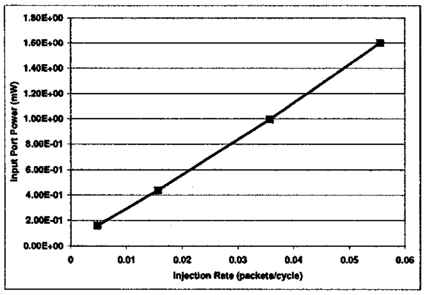


Fig. 2.6: Input port power consumption

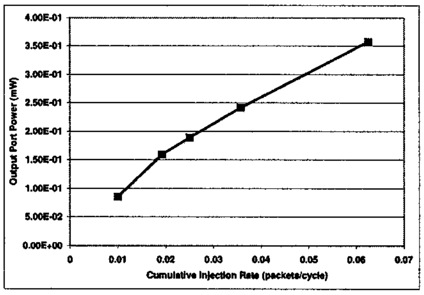


Fig. 2.7: Output port power consumption

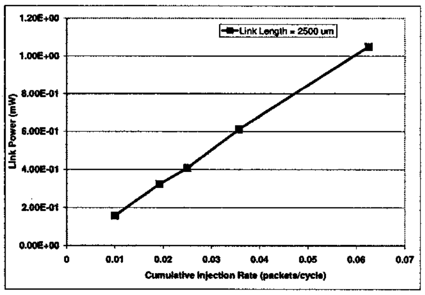


Fig. 2.8: Link power versus injection rate

2.3.2. Power consumption in output port of unit router

The second experiment evaluated the power consumed at a particular output port of a unit router due to traffic flowing in the outward direction to a neighboring router or core. We considered a 5-port router with processors (1-4) attached to four input ports as shown in Figure 2.5. All the four ports inject packets that traverse through the fifth port to processor 5 attached to the neighboring router. The contents of the packets where randomly generated, and the delay for a particular injection rate was also uniformly distributed within the mean delay interval. We evaluated the total power consumption of the arbiter, output FIFO, and link controller for varying cumulative injection rate from the four processors. Figure 2.7 plots the total power consumption (for the arbiter, output FIFO, link controller) versus the cumulative injection rate. The power consumption of the output port also varies linearly with the cumulative injection rate.

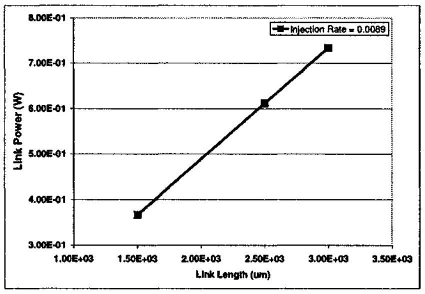


Fig. 2.9: Link power versus length

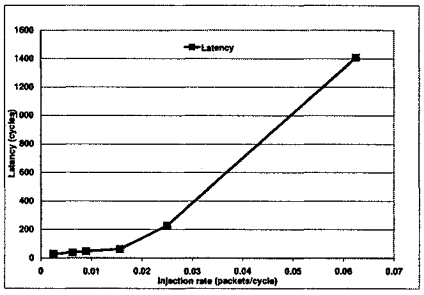


Fig. 2.10: Latency for 2 routers

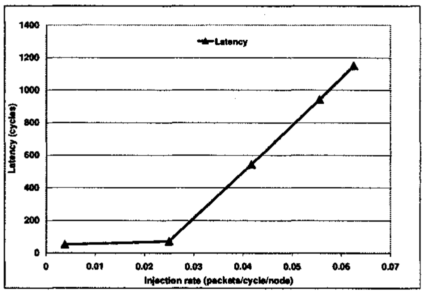


Fig. 2.11: Latency for 4x4 mesh

2.3.3. Power consumption in physical links

Figures 2.8 and 2.9 plot the variation in link power consumption versus injection rate (for a constant link length of 2.5 *mm*) and link length (for a constant injection rate 0.0089 packets/cycle), respectively. As can be observed from the figures the link power varies linearly with both injection rate and link length.

2.3.4. Latency

Figure 2.10 plots the average latency versus the injection rate for the packets in experiment 2 (Figure 2.5). Please note that the x-axis plots the injection rate due to one processor as opposed to the cumulative injection rate of the four processors as in Figure 2.7. As can be observed from the figure, the average latency remains constant until the output port gets congested. A similar trend is observed (see Figure 2.11) when we consider a 4 x 4 mesh architecture with 16 processors that are injecting to uniformly distributed random destinations. The average network latency remains constant until the network gets congested. The network congestion is marked by a sharp increase in average latency. Our synthesis technique prevents network congestion by static routing of the communication traces subject to the peak bandwidth constraint on the router ports. Since the network is always operated in the un-congested mode, we can represent the network latency constraint in terms of router hops (such as 1 or 2) instead of an absolute number (such as 100 cycles).

2.4. Problem Definition

Given:

- A directed communication trace graph $G(V, E)$, where each $v_i \in V$ denotes either a processing element or a memory unit (henceforth called a node), and the directed edge $e_k = \{v_i, v_j\} \in E$ denotes a communication trace from v_i to v_j. For every $v_i \in V$, the height and width of the core is denoted by $\mathcal{H}_i$ and $\mathcal{W}_i$, respectively.

- For every $e_k = \{v_i, v_j\} \in E$, $\omega(e_k)$ denotes the bandwidth requirement in bits per cycle, and $\sigma(e_k)$ denotes the latency constraint in hops.

- A router architecture, where η denotes the number of input/output ports of the router, and Ω denotes the peak input and output bandwidth that the router can support on any one port. Thus, each port of a router can support equal bandwidth in input and output modes. Since a node $v \in V$ is attached to a port of a router, the bandwidth to any node from a router, and from any node to a router is less than Ω. Two quantities Ψ_i and Ψ_o that denote the power consumed per *Mbps* of traffic bandwidth flowing in the input and output direction, respectively for any port of the router.

- A physical link power model denoted by Ψ_l per $Mbps$ per mm.
- Two constants $\mathcal{H}$ and $\mathcal{W}$ that denote the height and width constraints on the overall dimensions of the system-level floorplan.
- A maximum allowable link length between routers to ensure a single clock cycle data transfer.

Let $\mathcal{R}$ denote the set of routers utilized in the synthesized architecture, E_r represent the set of links between two routers, and E_v represent the set of links between routers and nodes. The objective of the NoC design problem is to:

- generate a system-level floorplan and
- a network topology $T(\mathcal{R}, V, E_r, E_v)$,

such that:

- for every $e_k = (v_i, v_j) \in E$, $\exists$ a route $p = \{(v_i, r_i), (r_i, r_j), \ldots (r_k, v_j)\}$ in T,
- the bandwidth constraints on the ports of the routers are satisfied,
- the bounding box of the floorplan satisfies $\mathcal{H}$ and $\mathcal{W}$, and
- the total system-level power consumption for inter-core communication is minimized.

2.5. Conclusion

In this chapter, we defined a flow for NoC design in nanoscale technologies. We observed that due to the increased percentage contribution of link power consumption, and its dependence on the length of the link, system-level floorplanning must be incorporated in the flow. Based on the characterization of the router architecture, we developed power and performance models that are utilized by the automated design tools. In the following chapters, we first present a literature survey of the existing NoC design techniques, and present our automated techniques for NoC design.

CHAPTER 3

LITERATURE SURVEY

3.1. Introduction

In recent years a number of researchers have proposed architectures, performance evaluation techniques and optimization approaches for NoC. This chapter classifies and presents the existing research under four categories: seminal work, router architectures, performance models, and automated optimization approaches. Our work in this thesis falls in the category of automated optimization approaches. In the following sections, we discuss previous work in the first three categories, and then compare and contrast our work with existing research in the fourth category.

3.2. Seminal work

Guerrier et al. [23] presented a NoC design called SPIN that was based on fat-tree topology. They also presented the router architecture and cycle accurate performance model for their NoC design. Hemani et al. [24] presented the NoC architecture as a solution to the challenge of billion transistor ASIC design in the nanoscale era. They proposed a NoC design methodology and supported it with a case study on a hypothetical architecture. Sgroi et al. [25] discussed a platform based SoC design methodology that proposed the inclusion of NoC for supporting on-chip communication. Dally et al. [6] demonstrated the feasibility of the NoC and estimated that the NoC places an area overhead of 6.6%. Benini et al. [7] in their conceptual paper on NoC, predict that packet switched on-chip interconnection networks will be essential to address the complexity of future SoC designs. Kumar et al. [26] presented a conceptual system-level architecture that allowed a mesh-based NoC to accommodate large resources such as memory banks, FPGA areas, or high performance multi-processors. Taylor et al. [11] designed and fabricated a 4x4 mesh based NoC architecture as part of MIT RAW processor. In this thesis, we build on the research cited above and present linear programming based techniques for automated synthesis of custom NoC architectures.

3.3. NoC architectures

Several researchers have proposed architectures, and related optimizations for on-chip interconnection networks. We classify the related research on NoC architectures based on the supported levels of traffic service classes, error control schemes, and power optimizations.

3.3.1. Architectures for best effort traffic

In this paragraph we review the NoC architectures that support only best effort traffic class. SPIN [23] [27] [28] was one of the seminal works to propose a detailed NoC architecture built with fat tree topology. Proteo [29] [30] is a VSIA-compliant NoC architecture that can be configured for ring, star, and bus topologies. xpipes [31] is a parameterized router architecture that can be utilized in arbitrary NoC topologies. The xpipesCompiler proposed by Jalabert et al. [32] is a custom topology instantiation framework. However, it does not provide any support for NoC synthesis. Therefore, our work can be considered to be complementary to xpipesCompiler.

3.3.2. Architectures for guaranteed throughput traffic

Nostrum [33] [34] is a protocol stack for mesh based NoC architecture that supports both best effort and guaranteed throughput traffic classes. AEthereal [35] [36] is also a mesh based NoC architecture that supports guaranteed throughput traffic by utilizing a centralized scheduler for allocation of link bandwidth.

3.3.3. Architectures with error control schemes

Bertozzi et al. [37] presented power versus performance results for point-to-point error control in an on-chip bus protocol based on AMBA bus. Zimmer et al. [38] presented a fault model for NoC architecture. They also proposed a QoS scheme that treated control traffic with higher reliability than data traffic.

3.3.4. Architectural optimizations for low power

Worm et al. [39] proposed an adaptive low power transmission scheme for NoC that minimized the voltage swing and frequency subject to the workload requirement. Chen et al. [40] proposed a power-aware buffer policy that minimized the leakage power consumption in virtual channels. Simunic et al. [41] proposed a system-level power reduction scheme for SoC architectures with on-chip interconnection networks. Their scheme applied dynamic voltage management and dynamic voltage scaling policies based on both local and global workload information. Nilsson et al. [42] proposed a globally pseudochronous clocking scheme to reduce latency and power consumption in mesh based architectures.

Existing research on NoC architectures and their optimization concentrate on architectures that conform to a regular topology. Communication trace driven application specific NoCs need not conform to a regular topology. In this thesis, we present techniques to generate irregular topologies that minimize the communication power and area of the NoC.

3.4. Performance Evaluation

[43] [44] [45] [46] presented performance evaluation models for micro-interconnection networks that do not consider NoC architectures. Wassal et al. [47] proposed system-level performance and power models for a shared-memory internet protocol/asynchronous transfer mode switching fabric. Ye et al. [48] analyzed the power consumption in the switch fabrics of network

routers and proposed system-level models for the same. Pamunuwa et al. [49] performed a system level analysis and estimated the wiring overhead and the gate count for implementing mesh-based NoC architecture. Wang et al. [50] proposed a power-performance simulator for interconnection network called Orion. Bolotin et al. [51] proposed analytical models for system-level performance and cost estimation of NoC architectures.

3.5. Automated design techniques

Existing automated design techniques for regular topologies have primarily focused on mesh based NoC architectures. Hu et al. [52] proposed a branch and bound technique for core to router mapping on mesh based NoC architectures. Murali et al. [53] presented a heuristic technique called NMAP for core to router mapping, and routing in mesh based architectures. Ascia et al. [54] presented a genetic algorithm based technique for the same problem. Guz et al. [55] presented a technique for link capacity allocation for mesh based NoC architectures. Murali et al. [56] explored the problem of mapping cores on to mesh based NoCs where each core supports multiple applications. They sum the bandwidth requirements of different applications to generate a single application, and based on the single application, invoke an existing heuristic that maps cores onto the mesh. Angiolini et al. [57] compared the power and performance of mesh based NoC architectures with AMBA bus architectures. Steenhof et al. [58] predicted NoC as a solution to high-end consumer electronics systems, and presented a SoC with mesh based NoC for a TV system architecture. In contrast to the above cited research, our techniques generate application specific custom NoC architectures that are optimized for a given application domain.

In recent years, researchers have turned their focus on application specific NoC design. Benini et al. [59] presented a survey of design techniques for application specific NoC architectures. Pinto et al. [60] proposed a technique for synthesis of point to point links that utilize at most two routers between source and sink. Thus, their problem formulation does not address routing.

Jalabert et al. [32] proposed a custom NoC instantiation framework based on designer specified inputs. In other words, it does not synthesize the custom topology. Our work in [13] was the first attempt at synthesizing application specific NoC. We presented linear programming based techniques for custom NoC synthesis. In [20], we presented a genetic algorithm based iterative method that addresses the same problem. In [21], we extended the technique to incorporate traffic scheduling on router ports for guaranteed throughput traffic. Ogras et al. [61] proposed graph decomposition based heuristic techniques for application specific NoC architectures. Ogras et al. [62] also proposed heuristic incremental techniques that modified mesh based topologies via long link insertion. The papers cited above assume a constant predetermined link length, and hence do not incorporate system-level floorplanning to estimate their actual lengths. We observed that observed that the physical links are expected to consume upward of 30% of the NoC power consumption. Therefore, in order to account for link power, we incorporate system-level physical design in the NoC design flow. In [12], we proposed ILP based techniques that incorporated system-level floorplanning to account for physical link power consumption. Since then, we have proposed approximation algorithms [18], heuristic techniques [15], and iterative methods for application specific NoC design that incorporates system-level floorplanning.

CHAPTER 4

INTEGER LINEAR PROGRAMMING BASED TECHNIQUES FOR APPLICATION SPECIFIC NOC DESIGN

4.1. Introduction

In this chapter, we present our integer linear programming (ILP) based technique for NoC design. Please refer to Chapter 2 for the definition of the NoC design problem. Taking the application specification in the form of a communication trace graph (CTG), and a set of characterized interconnection architecture elements as input, our technique generates optimal NoC topologies by formulating the problem as a linear objective to be minimized, subject to a set of linear constraints. Our technique operates in two phases. In the first phase, it invokes a ILP based floorplanner to generate an initial layout of the SoC. The floorplan helps us determine the link lengths, which are required to account for link power consumption. In the second stage, we invoke our ILP formulation that maps cores to their respective routers, generates a topology for the NoC, and determines a route for each traffic trace such that the total communication power is minimized, subject to performance constraints.

The ILP formulation is of immense value due to its ability to generate optimal solutions. Moreover, the solutions generated by the ILP can be used as a benchmark to evaluate other heuristic techniques. However, the ILP formulation is constrained by exponential runtimes that affect its scalability. In order to alleviate the problem, we propose a clustering based heuristic that forms smaller clusters of the input, solves the clusters optimally, and generates the final solution by merging the solutions of the clusters together.

We experimented with several multimedia and network processing benchmarks, and compared the custom NoC designs generated by our technique with optimal mesh and QNoC based topologies. We found that the custom NoC performs far better than mesh based and QNoC based techniques in terms of power, performance and area.

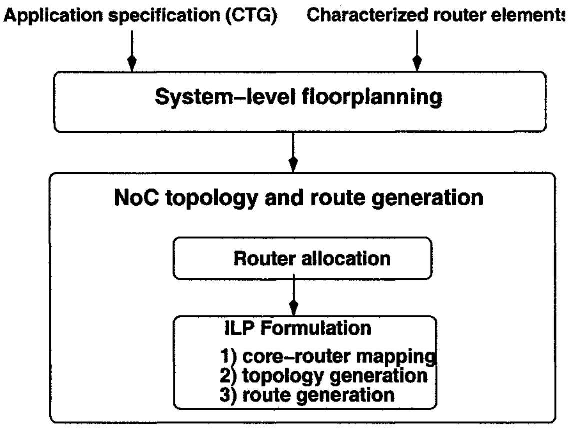

Application specification (CTG)
Characterized router elements
System–level floorplanning
NoC topology and route generation
Router allocation
ILP Formulation
1) core–router mapping
2) topology generation
3) route generation

Fig. 4.1: ILP based NoC generation

This chapter is organized as follows. In Section 4.2, we present the ILP formulations, in Section 4.3, we present a clustering heuristic to reduce the runtime of the formulation, in Section 4.4, we present our technique for addition of virtual channels to avoid deadlocks, and in Section 4.5, we present experimental results. Finally, in Section 4.6, we conclude the chapter.

4.2. ILP formulations

In this section we present the ILP formulations for solving the NoC synthesis problem. We address the problem by splitting it into two sub-problems: i) system-level floorplanning with an objective of minimizing the power consumption of the NoC subject to the layout constraints and ii) NoC topology and route generation again with an objective of minimizing the power consumption subject to the performance constraints. The overall flow of our technique is depicted in Figure 4.1. Our technique takes the application specification and characterized NoC elements and invokes a system-level floorplanner to generate an initial layout. This is followed by the interconnection architecture generation stage, in which routers are allocated to physical locations, and an ILP formulation is invoked to generate the NoC topology along with core to router mapping, and traffic routes for the traces.

4.2.1. System-level floorplanning

We present an NoC centric floorplanning formulation that minimizes communication power consumption by utilizing a unique cost function. At the floorplanning stage the power consumption due to the interconnection architecture can be abstracted as the power required to perform communication via point to point physical links between communicating cores. Although, such a cost function does not include the router power consumption, it is a true representation of the power consumption due to the physical links. However, inclusion of only the power consumption in the cost function ignores the performance requirements on the communication traces. Bandwidth constraints on the communication traces can be satisfied by finding alternative routes or adding more interconnection architecture resources. However, satisfying latency constraints is more difficult if the cores are placed wide apart. In addition to minimizing power and latency, it is also important that the layouts consume minimum area. Therefore, we specify our minimization goal as a linear combination of the power-latency function, and the area of the layout. Mathematically, we minimize

$$\alpha \cdot \left[\sum_{\forall e(u,v) \in E} dist(u,v) \cdot \Psi_l \cdot \frac{\omega(e)}{\sigma^2(e)} \right] + \beta \cdot [X_{max} + Y_{max}]$$

where $dist(u,v)$ is the distance between the cores u and v, α and β are constants, and X_{max} and Y_{max} represent the boundaries in positive X and Y directions, respectively. Note that minimizing X_{max} and Y_{max} minimizes the area that is given by $X_{max} \times Y_{max}$. The ILP is formulated such that the layout is obtained in the first quadrant. Thus, all the co-ordinates are greater than or equal to zero. The above optimization function gives a higher priority to latency constraint of a communication trace as opposed to the bandwidth. The values of the constants α and β determine the relative weight given to power minimization compared to area minimization and are specified by the designer. In the following section we describe the ILP formulation.

4.2.1.1. *Variables.* Independent variable: As mentioned before, we assume that the cores are placed in the first quadrant of the XY plane. For each core $v_i \in V$ let $(X_{i,min}, Y_{i,min})$ denote the lower left hand side co-ordinate of the placed core.

Dependent variables: The formulation utilizes the following derived variables:

- For each core $v_i \in V$ let $(X_{i,max}, Y_{i,max})$ denote the upper right hand side corner of the placed core. Thus,

$$X_{i,max} = X_{i,min} + \mathcal{W}_i, \quad Y_{i,max} = Y_{i,min} + \mathcal{H}_i$$

- For each pair of cores $v_i, v_j \in V$, let $\mathcal{DX}_{i,j}$ denote the difference between the x co-ordinates of the top right corner of the placed cores, and $\mathcal{DY}_{i,j}$ denote the corresponding difference between the y co-ordinates. Thus:

$$\mathcal{DX}_{i,j} = X_{i,max} - X_{j,max}, \quad \mathcal{DY}_{i,j} = Y_{i,max} - Y_{j,max}$$

Since these differences can be negative, $\mathcal{DX}_{i,j}$ and $\mathcal{DY}_{i,j}$ are un-restricted variables.

- For each pair of cores $v_i, v_j \in V$, let $\mathcal{X}_{i,j}$ and $\mathcal{X}'_{i,j}$ denote binary (0,1) variables that are given by:

$$\mathcal{X}_{i,j} = \begin{cases} 1 & \text{if } X_{i,min} \geq X_{j,max} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{X}'_{i,j} = \begin{cases} 1 & \text{if } X_{j,max} > X_{i,min} \\ 0 & \text{otherwise} \end{cases}$$

$\mathcal{X}_{i,j}$ and $\mathcal{X}'_{i,j}$ can be obtained by the following linear equations:

$$X_{i,min} - X_{j,max} - \mathcal{X}_{i,j} \cdot MAXVAL < 0$$

$$X_{j,max} - X_{i,min} - \mathcal{X}'_{i,j} \cdot MAXVAL \leq 0$$

$$\mathcal{X}_{i,j} + \mathcal{X}'_{i,j} = 1$$

where $MAXVAL$ is a very large integer.

- Let $\mathcal{Y}_{i,j}$ and $\mathcal{Y}'_{i,j}$ denote similar quantities along the y co-ordinates.

4.2.1.2. *Objective function.* The objective function for the floorplanning stage is to:

$$\text{Minimize} \quad (\mathcal{P} + \mathcal{A})$$

where

$$\mathcal{P} = \alpha \cdot \left(\sum_{\forall e(v_i,v_j) \in E} \Psi_l \cdot \frac{\omega(e)}{\sigma^2(e)} \cdot (|\mathcal{DX}_{i,j}| + |\mathcal{DY}_{i,j}|) \right)$$

$$\mathcal{A} = \beta \cdot [X_{max} + Y_{max}]$$

We model $|\mathcal{DX}_{i,j}|$ by introducing two variables $\mathcal{DX}^+_{i,j}$ and $\mathcal{DX}^-_{i,j}$. We define:

$$\mathcal{DX}^+_{i,j} - \mathcal{DX}^-_{i,j} = \mathcal{DX}_{i,j} \quad \text{and} \quad \mathcal{DX}^+_{i,j} + \mathcal{DX}^-_{i,j} = |\mathcal{DX}_{i,j}|$$

During minimization, the solver will set either $\mathcal{DX}^+_{i,j}$ or $\mathcal{DX}^-_{i,j}$ to zero, and the other to one. Similarly, we introduce $\mathcal{DY}^+_{i,j}$ and $\mathcal{DY}^-_{i,j}$ and define them as:

$$\mathcal{DY}^+_{i,j} - \mathcal{DY}^-_{i,j} = \mathcal{DY}_{i,j} \quad \text{and} \quad \mathcal{DY}^+_{i,j} + \mathcal{DY}^-_{i,j} = |\mathcal{DY}_{i,j}|$$

4.2.1.3. *Constraints.*

- Floorplanning requires that no two cores overlap when they are placed on the layout. Therefore, for each pair of cores $v_i, v_j \in V$ one of the following four conditions must be true:

  $$X_{i,min} \geq X_{j,max}, \quad X_{j,min} \geq X_{i,max}$$

  $$Y_{i,min} \geq Y_{j,max}, \quad Y_{j,min} \geq Y_{i,max}$$

  Therefore,

  $$\mathcal{DX}_{i,j} + \mathcal{DX}_{j,i} + \mathcal{DY}_{i,j} + \mathcal{DY}_{j,i} \geq 1$$

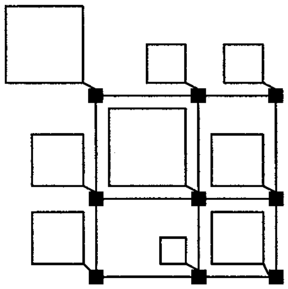


Fig. 4.2: Example mesh based floorplan

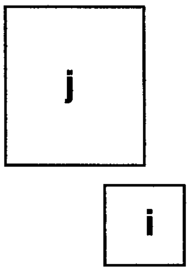


Fig. 4.3: Illegal layout in mesh based topology

- Apart from minimizing the power and area, the layout should satisfy the given aspect ratio constraints. Therefore,

$$Y_{max} \geq \gamma_{min} \times X_{max}$$

$$Y_{max} \leq \gamma_{max} \times X_{max}$$

- The layout should not violate the X and Y boundaries. Therefore, for each node $i \in V$,

$$X_{i,max} \leq X_{max}$$

$$Y_{i,max} \leq Y_{max}$$

4.2.1.4. *Additional constraints for mesh based topologies.* We compare and contrast the custom topologies generated by our techniques against mesh based topologies. An example of the mesh based layout is shown in Figure 4.2. The cores in the mesh based layout are aligned along a grid. The height and width of a row and column in the grid is determined by the largest core in the particular row or column, respectively. In the mesh based floorplan, in addition to the constraints described earlier for the generic layout, we require more constraints that avoid the illegal case shown in Figure 4.3. In Figure 4.3 the two cores are not aligned along a column, and therefore cannot be laid out on a grid.

For each pair of cores $v_i, v_j \in V$ we introduce pair of binary variables $\mathcal{GX}_{i,j}$ and $\mathcal{GY}_{i,j}$ defined as:

$$\mathcal{GX}_{i,j} = \begin{cases} 1 & \text{if } X_{i,max} > X_{j,max} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{GY}_{i,j} = \begin{cases} 1 & \text{if } Y_{i,max} > Y_{j,max} \\ 0 & \text{otherwise} \end{cases}$$

$\mathcal{GX}_{i,j}$ and $\mathcal{GY}_{i,j}$ can be obtained by similar linear equations as those defined for $\mathcal{X}_{i,j}$.

The various cores will be aligned along a grid if the following equations are satisfied for each pair of cores $v_i, v_j \in V$:

$$\mathcal{GX}_{i,j} + \mathcal{X}'_{i,j} \leq 1$$

$$\mathcal{GY}_{i,j} + \mathcal{Y}'_{i,j} \leq 1$$

4.2.2. Custom interconnection topology and route generation

The power consumption of the NoC is dependent upon the length of the physical links in the architecture. We utilize the floorplan from the previous stage to select router locations, and thus determine inter router, and node to router distances. By intelligently determining router locations, we can reduce the size of the ILP formulation and thus, reduce its runtime.

Initially, we create a bounding box for each node. A bounding box is a rectangular enclosure of the node such that the bounding boxes of two adjacent nodes abut each other. For example, in Figure 4.4 (A), the bounding box of node 4 extends to the top boundary of node 3, and that of node 10 extends to the top boundary of node 12, and to the left boundary of node 9. In the figure, all other bounding boxes are the co-ordinates of the respective nodes. A channel intersection graph (CIG) is a graph in which the bounding boxes form the edges, and the intersection of two perpendicular boundaries forms a node.

We observe that the physical dimensions of the routers are much smaller than the sizes of the cores. This assumption is supported by Dally et al. [6] who observed that the entire NoC places an area overhead of 6.6% on the SoC architecture. Moreover, we obtained area estimates on the router architecture presented by Banerjee et al. [22] and observed that a five port router architecture with two virtual channels per port, FIFO depth of 16 and width of 32 consumes an area of 0.25 mm^2 in 65 nm technology. Therefore, we assume that the routers are placed at the nodes of the CIG of the layout. Some of the routers will be incorporated into the NoC topology and un-utilized routers will be eliminated at the final design stage.

Finally, we remove all redundant routers. We remove all routers that are:

- placed along the perimeter of the layout, and
- placed less than a specified distance apart.

The motivation for removing routers can be explained as follows. The routers in the perimeter are not likely to be utilized, and are redundant. Similarly, one of two closely placed routers is redundant as it is unlikely to be utilized in the topology.

The location and number of routers available for the second stage depends on the algorithm used to remove redundant routers. Our algorithm for removing routers is shown in Figure 4.5. First, the algorithm removes the routers along the perimeter of the floorplan. Figure 4.4 (C) depicts the stage when the routers along the perimeter are removed. After removing routers along the perimeter, the algorithm calls the initialization function that marks all the internal routers as free. In lines 3-5, the algorithm generates a list L for each router that specifies the routers that are less than the minimum distance from it. Line 6 of the algorithm sets the current router (cur_rtr) to be the router at the bottom left hand corner of the layout, which is the router at the bottom left hand corner of node 12 in Figure 4.4. The next line calls the function rem_close_rtrs that removes all routers that are in $L(cur\_rtr)$, marks cur_rtr as tagged, and hops to the closest

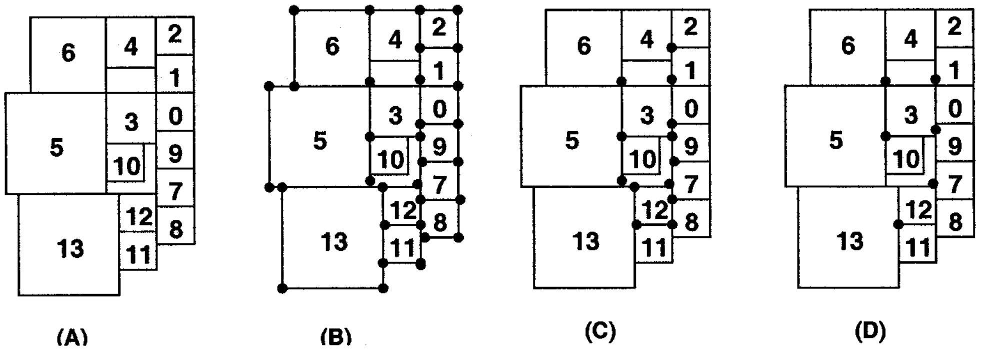


Fig. 4.4: Example of router allocation for custom topology

```
rem_redundant_rtrs()
    rem_perimeter_rtrs()
    initialize()
    for ($r \in \mathcal{R}$)
        $L(r)$ = get_close_rtr(r)
    end for
    cur_rtr = bottom_left_rtr()
    rem_close_rtrs(cur_rtr)
return
```

```
rem_close_rtrs(cur_rtr)
    for ($r \in L$(cur_rtr))
        set(r) = removed
    end for
    set(cur_rtr) = tagged
    next_rtr = get_next_rtr()
    if (next_rtr = NULL)
        return
    else
        rem_close_rtrs(next_rtr)
    end if
return
```

Fig. 4.5: Algorithm for removing redundant routers

available router (*next_rtr*) that is free (neither tagged nor removed). The function *rem_close_rtrs* recursively calls itself with *next_rtr* as parameter, until all routers are either tagged or removed. 4.4 (D) depicts the stage when routers are placed at more than a specified minimum distance apart.

Let $\mathcal{R}$ denote the total number of available routers. In the algorithm, each router is either tagged, or removed. A router cannot be both tagged and be removed. Therefore, the complexity of the algorithm is given by $O(|\mathcal{R}|)$.

We can further minimize the size of the formulation by limiting the number of routers that a node can be mapped to. Since the layout places communicating nodes close to each other, it is very unlikely that an optimal solution will have a node that is mapped to a router located at a large distance away from it. Therefore, for each node, we consider only those routers that are within a certain maximum distance from it. The distance is specified by the designer. Let $\mathcal{R}_i$ denote the set of routers available to node v_i.

Since we know the location of the routers, we can determine the shortest distance from a node v_i to the routers in $\mathcal{R}_i$. By the same argument, we can also determine all inter router distances.

The objective function of the formulation is minimization of the communication power. The power consumption in the NoC is the sum of the power consumed by the routers and the physical links. The power consumed by the routers is given by the product of the bandwidth of data flowing through the ports and the characterization function that specifies the power consumption per unit bandwidth. Similarly, the power consumption in a physical link is a product of the bandwidth of data flowing through the link, length of the link and the characterization function that specifies the power consumption per unit bandwidth per unit length.

In Section 4.2.2.1 we define the variables of the formulation, in Section 4.2.2.2 we state the objective function, and in Section 4.2.2.3, we present the constraints.

4.2.2.1. *Variables.* Base Variables: We define the following base (independent) variables.

- *Number of routers:* Let $r_i \in \mathcal{R}$, $0 \leq i \leq R_{max}$ denote a router. Each router in the NoC architecture is identical with the same number of ports "η", and peak bandwidth "Ω" per port. All ports are bidirectional.

- *Ports of the router:* Let $p_{i,j}$, $0 \leq j < \eta$, represent the j^{th} port of a router $r_i \in \mathcal{R}$.

- *Node-to-port mapping variables:* For each node $v_k \in V$, let $\mathcal{R}_k$ denote the set of routers that it can be mapped to. Let $\mathcal{NR}_{k,i,j}$ be a {0,1} variable that is 1 if node v_k is mapped to port $p_{i,j}$ of router $r_i \in \mathcal{R}_k$, otherwise 0. For each router $r_m \notin \mathcal{R}_k$, $\forall p_{m,j}$, $\mathcal{NR}_{k,m,j} = 0$

- *Port-to-port mapping variables:* For each port $p_{i,j}$ of router $r_i \in \mathcal{R}$, let $\mathcal{RR}_{i,j,k,l}$ be a {0,1} variable that is 1 if port $p_{i,j}$ of router $r_i \in \mathcal{R}$ is linked to port $p_{k,l}$ $(k \neq i)$ of router $r_k \in \mathcal{R}$, otherwise 0.

- *Variable for flow of traffic out of a port:* For each edge $\{v_i, v_j\} \in E$, let $\mathcal{O}_{i,j,k,l}$ be a {0,1} variable that is 1 if traffic from node v_i to node v_j flows out of port $p_{k,l}$, otherwise 0.

- *Variable for flow of traffic into a port:* For each edge $\{v_i, v_j\} \in E$, let $\mathcal{I}_{i,j,k,l}$ be a {0,1} variable that is 1 if traffic from node v_i to node v_j flows into port $p_{k,l}$, otherwise 0.

Variables $\mathcal{O}$ and $\mathcal{I}$ are utilized for modeling and satisfying the bandwidth and latency constraints on the various communication traces.

Derived Variables: We define the following derived variables.

- *Variable for the total traffic flowing out of a port:* Let $\mathcal{BO}_{k,l}$ be a variable that represents the total traffic flowing out of port $p_{k,l}$. $\mathcal{BO}$ can be derived as follows.

$$\mathcal{BO}_{k,l} = \sum_{\forall e_m=\{v_i,v_j\}\in E} \omega(e_m) * \mathcal{O}_{i,j,k,l}$$

- *Variable for the total traffic flowing into a port:* Let $\mathcal{BI}_{k,l}$ be a variable that represents the total traffic flowing into port $p_{k,l}$. $\mathcal{BI}$ can be derived as follows.

$$\mathcal{BI}_{k,l} = \sum_{\forall e_m=\{v_i,v_j\}\in E} \omega(e_m) * \mathcal{I}_{i,j,k,l}$$

- *Variable for flow of traffic on a link:* Let $\mathcal{Z}_{i,j,k,l,m,n}$ be a $\{0,1\}$ variable that is 1 if traffic (i,j) leaves port l of router k, and port l of router k is connected to port n of router m. Hence, $\mathcal{Z}_{i,j,k,l,m,n}$ can be represented as

$$\mathcal{Z}_{i,j,k,l,m,n} = \mathcal{O}_{i,j,k,l} \times \mathcal{RR}_{k,l,m,n}$$

The non-linear equation can be easily linearized by the following rule.

$$\mathcal{O}_{i,j,k,l} + \mathcal{RR}_{k,l,m,n} \geq 2 \times \mathcal{Z}_{i,j,k,l,m,n}$$

$$\mathcal{O}_{i,j,k,l} + \mathcal{RR}_{k,l,m,n} \leq \mathcal{Z}_{i,j,k,l,m,n} + 1$$

4.2.2.2. *Objective Function.* The objective is to minimize the power consumption of the NoC due to the cumulative traffic flowing through input and output ports, respectively of all the routers. The objective function can be expressed mathematically as follows:

$$Minimize\ (\mathcal{P}_R + \mathcal{P}_L)$$

where

$$\mathcal{P}_R = \Psi_i \cdot \sum_{\forall r_i \in \mathcal{R}} \sum_{\forall p_{i,j}} \mathcal{BI}_{i,j} + \Psi_o \cdot \sum_{\forall r_i \in \mathcal{R}} \sum_{\forall p_{i,j}} \mathcal{BO}_{i,j}$$

$$\begin{aligned}\mathcal{P}_L = \Psi_L \Bigg(& \sum_{i,j,k,l,m,n} \omega(i,j) \cdot \mathcal{RD}_{k,m} \cdot \mathcal{Z}_{i,j,k,l,m,n} + \\ & \sum_{i,j,k,l} \mathcal{ND}_{i,k} \cdot \omega(i,j) \cdot \mathcal{NR}_{i,k,l} + \\ & \sum_{i,j,k,l} \mathcal{ND}_{j,k} \cdot \omega(i,j) \cdot \mathcal{NR}_{j,k,l} \Bigg)\end{aligned}$$

where Ψ_i and Ψ_o are weights that denote the power consumed per *Mbps* of traffic flowing in the input and output directions, respectively, for any port of a router in the NoC, and Ψ_L is the link power per unit length per Mbps, $\mathcal{RD}_{k,m}$ denotes the distance between routers k and m, and $\mathcal{ND}_{i,k}$ denotes the distance of node i from router k.

4.2.2.3. *Constraints.* The following constraints are formulated.

- *Port capacity constraint:* The bandwidth usage of an input or output port should not exceed its capacity. Therefore,

$$\forall i \in \mathcal{R}, \forall p_{i,j}, \mathcal{BI}_{i,j} \leq \Omega, \mathcal{BO}_{i,j} \leq \Omega$$

- *Port-to-port mapping constraint:* A port can be mapped to one node, or to any one port that belongs to a different router:

$$\forall p_{i,j}, \sum_{\forall r_k \in \mathcal{R}, k \neq i} \sum_{\forall p_{k,l}} \mathcal{RR}_{k,l,i,j} + \sum_{\forall v_m \in V} \mathcal{NR}_{m,i,j} \leq 1$$
$$\forall p_{i,j}, \forall r_k \in \mathcal{R}, k \neq i, \mathcal{RR}_{k,l,i,j} = \mathcal{RR}_{i,j,k,l}$$

  The first constraint above is an inequality because it is possible that a port may not be mapped to any other port or node. The second equation models the symmetry of the variable $\mathcal{RR}$.

- *Node-to-port mapping constraint:* A node should be mapped exactly to one port. Therefore,

$$\forall v_i \in V, \sum_{\forall r_k \in \mathcal{R}_i} \sum_{\forall p_{k,l}} \mathcal{NR}_{i,k,l} = 1$$

- *Traffic routing constraints:* The traffic routing constraints discussed below ensure that for every $e_k = (v_i, v_j) \in E$, there exists a path $p = \{(v_i, r_i), (r_i, r_j), \ldots (r_k, v_j)\}$ in T.

  1. If a node is mapped to a port of a router, all traffic emanating from that node has to enter that port. Similarly, all traffic terminating at that node should leave from that port. Thus, for each router r_k, $\forall p_{k,l}$, and $\forall (v_i, v_j) \in E$, we require

$$\mathcal{I}_{i,j,k,l} \geq \mathcal{NR}_{i,k,l}, \ \mathcal{O}_{i,j,k,l} \geq \mathcal{NR}_{j,k,l}$$

  2. If a node is mapped to a port of a router, no traffic from any other node can either enter or leave that port. Thus, $\forall \{v_i, v_j\} \in E, \forall v_m \in V, m \neq i, m \neq j, \forall p_{k,l}:$

$$\mathcal{NR}_{m,k,l} + \mathcal{I}_{i,j,k,l} \leq 1, \ \ \mathcal{NR}_{m,k,l} + \mathcal{O}_{i,j,k,l} \leq 1$$

  3. If a traffic enters a port of the router, it should not enter from any other port of that router. Similarly, if a traffic leaves a port of a router, it should not leave from any

other port of that router. This constraint ensures that the traffic does not get split across multiple ports. Thus, for each router r_k, and $\forall(v_i, v_j) \in E$,

$$\sum_{\forall p_{k,l}} \mathcal{I}_{i,j,k,l} \leq 1, \sum_{\forall p_{k,l}} \mathcal{O}_{i,j,k,l} \leq 1$$

4. If a traffic enters a port of a router, it has to leave from exactly one of the other ports of that router. In the same way, if a traffic leaves a port of a router, it must have entered from exactly one of the other ports of that router. This constraint ensures the conservation of flow of traffic. Hence, for each router r_k, $\forall p_{k,l}$, and $\forall(v_i, v_j) \in E$,

$$\sum_{\forall p_{k,m}, m \neq l} \mathcal{O}_{i,j,k,m} \geq \mathcal{I}_{i,j,k,l}$$
$$\sum_{\forall p_{k,m}, m \neq l} \mathcal{I}_{i,j,k,m} \geq \mathcal{O}_{i,j,k,l}$$

5. If two ports of different routers are connected, traffic leaving from one port should enter the other, and vice versa. For example, if $p_{k,l}$ and $p_{m,n}$ are connected, $\mathcal{RR}_{k,l,m,n}$ will be 1. Therefore, a traffic $O_{i,j,m,n}$ leaving port n of router r_m should enter port l of router r_k. Therefore, $I_{i,j,k,l}$ should be set to 1. Similarly, if $I_{i,j,k,l} = 1$, $O_{i,j,m,n}$ should be set to 1. Therefore, for each pair of routers $\{r_k, r_m\}$, $k \neq m$, $\forall p_{k,l}, \forall p_{m,n}$ and, $\forall(v_i, v_j) \in E$,

$$\mathcal{RR}_{k,l,m,n} + \mathcal{I}_{i,j,k,l} - \mathcal{O}_{i,j,m,n} - 1 \leq 0$$
$$\mathcal{RR}_{k,l,m,n} - \mathcal{I}_{i,j,k,l} + \mathcal{O}_{i,j,m,n} - 1 \leq 0$$

6. If two ports of different routers are connected, a traffic can leave exactly one of the two ports. Similarly, a traffic can enter only one of the two ports. For example, if $p_{k,l}$ and $p_{m,n}$ are connected, for any traffic $(v_i, v_j) \in E$, $I_{i,j,m,n}$ and $I_{i,j,k,l}$ cannot be simultaneously 1. Similarly, $O_{i,j,m,n}$ and $O_{i,j,k,l}$ cannot be simultaneously 1. Thus, for

each pair of routers $\{r_k, r_m\}$, $k \neq m$, $\forall(v_i, v_j) \in E, \forall p_{k,l}, \forall p_{m,n}$

$$\mathcal{RR}_{k,l,m,n} + \mathcal{I}_{i,j,k,l} + \mathcal{I}_{i,j,m,n} - 2 \leq 0$$

$$\mathcal{RR}_{k,l,m,n} + \mathcal{O}_{i,j,k,l} + \mathcal{O}_{i,j,m,n} - 2 \leq 0$$

7. If a traffic enters a port of a router, that port must be mapped to a node or to a port of a different router. Therefore, if $\mathcal{I}_{i,j,k,l}$ is 1 for some traffic $(v_i, v_j) \in E$, some $\mathcal{NR}_{i,k,l}$ should be 1 or, some $\mathcal{RR}_{m,n,k,l}$ should be 1 where $p_{m,n}$ exists. Similarly, if a traffic leaves a port of a router, that port must be mapped to a node, or to a port of a different router. Therefore, if $\mathcal{O}_{j,i,k,l}$ is 1 for some traffic $\{v_j, v_i\} \in E$, some $\mathcal{NR}_{i,k,l}$ should be 1 or, some $\mathcal{RR}_{m,n,k,l}$ should be 1 where $p_{m,n}$ exists. The constraints can be modeled as follows. For each router r_k, $\forall p_{k,l}$, and $\forall\{v_j, v_i\} \in E$

$$\mathcal{NR}_{j,k,l} + \sum_{\forall r_m \in \mathcal{R}} \sum_{\forall p_{m,n}} \mathcal{RR}_{k,l,m,n} \geq \mathcal{I}_{j,i,k,l}$$
$$\mathcal{NR}_{i,k,l} + \sum_{\forall r_m \in \mathcal{R}} \sum_{\forall p_{m,n}} \mathcal{RR}_{k,l,m,n} \geq \mathcal{O}_{j,i,k,l}$$

- *Latency constraint:* The latency constraint refers to the maximum number of hops that is allowed to route the traffic from a source node to a sink node. For example, a latency of 2 means that the traffic can pass through at most two routers. The latency constraint is modeled as follows:

$$\forall e_k = \{v_i, v_j\} \in E, \sum_{\forall r_k \mathcal{R}} \sum_{\forall p_{k,l}} \mathcal{O}_{i,j,k,l} \leq \sigma(e_k)$$

Latency constraint of 1 is a special case in which no router to router connections are allowed. Therefore, for latency constraint of 1, all previous constraints pertaining to router to router connections can be removed. The imposition of latency constraint affects the feasibility of a NoC architecture. Latency and the number of ports in the router architecture are related by the following lemma.

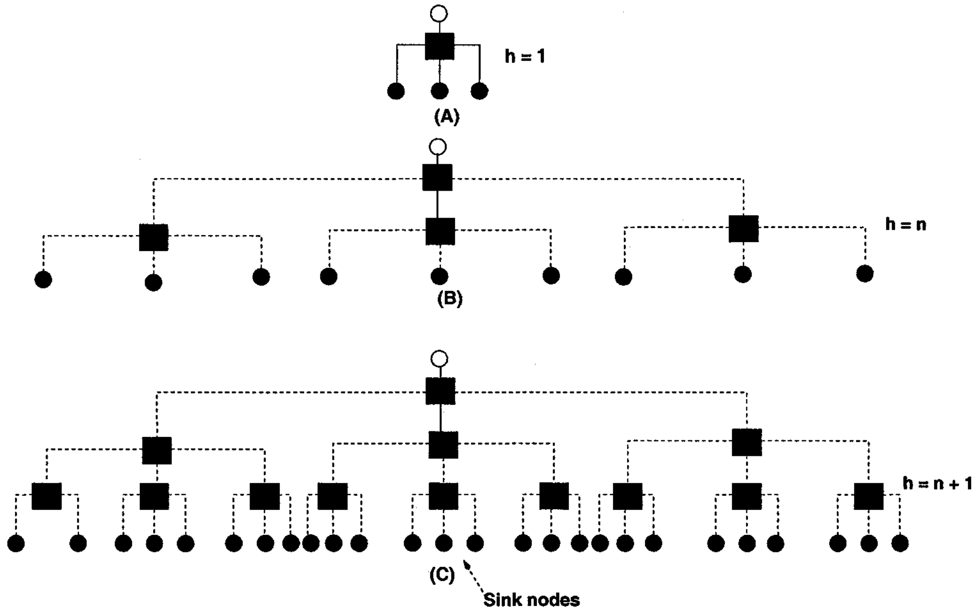


Fig. 4.6: Adding router to increase number of traffic

Lemma: If the router architecture has η ports per router, and σ is the maximum latency constraint on any edge, (i.e $\forall e_k \in E, \sigma(e_k) \leq \sigma$), a NoC topology is not possible if for any node, the total number of edges entering and leaving the node is more than $(\eta - 1)^{\sigma}$.

Proof: Without loss of generality, we will prove the lemma for multiple traffic traces originating from one source node, and ending at multiple respective sink nodes. As shown in Figure 4.6, the source node is denoted by an unfilled circle, the routers are denoted by filled square boxes, and the sink nodes are denoted by filled circles that form the leaves of the tree. We will prove our lemma by mathematical induction on σ. For simplicity, the proof assumes that bandwidth constraints are not violated.

Base case: Let $\sigma = 1$. In this case, all sink nodes have to be mapped to ports of the router to which the source node is mapped. As shown in Figure 4.6(A), the resulting architecture can be visualized as a η-ary tree with height 1. Since one port is taken by the source node, the maximum number of ports available for sink nodes is given by:

$$numtraces_1 = \eta - 1$$

Induction hypothesis: Suppose the assumption holds for $\sigma = \sigma_n$. Therefore, the maximum number of traces is given by:

$$numtraces_n = (\eta - 1)^{\sigma_n}$$

The resulting architecture is shown in Figure 4.6(B). The architecture is an η-ary tree of height σ_n, and the maximum number of traces is given by the number of leaf nodes in the tree ($(\eta - 1)^{\sigma_n}$).

Proof for $\sigma_n + 1$: An η-ary tree with height $\sigma_n + 1$ can be formed from an η-ary tree with height σ_n by introducing a router at each leaf node. As shown in Figure 4.6(C), introduction of a router at a leaf node is equivalent to routing traffic from source to the various sink nodes in $\sigma_n + 1$ hops. In a tree with height σ_n, each leaf node can be replaced by a router, thus increasing the number of leaf nodes in a tree with height $\sigma_n + 1$ by $\eta - 1$. Therefore, when all leaf nodes in the σ_n-height tree are replaced by routers, the maximum number of traces that can be mapped in the $\sigma_n + 1$ tree is given by:

$$numtraces_{n+1} = (\eta - 1) * (\eta - 1)^{\sigma_n} = (\eta - 1)^{\sigma_n + 1}$$

Q.E.D

4.3. Clustering based heuristic technique

The ILP formulation for interconnection topology and route generation is constrained by exponentially increasing solution times for large communication trace graphs. This section presents

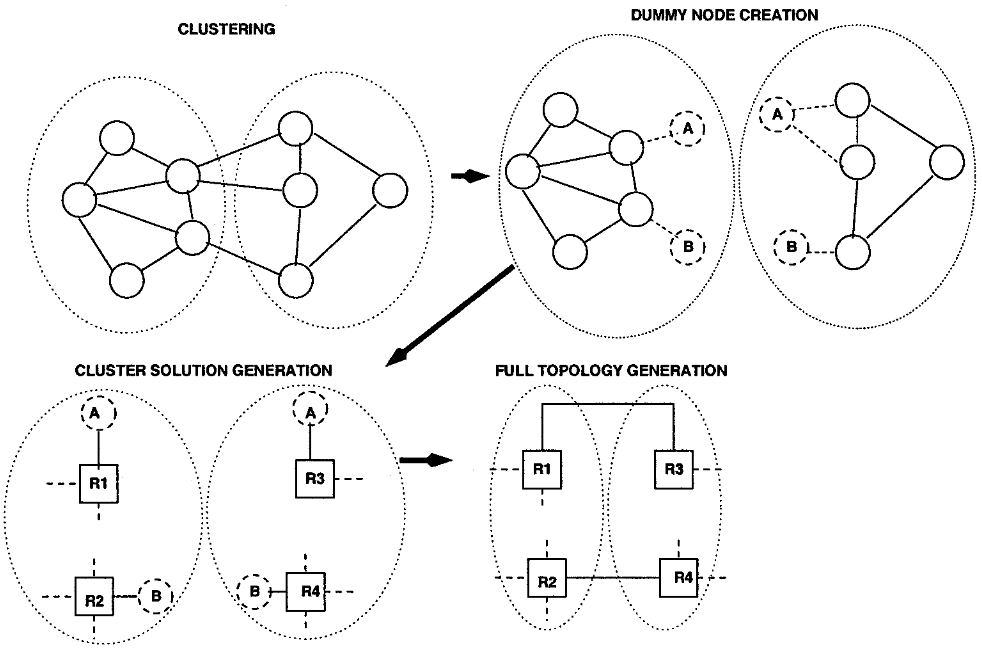


Fig. 4.7: Clustering based approach

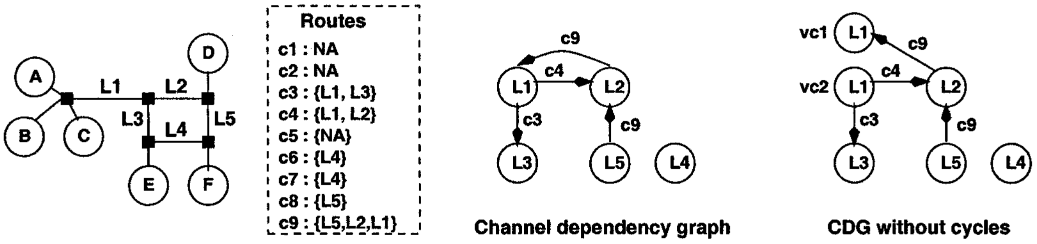
a clustering based heuristic technique for reducing the solution times. The clustering based heuristic technique is executed after the layout has been generated. The overall approach is shown in Figure 4.7.

The first stage is to form clusters of nodes. The sizes of the clusters are constrained by the maximum number of nodes in the clusters. This information is specified by the designer. We utilize an algorithm by Johnson et al. [63] [64] to form our clusters. For each edge $e \in E$, the clustering algorithm assigns a distance metric to the edge given by

$$\mathcal{DF}_e = \frac{\sigma_e^2}{\omega_e}$$

As discussed before, since it is more difficult to satisfy latency compared to bandwidth, we assign a higher weight to latency. Two communicating nodes that have low latency and high bandwidth are close to each other in terms of the distance metric, and are placed in the same cluster.

Once the clusters have been formed, for every communication trace that is cut across a



NoC topology and routes

Channel dependency graph

CDG without cycles

Fig. 4.8: Breaking deadlocks with additional virtual channels

cluster boundary, two dummy nodes are added to the respective clusters. If two edges share either a source or sink node, then only two dummy nodes are introduced instead of four. For example, in Figure 4.7 the top two edges that are cut share a common source in the left hand side cluster. Hence, only two dummy nodes (that are labeled as "A" in the figure) are introduced. The latency constraint on the original communication trace is split in half across the edges attached to the pair of dummy nodes. The bandwidth constraint is duplicated on the edges. The ILP formulation for topology design is then utilized to generate the partial solution for each cluster. In the figure, we assume that the routers have four ports, and are on the four sides of the rectangle. The full topology is generated from the partial solution by adding physical links between ports of routers that are in neighboring clusters, and are attached to identically named dummy nodes. For example, in Figure 4.7, routers R1 and R3 that are in different clusters are attached together with physical link since they both have a dummy node named "A" assigned to a port.

4.4. Deadlock avoidance

We note that the resulting topology and trace routes can lead to deadlocks in the NoC. Deadlocks are removed by introducing additional virtual channels in the routers [65] at design time. For deadlock avoidance, we first define the channel dependency graph (CDG) for the NoC. The NoC topology can be represented by a graph $G(R, L)$ where R denotes the set of routers, and L denotes the set of physical links, and a routing function $\mathcal{B}$. A CDG is a directed graph $G'(V', E')$, where each edge $l \in L$ has a corresponding unique node $v \in V'$, and there is an edge $e' \in E'$ for two adjacent links in $\{l1, l2\} \in L$ such that some trace is routed through $l1$ followed by $l2$. A routing technique is deadlock free if the CDG does not contain cycles [65]. Deadlocks are broken by introducing virtual channels at the deadlock causing router ports, such that the cycles in the CDG are broken.

Figure 4.8 depicts a NoC topology, and the corresponding CDG. For clarity, the physical dimensions of the cores are ignored in the NoC topology. The routing table denotes the links through which traffic traces are routed. A "NA" in the table denotes that the source and sink of the trace are connected to the same router, and hence, it is not routed through a physical link. The links of the NoC, and the corresponding nodes of the CDG are annotated by Lm, where m denotes the link number. The edges of the CDG are annotated by the trace that cause them. Deadlock can arise due to the cycle in the graph formed by $L1$ and $L2$. As shown in the right part of the figure, it is broken by introducing an additional virtual channel at router $r2$, introducing an additional copy of $L1$ in the CDG, and thus breaking the cycle. In the figure, vc1 and vc2 denote the two virtual channels required to break the deadlock.

With regards to the architecture level implementation, each trace is routed through only a fixed set of virtual channels at a router port, thus ensuring that no cycles occur in CDG. In our router architecture, the set of virtual channels through which a particular trace is routed is known. It is dependent upon the first virtual channel that the packet is injected into, by the source. Addition of an extra virtual channel to route the trace at an intermediate router can be easily specified by a minor modification in the header decoder of the router architecture. Therefore, addition of virtual channels ensures that deadlock is avoided.

Graph	Graph ID	Nodes	Edges
mp3 decoder	G1	5	3
263 encoder	G2	7	8
mp3 encoder	G3	8	9
263 decoder	G4	9	8
263 enc mp3 dec	G5	12	12
mp3 enc mp3 dec	G6	13	12
263 dec mp3 dec	G7	14	16
263 enc mp3 enc	G8	15	17
263 enc 263 dec	G9	16	16
263 dec mp3 enc	G10	17	17

Table 1: Graph Characteristics

4.5. Results

In this section, we present the results obtained by execution of our techniques on multimedia benchmark applications. In Section 4.5.1, we discuss the benchmark applications, in Section 4.5.2, we discuss the experimental setup, and in Section 4.5.3 we present and discuss the results.

4.5.1. Benchmark applications

We generated custom NoC architectures for four multimedia benchmarks namely, mp3 audio encoder, mp3 audio decoder, H.263 video encoder, and H.263 video decoder algorithms. In addition, we obtained results for six other benchmarks by mapping combinations of two applications from the above mentioned benchmarks simultaneously. The benchmarks are shown in Table 1. The communication trace graphs for the benchmarks were obtained from the work presented by Hu et al. [52].

4.5.2. Experimental setup

In our experimental setup, we obtained results for router architectures with 5 and 4 ports, respectively. The power consumption in 100 nm technology, for the input and output port was estimated to be $328nW/Mbps$ and $65.5nW/Mbps$, respectively. The link power consumption was estimated to be $79.6nW/Mbps/mm$. We utilized the Xpress-MP optimizer [66] to solve the ILP problems. The solver was configured with a timeout of 8 hours for the floorplanning stage, and

Node	263 dec mp3 dec	263 enc mp3 dec	mp3 enc mp3 dec
0	VLD	ME	FP
1	IQ	DCT	FFT
2	IDCT	FP	FILTER
3	MC	IDCT	MDCT
4	ADD	MC	ITER. ENC.1
5	MEM 1	VLE	ITER. ENC.2
6	MEM 2	MEM	BIT RES 1
7	HUFF 1	BIT RES 1	BIT RES 2
8	HUFF 2	BIT RES 2	BIT RES 3
9	BIT RES 1	IMDCT	BIT RES 4
10	BIT RES 2	SUM	IMDCT
11	IMDCT	BUF	SUM
12	SUM		BUF
13	BUF		

Table 2: Node descriptions

Graph	Area ratio (Mesh over Custom)	Runtime (sec)	
		Custom	Mesh
G1	1.26	< 1	< 1
G2	1.09	2	13
G3	1.21	17	56
G4	1.45	9	31
G5	1.74	507	9987
G6	1.56	13376	28800(t.o)
G7	1.39	2383	13746
G8	1.44	28800(t.o)	28800(t.o)
G9	1.36	28800(t.o)	28800(t.o)
G10	1.38	28800(t.o)	28800(t.o)

Table 3: Results for Floorplanning

No.	No. Ports	Graph	No of Clusters	Power (μW)			Routers			Runtime (secs)	
				ILP	CLUSTER	Ratio	ILP	CLUSTER	Ratio	ILP	CLUSTER
1	5	G1	1	2.622	2.622	1	1	1	1	< 1	< 1
2	5	G2	1	108.3	108.3	1	2	2	1	330	330
3	5	G3	1	5.7	5.7	1	2	2	1	1720	1720
4	5	G4	1	5.722	5.722	1	3	3	1	2318	2318
5	5	G5	2	179.5	110.9	0.61	5	4	0.8	41200 (t.o)	1103
6	5	G6	2	8.635	8.157	0.94	5	5	1	41200 (t.o)	2099
7	5	G7	2	11.91	8.535	0.71	5	5	1	41200 (t.o)	35522
8	5	G8	2	170.7	155.2	0.90	5	5	1	41200 (t.o)	1559
9	5	G9	2	245.4	115.6	0.47	7	5	0.7	41200 (t.o)	35467
10	5	G10	2	15.52	11.54	0.74	7	6	0.8	41200 (t.o)	1540
11	4	G1	1	2.631	2.631	1	2	2	1	< 1	< 1
12	4	G2	1	138.0	138.0	1	4	4	1	347	347
13	4	G3	1	5.944	5.944	1	3	3	1	1206	1206
14	4	G4	1	5.722	5.722	1	4	4	1	22222	22222
15	4	G5	2	194.6	140.7	0.72	5	5	1	41200(t.o)	2502
16	4	G6	2	10.94	12.47	1.12	6	6	1	41200(t.o)	41200(t.o)
17	4	G7	2	13.90	8.664	0.62	6	6	1	41200(t.o)	36733
18	4	G8	2	158.6	209.4	1.31	7	7	1	41200(t.o)	41200(t.o)
19	4	G9	2	241.3	147.2	0.61	7	7	1	41200(t.o)	41200(t.o)
20	4	G10	2	17.22	11.97	0.69	8	8	1	41200(t.o)	36544

Table 4: Comparison of ILP, and Clustering

No.	Graph	Power (μW)			Routers		
		Cluster	Mesh	Ratio	Cluster	Mesh	Ratio
1	G1	2.622	7.363	2.80	1	5	5
2	G2	108.3	291.4	2.69	2	7	3.5
3	G3	5.7	10.51	1.84	2	8	4
4	G4	5.722	12.51	2.18	3	9	3
5	G5	110.4	273.7	2.47	4	12	3
6	G6	8.157	18.02	2.21	5	13	2.6
7	G7	8.535	22.27	2.60	5	14	2.8
8	G8	155.2	277.0	1.78	5	15	3
9	G9	115.6	296.7	2.56	5	16	3.2
10	G10	11.54	28.63	2.15	6	17	2.8

Table 5: Comparison of Clustering, and Mesh

a timeout of 12 hours for the NoC architecture generation stage. If the solver failed to find the optimal solution, the best available solution generated within the timeout criterion was accepted. We obtained best results when:

- the minimum distance between two routers was set to one half the length of the maximum sized node,
- the distance of a node from the router to which it can be mapped was set to the length of the maximum sized node,

No.	Graph	Power (μW)			Routers		
		Cluster	QNoC	Ratio	Cluster	QNoC	Ratio
1	G1	2.622	2.627	1.00	1	2	2
2	G2	108.3	216.1	1.99	2	4	2
3	G3	5.7	6.368	1.11	2	4	2
4	G4	5.722	6.453	1.12	3	3	1
5	G5	110.4	244.9	2.21	4	6	1.5
6	G6	8.157	12.35	1.51	5	5	1
7	G7	8.535	22.37	2.62	5	6	1.2
8	G8	155.2	155.7	1.00	5	5	1
9	G9	115.6	352.4	3.04	5	7	1.4
10	G10	11.54	25.86	1.95	6	7	1.1

Table 6: Comparison of Clustering, and QNoC

No.	No. Ports	Graph	Lower bound (μW)		Cluster (μW) (C)	Ratio	
			ILP (ML)	Cluster (CL)		(C/ML)	(C/CL)
1	5	G1	2.622	2.622	2.622	1	1
2	5	G2	108.3	108.3	108.3	1	1
3	5	G3	5.7	5.7	5.7	1	1
4	5	G4	5.722	5.722	5.722	1	1
5	5	G5	93.26	110.9	110.9	1.18	1
6	5	G6	5.50	8.15	8.15	1.48	1
7	5	G7	8.10	8.53	8.535	1.05	1
8	5	G8	116.9	155.2	155.2	1.32	1
9	5	G9	90.74	115.6	115.6	1.27	1
10	5	G10	10.97	11.54	11.54	1.05	1
11	4	G1	2.631	2.631	2.631	1	1
12	4	G2	138.0	138.0	138.0	1	1
13	4	G3	5.944	5.944	5.944	1	1
14	4	G4	5.722	5.722	5.722	1	1
15	4	G5	121.4	140.7	140.7	1.15	1
16	4	G6	8.20	10.4	12.47	1.52	1.19
17	4	G7	6.88	8.66	8.664	1.25	1
18	4	G8	102.1	134.3	209.4	2.05	1.55
19	4	G9	80.33	113.7	147.2	1.83	1.29
20	4	G10	9.01	11.97	11.97	1.32	1

Table 7: Comparison of Clustering final solution with ILP and Clustering lower bounds

- the sizes of the clusters were limited to 9 nodes for the clustering based heuristic.

All results were obtained on a 950 MHz SPARC processor.

4.5.3. Results and discussion

In this section, we present and discuss the results for the floorplanning, and the topology generation and routing stages.

4.5.3.1. *Floorplanning stage.* Figures 4.9, 4.14, and 4.19 present the communication trace graphs for 263 dec-mp3 dec, 263 enc-mp3 dec, and mp3 enc-mp3 dec benchmarks, respectively.

The edges of the graphs are annotated with bandwidth requirement in *Kbps*. The node descriptions are depicted in Table 2. Figures 4.10, 4.15, and 4.20 present the corresponding floorplans obtained by executing our ILP based floorplanner on the benchmarks, for custom architectures. Figures 4.11, 4.16, and 4.21 present the corresponding floorplans for mesh architectures. The floorplanner places highly communicating nodes close to each other. For example, in Figure 4.10, nodes 1 and 2 that have high communication bandwidth, are placed next to each other.

Table 3 presents the ratio of the area consumption of mesh based topologies over that of custom topologies, and the runtimes of the floorplanning stage for custom and mesh topologies. On an average, the mesh topology consumed 1.38 times the area consumed by custom topology. Since the cores of the mesh are aligned in a grid, mesh topologies occupy more area compared to custom topologies. In the table, "t.o" denotes that the solver did not converge to an optimal solution within the timeout period of 8 hours. The longer running time for mesh may be attributed to the extra constraints required in the formulation to generate mesh topologies.

4.5.3.2. *Topology generation and routing stage.* Table 4 compares the results obtained for the ILP and clustering based techniques, respectively. In the table, column 1 gives the serial number, column 2 denotes the given router architecture, column 3 specifies the benchmark application, column 4 denotes the number of clusters for each benchmark, columns 5 and 6 present the total power consumption of solutions produced by ILP formulation, and the clustering technique respectively, column 7 gives the ratio of power consumption of the clustering technique solutions over the ILP solutions, columns 8 and 9 present the router requirements of ILP and clustering techniques, respectively, column 10 denotes the ratio of routers required by the clustering solutions over the ILP solutions, and finally columns 11 and 12 denote the run times of the ILP and clustering techniques, respectively. In the table, "t.o" denotes that the solver did not converge to an optimal solution within the timeout period of 12 hours, and the best available solution was accepted.

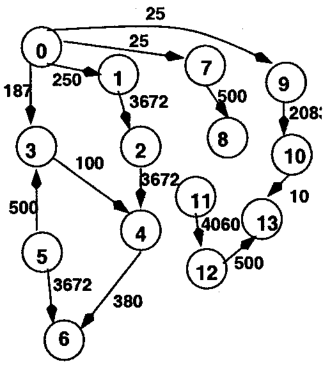

25
25
0
250
1
7
9
187
3672
500
208:
3
100
2
8
10
3672
11
10
500
4060
13
4
5
12
500
3672
380
6

Fig. 4.9: 263 dec mp3 dec: Communication Trace Graph

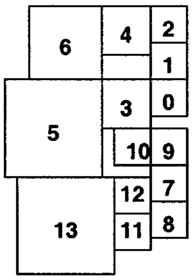

6
4
2
1
5
3
0
10
9
12
7
13
11
8

Fig. 4.10: Custom layout for 263 dec mp3 dec

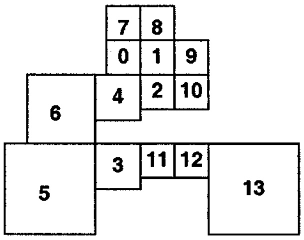

7
8
0
1
9
6
4
2
10
3
11
12
5
13

Fig. 4.11: Mesh layout for 263 dec mp3 dec

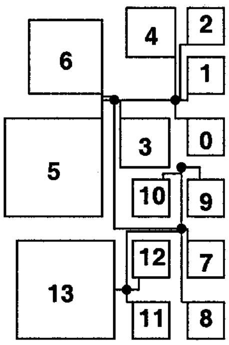

6
4
2
1
5
3
0
10
9
12
7
13
11
8

Fig. 4.12: 5 port topology for 263 dec mp3 dec

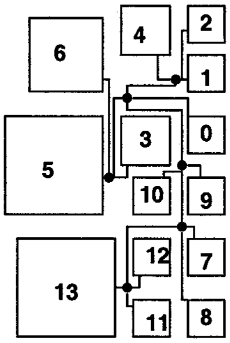

6
4
2
1
3
0
5
10
9
12
7
13
11
8

Fig. 4.13: 4 port topology for 263 dec mp3 dec

The clustering technique performed better than ILP for the timeout criterion of 12 hours. Due to its smaller size compared to the benchmark application, the solver was able to generate optimal solutions for the clusters. On the other hand, the solver timed out for many benchmarks when only the ILP was invoked. On an average, the clustering technique produced results that consumed only 85%, and 96% of the power and number of routers, respectively, in comparison to ILP.

The custom topologies of the three benchmarks (263 dec-mp3 dec, 263 enc-mp3 dec, and mp3 enc-mp3 dec benchmarks) produced by clustering based techniques are shown in Figures 4.12, 4.17, and 4.22, respectively, for 5 port router architectures, and Figures 4.13, 4.18, and 4.23,

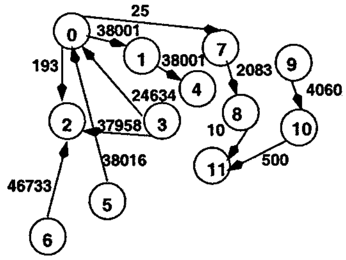

25
38001
0
1
38001
7
2083
9
193
24634
4
4060
8
2
37958
3
10
10
11
500
38016
46733
5
6

Fig. 4.14: 263 enc mp3 dec:Communication Trace Graph

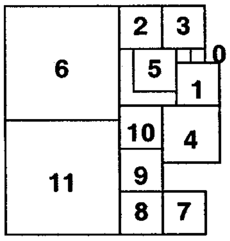

6
2
3
5
0
1
10
4
11
9
8
7

Fig. 4.15: Custom layout for 263 enc mp3 dec

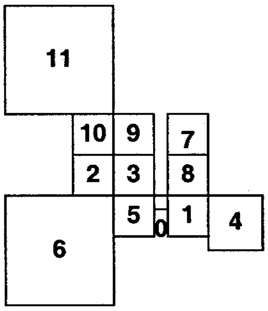

11
10
9
7
2
3
8
6
5
0
1
4

Fig. 4.16: Mesh layout for 263 enc mp3 dec

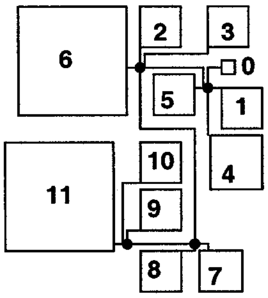

6
2
3
0
5
1
11
10
4
9
8
7

Fig. 4.17: 5 port topology for 263 enc mp3 dec

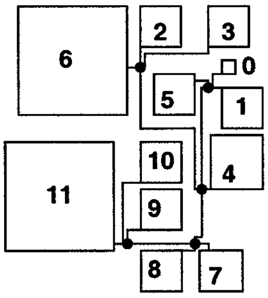

6
2
3
0
5
1
11
10
4
9
8
7

Fig. 4.18: 4 port topology for 263 enc mp3 dec

respectively, for 4 port router architectures.

We also compared our approach of synthesizing customized NoC designs for application specific SoC architectures against solutions with mesh based NoC topologies. The floorplans for custom and mesh architectures were obtained by invoking the ILP solver on the formulation presented in Section 4.2.1. Once the floorplan was obtained, we obtained the power and router consumption of a regular mesh topology by considering the shortest distance in terms of the number of routers from the source node to the sink node for each trace. The value thus calculated serves as a lower bound on the power consumption of the regular mesh topology. Similarly, we also obtained the number of routers and power consumption for the QNoC architecture [51]. The QNoC architecture is identical to a mesh topology except that it permits multiple cores to be attached to routers that are on the periphery of the mesh. Tables 5 and 6 show the results of

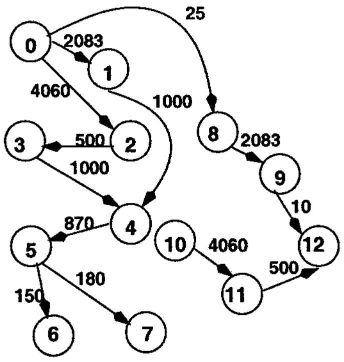

0
1
2
3
4
5
6
7
8
9
10
11
12
25
2083
4060
1000
500
1000
2083
10
870
4060
500
180
150

Fig. 4.19: mp3 enc mp3 dec:Communication Trace Graph

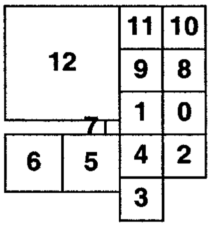

12
11
10
9
8
7
1
0
6
5
4
2
3

Fig. 4.20: Custom layout for mp3 enc mp3 dec

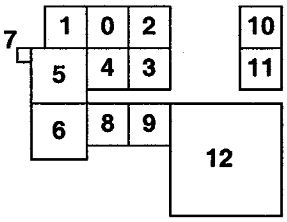

7
1
0
2
10
5
4
3
11
6
8
9
12

Fig. 4.21: Mesh layout for mp3 enc mp3 dec

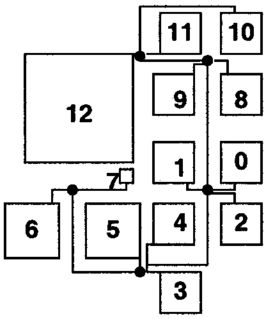

11
10
12
9
8
7
1
0
6
5
4
2
3

Fig. 4.22: 5 port topology for mp3 enc mp3 dec

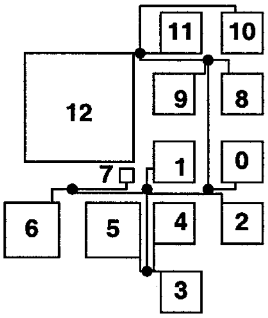

11
10
12
9
8
7
1
0
6
5
4
2
3

Fig. 4.23: 4 port topology for mp3 enc mp3 dec

the comparative study for each benchmark application. The number of ports in the routers was 5 in both cases. The number of nodes in the communication trace graph place a lower bound on the number of routers in both regular mesh and QNoC based topologies. The regular mesh based topology on an average consumes over 2.3 times more power and requires 3.5 times as many routers as a customized topology. The QNoC based topology on an average consumes 1.75 times more power and requires 1.4 times as many routers as a customized topology. The pre-designed physical connections in both the mesh based topologies force the communication traces to pass through more routers, thus, leading to the increased power consumption.

We compared the final solution generated by the clustering based heuristic with the theoretical lower bound on the power consumption of the ILP and clustering based formulations, respectively (see Table 7). We would like to emphasize that for every graph for which the ILP formulation resulted in a time out (rows 5-10 and 15-20), the lower bound is strictly a theoretical value, and it denotes the best lower bound that was generated by the formulation for a particular graph within the specified time. The lower bound for the clustering based technique for a graph is the summation of the lower bounds for the individual formulations for different clusters of the same graph. On an average for the larger graphs (rows 5-10 and 15-20) the final result of the clustering based technique is 1.37 (standard deviation of 0.31) and 1.08 (standard deviation of 0.17) times of the lower bounds due to the ILP and clustering based formulations, respectively. Thus, the clustering based heuristic generates results that are within 40 % of the theoretical optimal lower bound.

We also evaluated the number of additional virtual channels that are required for the custom architectures synthesized by our techniques. We found that for our set of applications, deadlock did not occur in any of the synthesized designs. We next measured the maximum number of traces flowing through any port of a router in the topology. This value acts as an upper bound on the number of virtual channels required in the design if a deadlock were to occur. We found that for most of the cases the number of traces were either 1 or 2. In only 5 (out of a total of 20) of the cases we had some ports that supported 3 traffic traces. Warnakulasuriya et al. [67] show that the deadlock probability in irregular networks becomes negligible with three virtual channels.

4.6. Conclusion

In this chapter, we defined the application specific NoC synthesis problem and proposed linear programming based solutions. We addressed the complexity of NoC synthesis problem by dividing it into two stages namely, floorplanning and interconnection network generation. We presented optimal ILP formulations for the two stages, and presented a clustering based heuristic for the second stage to reduce the run time of the formulation. We performed extensive experimentation to validate the quality of our techniques. The optimal ILP formulation timed out for many benchmarks. On the other hand, our clustering based technique was able to generate results with superior quality in reasonable time. On an average, the clustering technique consumed only 85% of the power and 96% of the router resources respectively, compared to the corresponding results generated by the ILP formulation. We also compared the custom topologies synthesized by our technique with regular mesh and QNOC based interconnection networks. The mesh and QNoC based topologies on an average consumed 2.3 and 1.75 times more power, and required over 3.5 and 1.4 times the router resources as compared to our custom topologies, respectively.

CHAPTER 5

LOW COMPLEXITY HEURISTICS FOR DESIGN OF CUSTOM NOC ARCHITECTURES

5.1. Introduction

In this chapter, we present a low complexity heuristic technique called ANOC for integrated core to router mapping, topology generation, and traffic routing for application specific NoC architectures. The problem has been formally defined in Chapter 2 of the thesis. Our technique operates on the system-level floorplan, and the corresponding Channel Intersection Graph (CIG). The nodes of the CIG are the possible locations where routers are allocated, and the edges are the possible physical links. Our technique simultaneously generates the NoC topology and routes by recursively partitioning the CIG, and mapping the traces on the links intersecting each partition. We demonstrate that the technique has a low complexity and can be utilized in an iterative search based framework for SoC design. We compare our techniques with the ILP formulation presented in Chapter 4 and demonstrate that for the multimedia benchmarks, ANOC is able to generate solutions that on average consume within 1.25 times the power consumption, and 0.95 times the router resources, compared to the ILP based technique. The chapter is organized as follows. In Section 5.2, we present the ANOC technique, in Section 5.4 we present experimental results, and finally in Section 5.5, we conclude the chapter.

5.2. Heuristic for Interconnection network generation

The different steps of ANOC technique are shown in the right-hand side of Figure 5.1. For explanation purposes, we consider the CTG and the corresponding floorplan shown in Figures 5.2(a) and 5.2(b), respectively. ANOC takes the SoC floorplan as input, and generates an interconnection network that minimizes power consumption subject to performance constraints on the traces, and bandwidth constraints on the router ports. ANOC attempts to route the traces

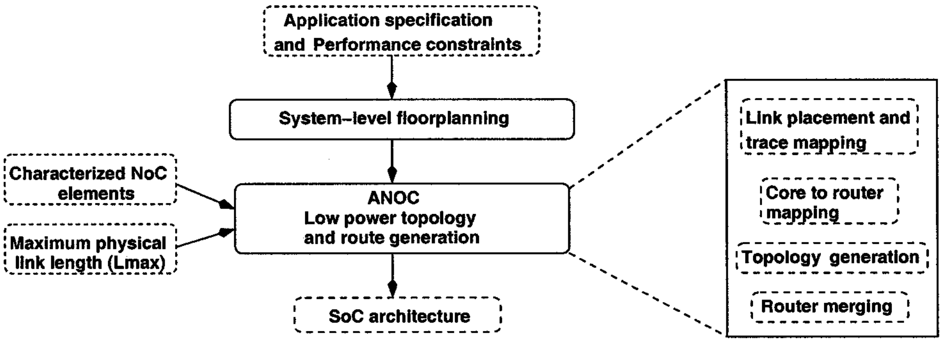
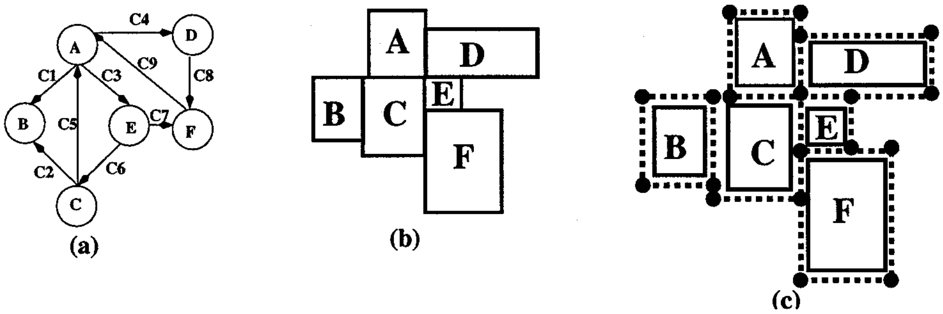

Application specification
and Performance constraints
System-level floorplanning
Characterized NoC elements
Maximum physical link length (Lmax)
ANOC
Low power topology
and route generation
SoC architecture
Link placement and trace mapping
Core to router mapping
Topology generation
Router merging

Fig. 5.1: ANOC design flow


A
B
C
D
E
F
C1
C2
C3
C4
C5
C6
C7
C8
C9
A
D
B
C
E
F
A
D
B
C
E
F

(a) (b) (c)

Fig. 5.2: CTG, floorplan and channel intersection graph

through shortest paths subject to the bandwidth constraints on the router ports, and thus minimizes power consumption.

Initially, our technique generates the channel intersection graph (CIG) of the given floorplan. To define a CIG, we introduce the notion of a bounding box. A bounding box is a rectangular enclosure of the core such that the bounding boxes of two adjacent cores abut each other. A CIG is defined on the bounding boxes, and is described by a graph in which the boundaries of the bounding boxes form the edges, and the intersection of two perpendicular boundaries form a node. Figure 5.2(c) depicts the CIG for the floorplan depicted in Figure 5.2(b). The dotted lines denote the edges of the CIG, and the black circles denote the nodes.

Once the CIG is obtained, we note that the area consumption of the router resources is much lower than the cores. Dally et al. [6] observed that the entire NoC is expected to consume only 6.6% of the total layout area. Moreover, we obtained area estimates on the router architecture presented by Banerjee et al. [22] and observed that a five port router architecture consumes an area of 0.38 mm^2 in 130 nm technology. On the other hands, the sizes of the cores range from 10 mm^2 to 50 mm^2 [68]. Therefore, the nodes of the CIG are assigned to be possible locations where routers can be allocated. The edges of the CIG are the possible locations of the physical links.

ANOC operates in four stages. In the first stage, it recursively bisects the channel intersection graph (CIG) of the given floorplan to determine the location of the physical links, and maps traces on the links. In the second stage, it attaches each core to one of the routers located at the four corners of its bounding box. In other words, the second stage performs the core to router mapping function. In the third stage, ANOC generates the NoC topology by determining trace routes, and placing routers at appropriate nodes of the CIG. Finally, ANOC invokes a merging algorithm that removes redundant routers from the topology. In the following sections we describe the four stages of ANOC in detail.

5.2.1. Link placement and trace mapping

ANOC generates an interconnection network by recursively bisecting the CIG. The dotted lines denote the edges of the graph, and the dark circles denote the nodes. As mentioned before, the edges of the CIG denote the possible locations of physical links on which the traffic traces are routed, and the nodes represent possible locations for router placement. A router port is composed of input and output ports that are attached to distinct physical links. Thus, each edge in the CIG consists of two links that support traffic in opposite directions. Therefore, the bandwidth of traffic that can be supported on each physical link is constrained by the port capacities.

We define a vertical cut as a line from top to bottom that divides the CIG into left and right

المنارة للاستشارات

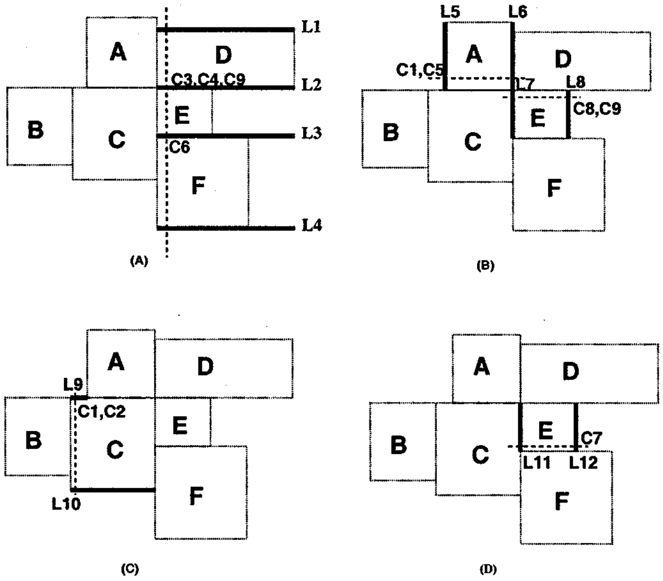


Fig. 5.3: Recursive partitioning link placement and trace mapping

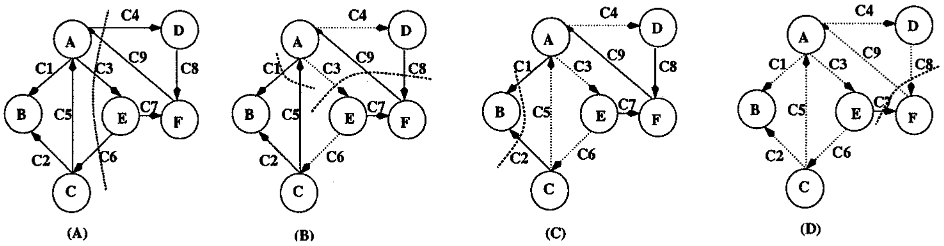


Fig. 5.4: Recursive cuts on the CTG

partitions. Similarly, we define a horizontal cut as a line from the left to right that divides the CIG into top and bottom partitions. A cut on the CIG divides it into two partitions of almost equal span in X or Y axis for a vertical and horizontal cut, respectively. Consider a vertical cut. It divides the graph into left and right sub-graphs of almost equal span in the X axis. All traffic traces crossing the cut are mapped on the horizontal links that are intersected by the cut. For each of the sub-graphs thus formed, a horizontal cut divides it into top and bottom sub-graphs of almost equal span in the Y axis. As before, all traffic traces crossing the cut are mapped on the vertical links that intersect the cut. The horizontal and vertical cuts are recursively repeated until each traffic trace is mapped to a link incident on the respective nodes of the CIG that define the bounding box of the source and sink cores.

The actual mapping of traffic traces to a finite set of physical links is a bin packing problem, which is known to be NP Complete. Hence, we propose a heuristic algorithm to map traces to the links. In order to minimize power consumption, ANOC routes traces with high bandwidth requirements first, so that they can be routed by shortest paths. Therefore, it selects traces from the cut in the decreasing order of the edge weight ($\omega(e)$). The trace is mapped on any one of the physical links that is within the bounding box (BB) defined by the closest corners of the source and sink nodes respectively, subject to the bandwidth constraint on the physical link. Such a mapping ensures that the trace is routed through the shortest path from its source to sink, and thus results in lower power consumption. In case no such link is available, the trace is mapped to the closest link outside the bounding box.

The mapping of traces in the subsequent cuts is performed by updating the bounding box for the traces at the end of each cut. For a trace tr, assume that $cur\_loc(tr, tr.src)$, and $cur\_loc(tr, tr.sink)$ denote the diagonally opposite end points of its bounding box. For the first cut, the algorithm sets $cur\_loc(tr, tr.src)$, and $cur\_loc(tr, tr.sink)$ for each traffic trace $tr \in E$ to be the respective locations of nearest corners of the source and sink cores in the layout. Without loss of generality, assume that the algorithm initially invokes a vertical cut, and the source of a traffic trace lies in the left partition. After mapping the trace on a link, the algorithm updates $cur\_loc(tr, tr.src)$ for the right partition, and $cur\_loc(tr, tr.sink)$ for the left partition to be the co-ordinates of the point of intersection of the link that maps the trace, and the cut. The updated locations are utilized to determine the bounding box of the trace in subsequent cuts. For example, for the vertical cut in Figure 5.3(A), trace c9 is mapped on link L2. For the subsequent horizontal cut that has c9 as a crossing traffic trace, the bounding box is generated with the first end being the intersection of L2 and the vertical cut, and the second end being node F. The algorithm recursively performs the link placement and traffic trace mapping, and thus establishes a route for all traffic traces.

Figures 5.3 and 5.4 illustrate the execution of the algorithm. The first cut shown in Figure 5.3(A) is a vertical cut that partitions the CIG into node sets {A,B,C}, and {D,E,F} respectively. The links that intersect the cut are {L1,L2,L3,L4}. The traces that intersect the vertical cut in Figure 5.4(A) (C3, C4, C6 and C9) are mapped on the physical links. The vertical cut is followed be two horizontal cuts, shown in 5.3(B). The two horizontal cuts partition the left and right CIGs into sets {A}, and {B,C}, and {D}, and {E,F}, respectively. The corresponding cuts on the traces are (shown in 5.4(B)) C1 and C5 for the left partition, and C8 and C9 for the right partition, which are mapped on the links {L5,L6} and {L7,L8}, respectively. Similarly, traces C1 and C2 are mapped on links {L9,L10} (in Figure 5.3(C)), and trace C7 is mapped on links {L11, L12} (in Figure 5.3(D)).

5.2.2. Core to router mapping

For each processing core, the technique inspects the closest links that carry traffic traces with the processing core as a source or sink, and places a router at the location of the node of the CIG, on which at least one link containing the traffic traces is incident. The processing core is connected to the router, and all traces that have the processing core as a source or sink, are routed through that router. This introduces extra link lengths for traces that were not originally incident on the router to which the core is mapped.

Figure 5.5(A) depicts the trace mapping stage of the algorithm. Now, the core to router allocation heuristic is invoked that maps cores to routers as shown in 5.5(B). In the figure, the black circles denote the routers, and the line joining a circle to a core denotes that the core is mapped to the particular router. Core to router mapping is accompanied by adjustment of traffic traces such that all traces are routed through the router to which the source and sink cores are

المنارة للاستشارات

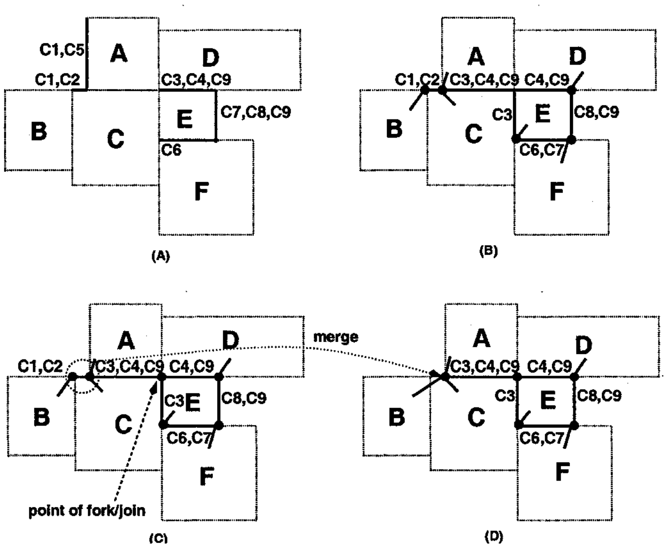

C1,C5
A
D
C1,C2
C3,C4,C9
B
C
E
C7,C8,C9
C6
F
(A)
A
D
C1,C2
C3,C4,C9
C4,C9
B
C
C3
E
C8,C9
C6,C7
F
(B)
A
D
merge
C1,C2
C3,C4,C9
C4,C9
B
C
C3
E
C8,C9
C6,C7
F
point of fork/join
(C)
A
D
C3,C4,C9
C4,C9
B
C
C3
E
C8,C9
C6,C7
F
(D)

Fig. 5.5: Topology generation and router merging

mapped. For example, due to the router mapping of core A in Figure 5.5 (B), extra link lengths are introduced for trace $c3$, as shown by the dark lines annotated with $c3$.

Since our objective is to minimize power consumption, ANOC places the router at the node of the CIG on which link containing maximum bandwidth is incident thus incurring extra link lengths for lower bandwidth traces in lieu of the high bandwidth traces, and therefore minimizing power consumption.

5.2.3. Topology generation

The mapping of traces on links by recursive partitioning of the floorplan generates a route for each traffic trace. A route can easily be determined by starting from the source node, and following the links that map the trace to the sink node. Once the routes are determined, the technique proceeds to generate the interconnection network topology. Intuitively, if a node of the CIG is a point of fork or join of three or more edges (or physical links) with some traffic flow, a router is required at the location of the node. As mentioned before, the dimensions of the routers

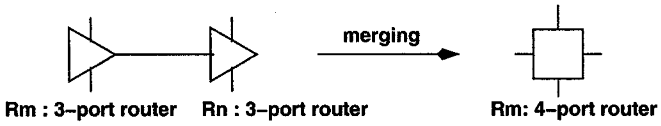


Fig. 5.6: Router merging

are much lower than the sizes of the cores. Hence, the technique considers the nodes of the CIG as possible locations for the routers. The introduction of routers in this manner generates a partial NoC topology.

Figure 5.5(C) depicts the stage when trace routes and physical links have been determined. At this point, routers are introduced at points of fork and join of the physical links, as shown in the figure.

5.2.4. Router merging

Finally, ANOC incorporates a merging step to reduce the number of routers in the topology and related power consumption. The technique examines pairs of adjacent routers that are connected by a physical link, and merges them if they are i)adjacent to each other, ii) a router architecture corresponding to the number of ports in the merged router is available in the library, iii) the total power consumption is minimized, and iv)merging the two routers does not cause the physical links to other routers in the architecture to exceed the maximum inter-router distance $|L_{max}|$. If two routers are closer than the specified maximum distance, they are candidates for merging. Merging of two routers is performed by removing the routers from the topology, and the introducing a new router that duplicates the router ports of the two routers.

Merging two routers that are not connected by a physical link always results in higher power consumption. This is due to the fact that on merging, the link lengths for traffic traces being routed through the collapsed router increases, thus increasing power consumption. Therefore, ANOC does not merge unconnected routers.

Figure 5.6 depicts the merging of two routers. Let r_m and r_n be two routers. Let η_m and η_n be the number of ports in r_m and r_n, respectively. As mentioned before, our technique only merges adjacent routers connected by a physical link. In the figure, r_m and r_n are merged to generate a new router r_k. After merging is performed, r_k will have $\eta_m + \eta_n - 2$ ports. The merging procedure does not alter the bandwidth flow in the different ports of the routers. In other words, except for the ports that are removed during the merging of two connected routers, the router after merging has a unique router port corresponding to each of the ports of the two un-merged routers. Therefore, the bandwidth constraints on the ports is always satisfied.

Figure 5.5(D) depicts the algorithm execution stage after merging. ANOC merges routers such that the power consumption in the topology is further minimized. The router that is introduced as the result of merging (r_k) can allocated to either one of the locations of the original routers (r_m or r_n). ANOC examines both possibilities, and allocates r_k to the location that minimizes power consumption.

The overall algorithm of ANOC is shown in Figure 5.7. Link placement and trace mapping is performed by function called *map_traces*() of algorithm *ANOC* shown in Figure 5.7. The function takes the CIG, and the direction of cut *dir* as input. The function returns if the number of nodes in the CIG is 1, denoting that no cuts are required. Line 4 determines the left and right sub-graphs by bisecting the CIG in the direction given by *dir*. Line 5 determines the physical links that intersect the cut and adds them to a list L. Line 6 determines the crossing traces. Lines 7 through 11 map the crossing traffic traces to the physical links. Lines 12 through 14 set the direction of cut of the sub-graphs to be complementary to the current direction of cut. Lines 15 and 16 recursively invoke the function for each of the sub-graphs.

The *generate_topology*() function of the ANOC algorithm performs the topology generation, and merging operations. In the function, lines 1 through 3 perform core to router mapping. Lines

```
ANOC (CIG, CTG)
1   map_traces(CIG, V)
2   generate_topology()
end
```

```
map_traces(cig, dir)
1   if (|cig| == 1)
2       return
3   end if
4   {lg, rg} = get_cut(cig, dir)
5   L = get_links(cig, dir)
6   C = get_traces(cig,dir)
7   while $C \neq \psi$
8       tr = select(C)
8       C = C - tr
8       get_bb(tr)
9       l = map_link(tr, L)
10      update_locations(tr, l)
11  end while
12  if (dir == V) cd = H
13  else cd = V
14  end if
15  map_traces(lg,cd)
16  map_traces(rg, cd)
end
```

```
generate_topology()
1   for core $c \in V$
2       assign_router(c)
3   end for
4   for $e \in E$
5       get_route(E)
6   end for
7   for node $n \in CIG$
8       if (fork_join(n))
9           place_router(n)
10      end if
11  end for
12  merge_routers()
end
```

Fig. 5.7: Interconnection network generation

4 through 11 generate the NoC topology by determining traffic routes (lines 4 through 6), and placing routers at locations of fork and join of the traces (lines 7 through 11). Finally, the merging of routers is performed by line 12 of the *generate_topology*() function.

We note that the resulting topology and trace routes can lead to deadlocks in the NoC. As mentioned in Chapter 4, deadlocks are removed by introducing additional virtual channels in the routers [65] at design time.

5.3. Complexity analysis

In our technique, there can be at most 4 physical links per node. Hence, the total number of physical links is $K = O(|V|)$. Since traces are mapped by recursive partitioning of the floorplan, each trace is mapped at most $O(log(|V|))$ times. For each trace mapping, at most K links

Benchmark	ID	Nodes	Edges
mp3 decoder	G1	5	3
263 encoder	G2	7	8
mp3 encoder	G3	8	8
263 decoder	G4	8	9
MPEG4	G5	12	13
MWD	G6	12	13
263 enc mp3 dec	G7	14	12
mp3 enc mp3 dec	G8	15	12
263 enc mp3 enc	G9	15	17
263 enc 263 dec	G10	16	17
Set top box	G11	25	28

Table 8: Benchmarks

Floorplanning	Interconnection generation algorithm	ID
ILP	ILP	MM
ILP	ANOC and merging	MA_m
ILP	ANOC no merging	MA_{nm}
Parquet	ILP	PM
Parquet	ANOC and merging	PA_m
Parquet	ANOC no merging	PA_{nm}

Table 9: Techniques

are explored, and there are a total of $|E|$ traces. Therefore, mapping of traces takes at most $O(K|E|log(|V|))$ operations.

The complexity of the topology generation operation is determined by the *get_route*, *assign_router*, *place_router*, and the *merge_routers* functions. For each trace, the *get_route* function examines at most K links. Hence, its complexity is $O(|E||V|)$. The *assign_router* function examines the four nodes at the corners of each core, and has a complexity of $O(|V|)$. The *place_router* function examines each node in the CIG, and as there are four nodes per core, has a complexity of $O(|V|)$. Finally, the *merge_routers* examines each pair of nodes to determine whether to merge them or not. Hence, it has a complexity of $O(|V|^2)$. Therefore complexity of the topology generation function is given by $O(|E||V| + |V|^2)$, where $|E||V|$ is for topology generation, and $|V|^2$ is for router merging. Hence, the overall complexity of the ANOC technique can be represented as $O(|V||E|log(|V|) + |V|^2)$.

5.4. Experimental results

In this section we present the results obtained by the execution of our technique on various multimedia benchmark applications. We first present details about the benchmark applications, the experimental setup, and then proceed to discuss the results in detail.

5.4.1. Benchmark applications

We generated custom NoC architectures for several combinations of multimedia benchmarks namely, i) MP3 audio encoder ii) MP3 audio decoder iii) H.263 video encoder, and iv) H.263 video decoder [52]. In addition, we obtained results for two other benchmarks namely, MPEG4 decoder, and multi-window display (MWD) applications [32]. We also generated results for a large set-top box application that we obtained from [52]. The description of the benchmarks is shown in Table 8.

5.4.2. Experimental setup and result analysis

We estimated the power consumption for the input and output traffic of a port in 100 *nm* technology to be $328\mu W/Mbps$ and $65.5\mu W/Mbps$, respectively. We estimated the physical link power consumption to be $79.6\mu W/Mbps/mm$ [12]. All results were obtained on a 950 MHz dual sparc processor.

We compared the custom NoC designs generated by ANOC with optimal ILP formulations [12]. For the floorplanning phase, we utilized two existing techniques namely, ILP based floorplanner [69], and a heuristic technique called Parquet [70]. Table 9 summarizes the different techniques utilized to compare our results. In the table, the first column represents the floorplanning technique, the second column represents the NoC topology and route generation technique, and the third column gives a unique name for each combination.

5.4.2.1. *Comparison with ILP formulation.* We compared our technique with an optimal ILP formulation proposed in [12]. The computational complexity of the ILP formulation is exponential in the number of inputs. We set the timeout period of the ILP formulation at 12 hours. The ILP formulation took several hours to generate results. In many cases, the formulation timed out before generating optimal results. On the other hand, the technique presented here is based on deterministic algorithms, and we show in Section 5.3 that the computational complexity of the

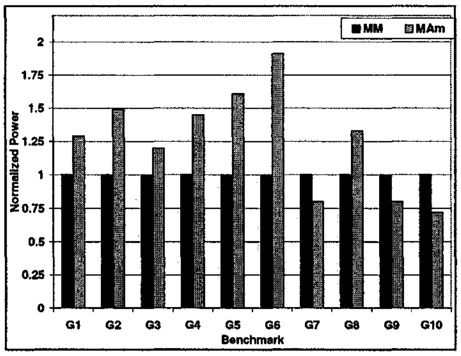


Fig. 5.8: Power comparison between MM and MA_m

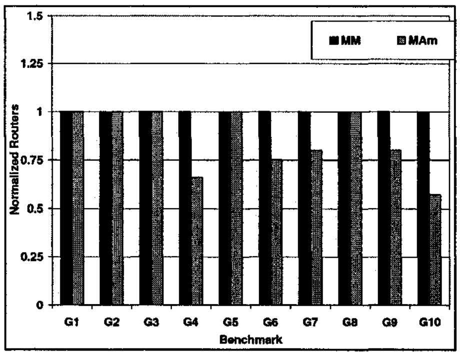


Fig. 5.9: Router comparison between MM and MA_m

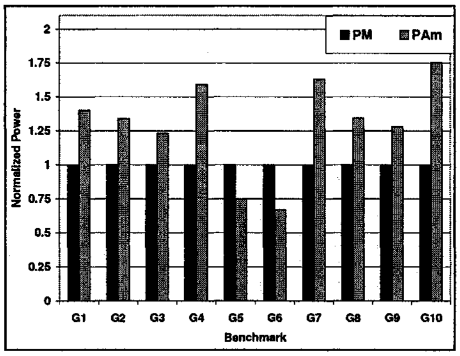


Fig. 5.10: Power comparison between PM and PA_m

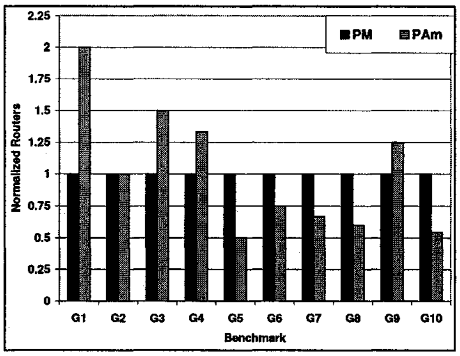


Fig. 5.11: Router comparison between PM and PA_m

technique is low. *Our technique was able to generate results for all benchmarks in negligible time (< 1sec), including the large set-top box application.*

The results are presented in Figure 5.8, and 5.9 for the ILP floorplan, and 5.10 and 5.11 for the Parquet floorplan, respectively. Figures 5.8 and 5.10 compare the power consumption of our technique with the ILP formulation, and Figures 5.9 and 5.11 compare the corresponding router resource consumption. Our technique was able to generate topologies that on average consumed only 1.25 times the power, and 0.85 times the router resources, compared to the corresponding topologies generated by the ILP formulation for ILP floorplan. For the Parquet floorplan, the corresponding values were 1.29 and 1.01, respectively. Our solutions generated NoCs with lesser

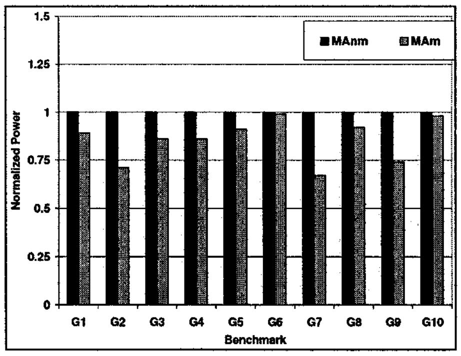


Fig. 5.12: Power comparison between MA_m and MA_{nm}

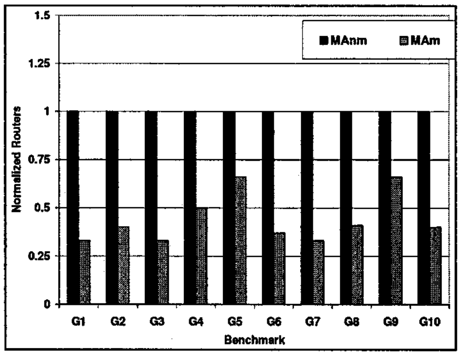


Fig. 5.13: Router comparison between MA_m and MA_{nm}

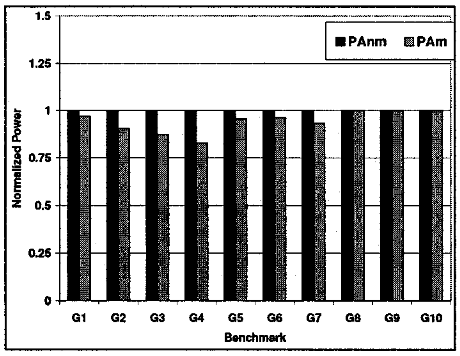


Fig. 5.14: Power comparison between PA_m and PA_{nm}

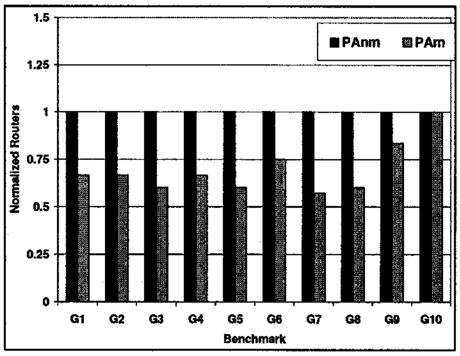


Fig. 5.15: Router comparison between PA_m and PA_{nm}

number of routers because the ILP formulation timed out for many benchmarks, thus resulting in sub-optimal solutions.

5.4.2.2. *Comparison of the techniques with and without merging.* Figures 5.12 and 5.13 compare the power consumption, and router resource consumption respectively, for the ANOC algorithm with and without merging respectively, on the floorplan generated by the ILP formulation. Figures 5.14 and 5.15 present a similar comparison for Parquet floorplan. On average, the merging of routers resulted in 15% reduction in power consumption and 45% reduction in consumption of router resources for the ILP based floorplan. The corresponding values for the Parquet floorplan were 6% and 31%, respectively. The reduction in router resources due to merging was much more

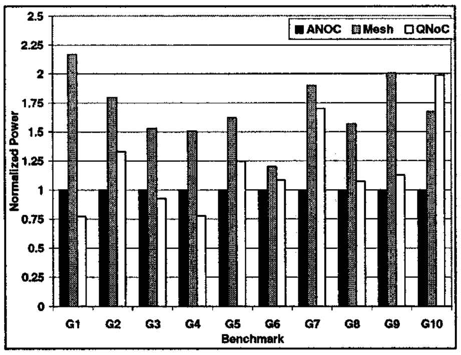


Fig. 5.16: Power comparison between ANOC, Mesh and QNoC for MILP floorplan

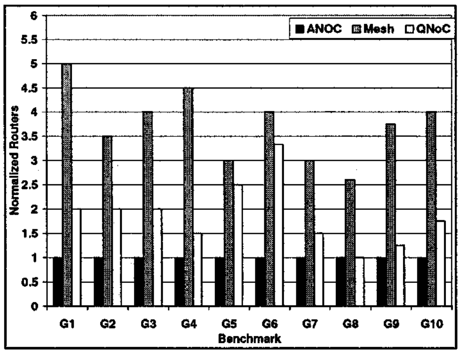


Fig. 5.17: Router comparison between ANOC, Mesh and QNoC for MILP floorplan

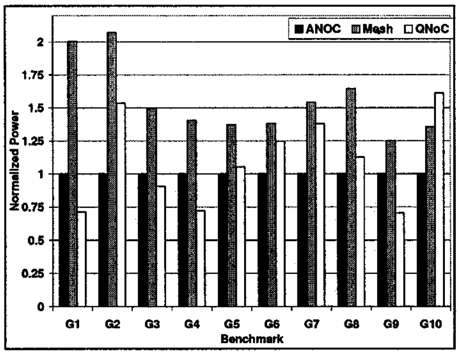


Fig. 5.18: Power comparison between ANOC, Mesh and QNoC for Parquet floorplan

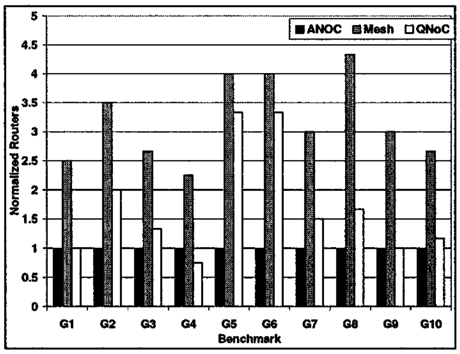


Fig. 5.19: Router comparison between ANOC, Mesh and QNoC for Parquet floorplan

than the reduction in power. The technique merges redundant routers, thus generating topologies that have higher router utilization. Although we do not address leakage power reduction explicitly, higher router utilization results to reduced leakage power.

5.4.2.3. *Comparison with optimal mesh based architectures.* We compared the results generated by our technique with optimal mesh and QNoC [51] based designs. Optimal results for mesh and QNoC based architectures were obtained by invoking the integer linear program formulation for mesh, presented in [12]. Figures 5.16 and 5.17 present the comparison of ANOC with Mesh and QNoC architectures when MILP based floorplanner was invoked. The corresponding comparisons for Parquet floorplanner are shown in 5.18 and 5.19, respectively. On average, our

Node	MPEG4	MWD
0	VU	IN
1	SDRAM	NR
2	ADSP	MEM1
3	AU	VS
4	UPSP	HS
5	MCPU	MEM2
6	SRAM1	HVS
7	RAST	JUG1
8	SRAM2	MEM3
9	BAB	JUG2
10	IDCT	SE
11	RISC	BLEND

Table 10: Node descriptions for MPEG and MWD

ID	Core type	Functionality
0	ASIC	ME
1	DSP	DCT and IDCT
2	DSP	Qnt & IQnt
3	DSP	FP
4	ASIC	VLE
5	CPU	MC, and ADD
6	MEM	FS0, FS1, FS2
7	DSP	FP
8	DSP	FFT, PA model
9	DSP	Filter, MDCT
10	CPU	It.enc.
11	ASIC	Bitres.
12	ASIC	Synch
13	MEM	Buf.
14	ASIC	Demux
15	DSP	VLD
16	DSP	IQ
17	DSP	IDCT
18	CPU	MC/Add
19	MEM	FS4, FS5
20	DSP	Huff.dec.
21	DSP	Bitres.
22	DSP	IMDCT/sum
23	MEM	Buf.
24	ASIC	Synch.

Table 11: Node description for Set-top box

technique generated results that consumed only 0.63 times the power and 0.3 times the router resources compared to the mesh based NoC architectures. The corresponding values for QNoC based architectures was 0.9 and 0.64, respectively. The pre-designed physical connections in both the mesh based topologies force the communication traces to pass through more routers, thus, leading to the increased power consumption.

5.4.2.4. *NoC designs.* Figures 5.20 , and 5.23 present the communication trace graphs for two multimedia applications namely, MPEG4 decoder [32], and Multi Window Display (MWD) [32]. The edges of the graphs are annotated by their bandwidth requirements. The description of the processing cores is shown in Table 10. Figures 5.21, and 5.24 present the corresponding physical layout, and interconnection network topology of the applications for the ILP floorplan for the MPEG-4 decoder and MWD benchmarks. Figures 5.22, and 5.25 present the corresponding

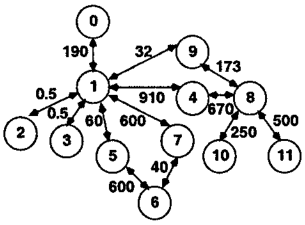

0
190
32
9
173
1
0.5
910
4
8
0.5
670
60
600
500
250
2
3
5
7
10
11
40
600
6

Fig. 5.20: MPEG4 CTG

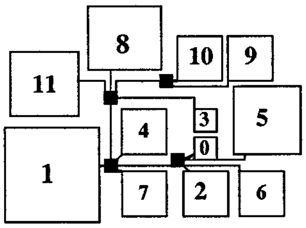

8
10
9
11
4
3
5
0
1
7
2
6

Fig. 5.21: MA_m MPEG4 topology

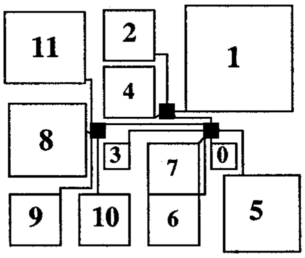

11
2
1
4
8
3
7
0
9
10
6
5

Fig. 5.22: PA_m MPEG4 topology

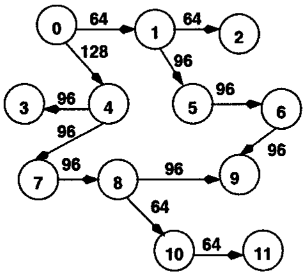

0
64
1
64
2
128
96
3
96
4
5
96
6
96
96
7
96
8
96
9
64
10
64
11

Fig. 5.23: MWD CTG

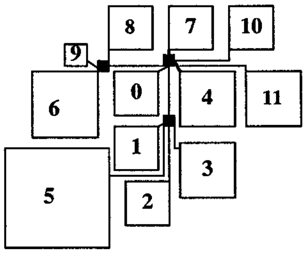

8
7
10
9
6
0
4
11
1
3
5
2

Fig. 5.24: MA_m MWD topology

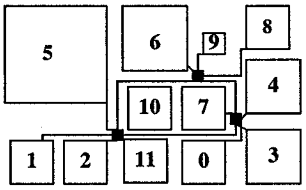

5
6
9
8
4
10
7
1
2
11
0
3

Fig. 5.25: PA_m MWD topology

physical layout, and interconnection network topology of the applications for the parquet floorplan. In the figure, the black squares denote the router elements.

Finally, we executed our algorithm for a large set-top box application obtained from [52]. Figure 5.26 depicts the CTG, and Table 11 presents the description of the processing cores. The first column of the table represents the core, the second column presents its type (ASIC, DSP etc), and the third column presents the functionality mapped to the core. Figure 5.27 presents the

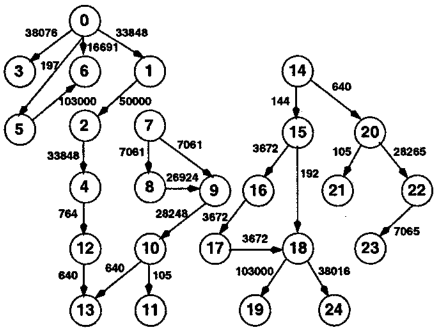

0
38076
33848
16691
3
197
6
1
103000
50000
5
2
7
33848
7061
7061
4
8
26924
9
764
28248
12
10
640
640
105
13
11
14
640
144
15
20
3672
105
28265
192
16
21
22
3672
17
3672
18
23
7065
103000
38016
19
24

Fig. 5.26: CTG for set-top box application

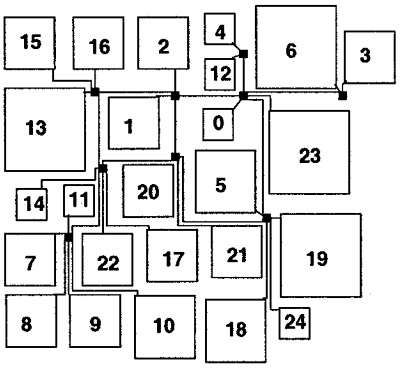

15
16
2
4
6
3
12
13
1
0
23
20
5
14
11
7
22
17
21
19
8
9
10
18
24

Fig. 5.27: Floorplan and interconnection network architecture

system-level floorplan and the interconnection architecture for the application. While technique was able to generate results for this benchmark in less than one second, the ILP based technique timed out without generating any results.

5.5. Conclusion

In this chapter, we presented a low complexity heuristic technique for the generation of custom NoC architectures. Our heuristic has a low complexity, and is able to generate solutions for large benchmarks in negligible time. We demonstrated the quality of the technique by comparing experimentation with several multimedia benchmarks, and comparisons with the optimal ILP formulation. Due to its low complexity, the technique is highly suitable for large problem sizes, and can also be utilized in a iterative SoC design framework. We compared the technique with the ILP based formulation presented in Chapter 4 and observed that on average ANOC consumes within 1.25 times the power and 0.95 times the router resources.

CHAPTER 6

APPROXIMATION ALGORITHMS FOR MAPPING AND ROUTING PROBLEMS DURING NOC DESIGN

6.1. Introduction

In this chapter, we discuss an approximation technique for NoC design problem defined in Chapter 2 of the thesis. Approximation algorithms guarantee that the quality of the solution generated by the technique is never worse than the proven approximation bound. Since the algorithm is polynomial time, it scales well with increasing problem sizes. Approximation algorithms are of immense value not only due to their ability to generate good solutions, but also in serving as a benchmark for large problem sizes, where ILP based techniques are not applicable due to their limited scalability.

In this chapter, we take the CTG, interconnection architecture elements, and a system level floorplan as input and address the interconnection architecture generation problem. We divide the problem into two sub-problems namely, router allocation and core to router mapping, and routing and topology generation. We present an optimal core to router mapping technique, and a 2-approximation for traffic routing and topology generation.

We compared the approximation algorithms with the ILP based technique and ANOC heuristic. Overall, the algorithm is able to generate close to optimal solutions in much less time compared to the ILP technique. Although the runtime of the technique is slightly more than ANOC, it is able to generate better quality solutions than ANOC.

The chapter is organized as follows. In Section 6.2, we present our algorithms for core-router mapping, in Section 6.3 we present our 2-approximation for routing and topology generation, in Section 6.4, we present our results, and finally in Section 6.5, we conclude the chapter.

6.2. Core to router mapping

Given a system-level floorplan, our technique assigns possible router locations to be the corners of the cores in the floorplan. Once the possible router locations are determined, our technique connects each core in the computation architecture to a unique router in the layout. Note that while several cores can be mapped to the same router, each core is uniquely mapped to a particular router. We assume that a core can be mapped to one of the four routers located at its corner. Thus, if (x, y) denotes the lower left hand side corner of a node v, the core is mapped to one of the routers located at (x, y), $(x + \mathcal{W}_v, y)$, $(x, y + \mathcal{H}_v)$ or $(x + \mathcal{W}_v, y + \mathcal{H}_v)$, where $\mathcal{W}_v$ is the width of the core, and $\mathcal{H}_v$ is its height.

The objective of the core to router mapping is to minimize the power consumption. As the topology of the NoC is not defined at this stage, we abstract the power consumption as the power consumed by point to point physical links between two routers. For each core i, let R_i denote the set of routers located at its four corners. Core i is mapped to one of the routers in R_i. Let $X_{i,j}$ denote a (0,1) integer variable that is set to 1 if node i is mapped to router $j \in R_i$, else 0. Let $X_{i,j,k,l}$ denote a (0,1) integer variable that is set to 1 if node i is mapped to router j, and node k is mapped to router l, else 0. We define these variables only when $(i, k) \in E$ or $(k, i) \in E$. The objective is to minimize the power consumption expressed as:

$$Minimize\ Z = \sum_{(i,k)\in E} \sum_{j\in R_i} \sum_{l\in R_k} \omega(i,k) \cdot \psi_l \cdot dist(j,l) \cdot X_{i,j,k,l} \tag{6.1}$$

where $dist(j, l)$ is the Manhattan distance between the two routers j and l. In this section we prove that the core to router mapping problem is equivalent to max-flow min-cut problem, and therefore can be solved optimally in polynomial time.

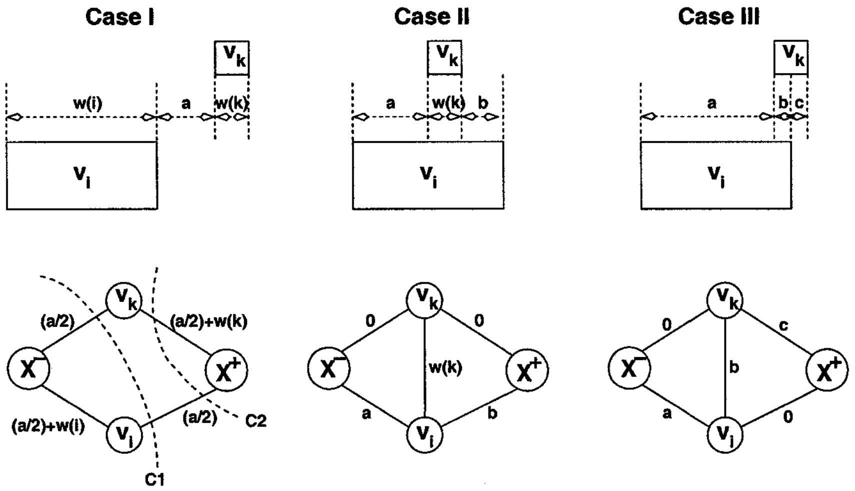


Fig. 6.1: Core alignments and flow graphs

6.2.1. Equivalence to max-flow min-cut problem

The minimization goal can be split as the sum of two terms:

$$Z = \sum_{(i,k),j,l} \sigma(i,k) \cdot x(j,l) \cdot X_{i,j,k,l} + \sum_{(i,k),j,l} \sigma(i,k) \cdot y(j,l) \cdot X_{i,j,k,l} \tag{6.2}$$

where $\sigma(i,k) = \omega(i,k) \cdot \psi_l$, $x(j,l)$ is the x-offset between the two routers, and $y(j,l)$ is the y-offset between the two routers. We can consider the overall problem as a composition of two sub-problems that determine the x and y co-ordinate, respectively of the router to which the core is mapped.

Without loss of generality we first consider the sub-problem that determines the x co-ordinates of the routers for all the cores. Thus, we wish to determine if the core should be mapped to a router in the x^- (left hand of the core) or x^+ (right hand side of the core) location. Based on the relative locations of two communicating cores v_i and v_k on the floorplan we have three cases as shown in top row of Figure 6.1. For each case we construct a flow graph as shown in the lower row of the figure. In each graph we introduce two nodes x^- and x^+ in addition to v_i and v_k. We also add edges between the various nodes and annotate them with weights. The weight of an edge is derived from the various distances as specified in top row of the figure. A cut of each of the graphs that assigns x^- and x^+ to different partitions denotes the mapping of the cores to the routers. The weight of the cut given by the summation of weights of edges that are cut denotes the x-offset between the routers to which the cores have been mapped. For example in Case I, the cut C1 denotes that v_k is mapped to x^- and v_i is mapped to x^+. The weight of C1 given by a denotes the x-offset between the x^- router of v_k and x^+ router of v_i. Similarly, the cut C2 denotes that v_k and v_i are both mapped to their respective x^+ router, the x-offset is $a + \mathcal{W}(k)$. The cut weight multiplied by the bandwidth of traffic between the two cores ($\omega(i,k)$) and link power model (ψ_l) denotes x-offset component of the power consumption due to the mapping (first term of equation 6.2). For a pair of communicating cores (i, k) we can find the lowest power consumption router mapping by generating the minimum cost cut in the flow graph.

We now generalize the above construction to the entire CTG. We construct a flow graph $G(W, F)$ with $W = V \cup x^+ \cup x^-$ where x^+ and x^- are additional nodes that represent the routers. For every trace $(i, k) \in CTG(V, E)$ we classify the trace in to one of the three categories based on the core locations, and introduce edges in flow graphs. The edge weights are given by the product of the communication bandwidth ($\sigma(i,k)$), link power model (ψ_l) and distance weights as described in the previous paragraph.

Theorem 1 *A cut in Graph $G(W, F)$ that assigns x^- and x^+ to different partitions represents a solution to the x co-ordinate sub-problem.*

Proof 1 *Consider a core v_i. A cut either intersects the edge from x^- to v_i, or the edge from x^+ to v_i, but not both. Moreover, a cut must intersect one of the two edges. If the edge from i to x^- is cut, it represents that core v_i is mapped to router at x^- off-set (and vice versa). The weight of the cut represents the power consumption contribution due to the mapping of the routers to the respective x^+ or x^- routers. Thus, the cut captures both the mapping of the cores to the routers, as well as the power consumption incurred due to the assignments. Therefore, the cut represents* *a solution to the x co-ordinate sub-problem.*

From Theorem , it follows that if we could generate a cut of minimum cost, we have an optimal solution to the x co-ordinate sub-problem. This is a polynomial time solvable max-flow-min-cut problem and we solve it by invoking the Push-Relabel algorithm [71] that has a complexity of $O(n^3)$ in the number of nodes of the graph. We can similarly solve the y co-ordinate sub-problem, the utilize the two solutions to determine the unique router for mapping each core.

6.3. Routing and topology generation

The output of the first stage of our custom NoC design technique is a system-level layout with cores and routers, and an assignment of the cores to specific routers (see left hand side of Figure 6.2). Thus, the source and sink routers for each trace are known. The routing and topology generation problem must minimize power consumption and router resources subject to several constraints posed by the router architecture library that is available, and target frequency of operation. Our technique addresses the following issues.

1. *Power consumption:* The primary objective of the NoC is that it should consume minimum power. Since power consumption is directly proportional to the bandwidth flow in the network, the traces must be routed through minimum power consuming paths. We present graph transformations, and invoke the routing technique on the transformed graph such that all routes consume minimum power.

2. *Minimum number of routers:* Along with power minimization, it is highly desirable to have minimum number of routers in the topology. Minimum number of routers directly correspond to lesser static power consumption, lesser wiring and faster design and verification. We present an ILP formulation, and a integer relaxation based technique that generates NoC topologies with a guarantee that the number of routers is at most twice the corresponding

optimal solution. When the technique is invoked on the transformed graph mentioned above, it guarantees that the power consumption is minimized, along with the approximation guarantee on the routers.

3. *Bandwidth constraints:* The routing and topology generation technique must satisfy bandwidth constraints on the router ports. We propose a technique to introduce additional ports in the routers to ensure that bandwidth constraints are also satisfied along with minimum power and router resource consumption.

4. *Router port constraints:* The routing and topology generation technique is constrained by the largest router available in the library. For example, if the library can support only upto 5 port routers, the technique cannot generate a solution in which a router has 6 ports or more. We present additional constraints to the ILP and heuristics to generate NoC topologies that satisfy the port constraints in the routers.

5. *Maximum link length between routers:* At high frequency operations, the delay in link and the router are comparable. For example, if the router operates at $6GHz$, (clock width of 166 ps) and the link delay is 20 ps/mm, it puts a upper limit of 8.3 mm on the link length to ensure single clock cycle data transfer. Since we already know the locations of the routers, we can easily address this constraint. The topology generation technique ensures that two connected routers are never further than the maximum link length apart.

6. *Deadlocks:* A routing protocol must be deadlock free. Utilizing the static nature of the routing technique, we add virtual channels to break any possible deadlocks in the network.

We present our routing and topology generation technique by first discussing a base case that minimizes the number of routers in the NoC, and building upon the base case to incorporate low power requirements, bandwidth constraints, and port constraints. The presentation of our technique is organized as follows.

i First, we consider the problem of NoC topology generation such that the total number of routers in the topology is minimized. This problem does not address low power requirements, bandwidth constraints, port constraints or deadlocks. For this problem, we present a linear programming based technique that guarantees a 2-approximation on the quality of the solution.

ii Second, we address the problem of NoC topology generation with minimum number of routers, subject to low power requirements. To solve this problem, we propose graph transformations to ensure minimum power consumption, and utilize the technique for the first problem to minimize the routers.

iii Third, we address the problem of NoC topology generation such that the total number of routers are minimized subject to low power consumption, and bandwidth constraints on the router ports. A NoC topology with minimum routers subject to low power consumption requirements is obtained by invoking the technique for the second problem. Bandwidth constraints are satisfied by introducing additional ports in the router architecture, and dividing the traffic traces across the ports.

iv Fourth, we address the problem of NoC topology generation such that the total number of routers are minimized subject to low power consumption, and bandwidth constraints on the router ports when the router architecture is constrained by a maximum number of ports. We present linear programming formulation, and rounding based heuristics to solve the problem.

v Finally, at the end of each of the above mentioned techniques, we invoke a router merging step that merges routers that are close to each other, such that the power consumption and router resource consumption is minimized further.

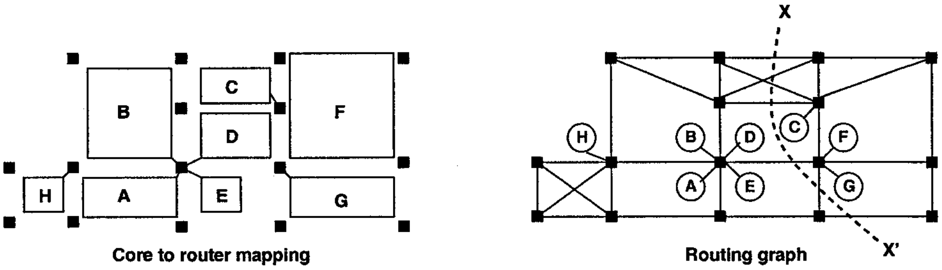


Fig. 6.2: Topology design

6.3.1. Topology generation with least number of routers

In this section, we present our technique to solve the problem of NoC topology generation such that the total number routers utilized in the topology is minimized. This problem is a node weighted generalized steiner forest problem, and is known to be NP hard [72]. We formulate the problem as an ILP. As mentioned before, it is known that integer relaxation of ILP formulations can be solved in polynomial time [71]. We prove that by utilizing iterative rounding on the integer relaxation of the ILP we can establish a 2-approximation on the quality of the solution in polynomial time. In other words the number of routers in the topology generated by our solution will be at most twice that of the optimal solution. In the following paragraphs, we first present a cut based ILP formulation for our problem. The cut based ILP has exponential constraints, and hence, is not suitable for practical applications. Therefore, we also present a equivalent flow based LP with polynomial number of constraints. We prove our 2-approximation bound by utilizing the LP of the cut based ILP, and iteratively solve the equivalent flow based LP to obtain the solution in polynomial time.

6.3.1.1. *Cut based ILP for minimizing routers.* Given the set of routers and core to router mappings, our technique initially determines the available paths from the source to the sink of each trace. This is done as follows. We construct a graph $G_r(V_r, E_r)$ called the routing graph, where V_r is the set of available routers, and there exists an edge $e = \{r_i, r_j\}$ whenever the Manhattan distance between r_i and r_j is less than D_{max}. Figure 6.2 depicts a core to router mapping and the corresponding routing graph. In the figure, the black squares that denote the routers, are the nodes of the routing graph. The edges of the routing graph denote the possible physical links in the final NoC.

Let $\mathcal{C}$ denote a cut that divides the nodes of the routing graph G_r into two subsets S and S'. In Figure 6.2, the curve $X - X'$ represents a cut. For the cut, let $\delta(S)$ denote the set of edges that cross the cut. The edges intersected by cut $A - A'$ form the set of edges of δ. For each cut $\mathcal{C}$ that divides the routers into sets S and S', we define a function called $F(\mathcal{S})$ that is set to 1 if there exists a trace such that its source lies in S and sink lies in S', or vice versa. Otherwise it is set to 0. Let X_r denote a (0,1) variable that is set to 1 if router r of the CTG is utilized in the NoC topology. Now, the LP can be formulated as follows:

$$Minimize\ Z = \sum_r X_r \qquad such\ that$$

$$\forall \mathcal{C} \quad \sum_{\forall e(r,s) \in \delta(\mathcal{S})} X_r \geq F(\mathcal{S})$$

The above constraint states that for a cut $\mathcal{C}$ that partitions the routers into sets S and S' such that $F(\mathcal{S}) = 1$, at least one router which is adjacent to the cut must be utilized in the final topology. Applying this constraint on all the cuts in the graph ensures that a route exists for all traces. Since the number of cuts in a graph is exponential, the cut based formulation defines a polytope with exponential constraints. We can reduce the number of constraints by utilizing an alternative flow based formulation. It is a well known fact that the cut based formulation and the flow based formulation are equivalent [73].

6.3.1.2. *Flow based ILP for minimizing routers.* The objective of the flow based formulation is same as that of the cut based formulation. Let $Y_{r,s,k}$ denote a (0,1) variable that is set to 1 if trace k passes through the physical link $(r, s) \in L$. For each core m in V, let $m\mathcal{M}r$ denote the relation that m is mapped to router r. We have the following constraints :

- Any router that maps a core must be present in the network. Therefore, we have the following constraint.

  $$\forall r, \exists m \in V : m\mathcal{M}r, X_r = 1$$

- A trace in the CTG always passes through the source and sink routers. For trace $t = (m, n)$, let q denote the router that maps m, and r denote the router that maps n. Since the trace is routed through the two routers, one physical link connected to the respective routers must be present in the topology. Hence, the following equality must hold.

  $$\sum_{q:(q,s)\in L} Y_{q,s,t} = 1 \quad \sum_{q:(s,q)\in L} Y_{s,q,t} = 0$$

  The first constraint ensures that there is exactly one link through which the trace t leaves router q. The second constraint ensures that the trace does not enter router q from any other router.

  Similar constraints for the router mapping the sink are as follows.

  $$\sum_{r:(s,r)\in L} Y_{s,r,t} = 1 \quad \sum_{r:(r,s)\in L} Y_{r,s,t} = 0$$

- For each trace, the flow due to the trace must be conserved in all routers except the routers that map the source and sink.

  $$\forall k \in E \sum_{r:rs\in G_r} Y_{r,s,k} = \sum_{w:st\in G_r} Y_{s,t,k}$$

- If there exists a flow through a router, that router is utilized in the topology.

  $$\forall r, \forall k, X_r \geq \sum_{s:qr\in L} Y_{q,r,k}$$

The flow-based ILP only has $O(|E||L|)$ constraints. Therefore, the integer relaxation of the flow based ILP is suitable for direct solution by solvers.

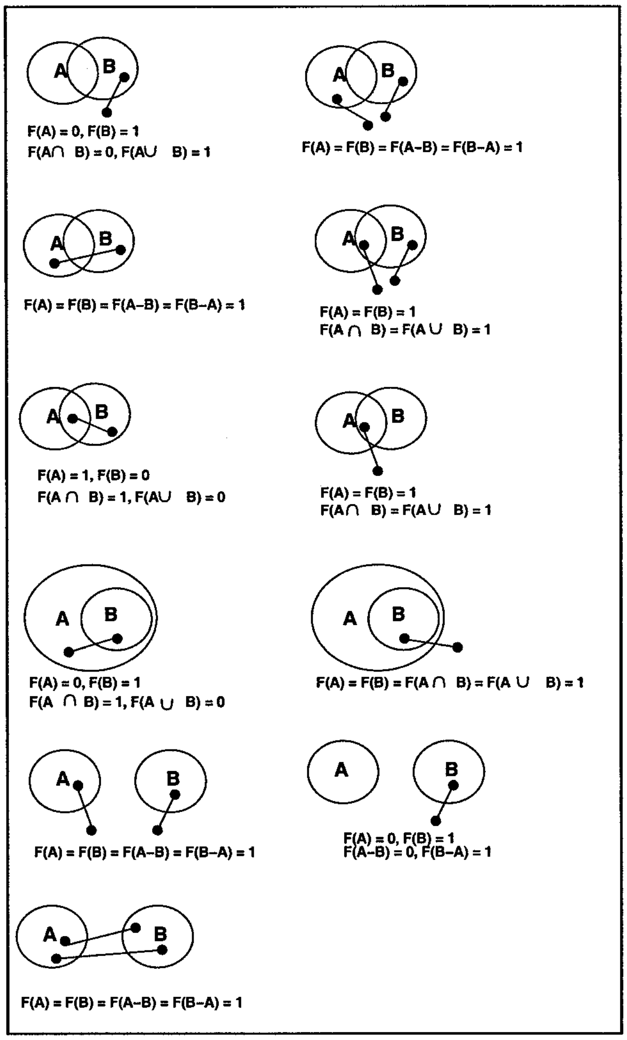

A
B
F(A) = 0, F(B) = 1
F(A∩ B) = 0, F(A∪ B) = 1
F(A) = F(B) = F(A−B) = F(B−A) = 1
F(A) = F(B) = F(A−B) = F(B−A) = 1
F(A) = F(B) = 1
F(A ∩ B) = F(A ∪ B) = 1
F(A) = 1, F(B) = 0
F(A ∩ B) = 1, F(A∪ B) = 0
F(A) = F(B) = 1
F(A∩ B) = F(A∪ B) = 1
F(A) = 0, F(B) = 1
F(A ∩ B) = 1, F(A ∪ B) = 0
F(A) = F(B) = F(A ∩ B) = F(A ∪ B) = 1
F(A) = F(B) = F(A−B) = F(B−A) = 1
F(A) = 0, F(B) = 1
F(A−B) = 0, F(B−A) = 1
F(A) = F(B) = F(A−B) = F(B−A) = 1

Fig. 6.3: Proof of supermodularity

6.3.1.3. *Algorithm and proof of 2-approximation.* In this section, we present our technique for obtaining integer solution from the LP relaxation, and prove that our technique utilizes at most twice the number of routers compared to the optimal solution.

Definition 1 *A function* $F : 2^R \rightarrow \mathbf{Z}$ *is said to be weakly supermodular [72] if* $F(R) = 0$*, and at least one of the following conditions is true for any two sets* $A, B \in R$ *:*

- $F(A) + F(B) \leq F(A - B) + F(B - A)$
- $F(A) + F(B) \leq F(A \cap B) + F(B \cup A)$

Lemma 1 *The function F defined in the cut based ILP is weakly supermodular.*

Proof 2 *As all traces are contained in the G_r, $F(R) = 0$. In order to prove the second part of the theorem, we enumerate all cases, and show that the supermodularity condition holds. The different cases are shown in Figure 6.3. The ovals A, B denote the two sets, the black circles denote two cores which may either lie inside or outside the sets, and the line joining the cores denotes the communication trace. Each case in the figure is annotated with one of the inequalities that satisfies the supermodularity conditions.*

Fact 1 *For any weakly supermodular function F, any extreme point solution x to the LP must pick some router to the extent of at least a half. In other words, $X_r \geq \frac{1}{2}$ for at least one router r [72].*

Our algorithm exploits Lemma 1 and Fact 1, and utilizes an iterative rounding procedure on the integer relaxation solutions of the flow based ILP to generate the topology. The algorithm is presented below.

```
    Begin algorithm LP_round
  Initialize $H \leftarrow \phi : F' \leftarrow F$
  While $F' \neq 0$ do
    Solve LP to obtain solution $\mathbf{X}$.
    For each variable $x \in \mathbf{X}$ do
      If $(x \geq \frac{1}{2})$ then
        round $x$ to 1
        $H = H \cup x$
      end-If
    End for
    Update $\forall S \subseteq V, F'(S) = F(S) - |\delta_H(S)|$
```

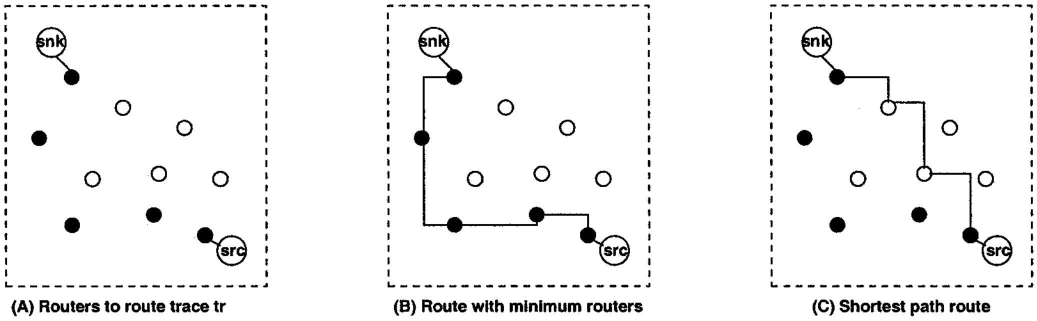


Fig. 6.4: Routing with shortest path and minimum routers

```
    End While
    Output H
End algorithm LP_round
```

Theorem 2 *Algorithm LP_round achieves an approximation ratio of 2 for the routing problem.*

Proof 3 *Jain [72] gave an LP formulation for the edge weighted steiner tree problem, and utilized an iterative rounding procedure that results in factor 2 approximation for the problem. Our problem is an instance of the node weighted steiner tree. Consider an optimal solution* (X, Y) *to our LP. First, note that* Y *alone is feasible for the LP of Jain, because the constraints do not involve* X *variables: they require that a certain flow be supported (or that certain cuts be crossed) by the selected edges. Thus Jain's result implies that in an optimal (extreme point) solution to the LP there exists a triple* (q, r, k) *such that* $Y_{q,r,k} \geq 1/2$*. Now for this particular* r*, we have the constraint* $X_r \geq \sum_{s:qr \in L} Y_{q,r,k}$*, and so* $X_r \geq 1/2$*. That is, in any optimal (extreme point) solution to our LP, there exists an* r *with* $X_r \geq 1/2$*, and so in the for loop that the algorithm exercises in each iterative rounding phase, at least one more variable is fixed to* 1*. By Lemma 1, and Fact 1, our LP in the next phase satisfies all the requirements for Jain's theorem, and so the algorithm proceeds until a feasible integral solution is found.*

Now we note that every time any variable X_r *was fixed to* 1*, this was either as part of an* *optimal LP solution, or because its fractional value was at least* $1/2$. *To prove the approximation bound, we operate on the number of iterations (phases). Suppose now for that* X_1 *is the LP solution found in the first iteration. Let* $C(X_1)$ *denote the cost of the LP. Let* W_1 *be the set of vertices (routers) picked by rounding in the first iteration. By Fact 1,* W_1 *is nonempty. In other words,* W_1 *has at least one variable that is set to a value greater than half. Since the rounding was applied only to the variables with values at least* $1/2$, $C(W_1) \leq 2C(X_1)$. *Let* W_1' *denote the routers that were not picked in the first iteration. Let* X' *denote the partial solution of the LP corresponding to* W_1'. *For example, if the LP assigned the values* $\{x1 = 0.4, x2 = 0.4, x3 = 0.4, x4 = 0.7\}$ *to its variables,* W_1' *would be* $\{x1, x2, x3\}$, *and* X_1' *would be* $\{x1 = 0.4, x2 = 0.4, x3 = 0.4\}$. *The cost of* X_1' *is given by*

$$C(X_1') = C(X_1) - \sum_{X_r \geq \frac{1}{2}} X_r$$

In the second iteration, only routers belonging to W_1' *are considered. The constraints are satisfied by assigning appropriate values for the variables in* W_1'. *Note that these constraints do not depend on the routers in* W_1 *that have been picked after the first iteration. This is because, all constraints that depend on the routers in* W_1 *have already been satisfied, and do not occur in the second iteration. Moreover, the constraints of the LP in the second iteration are a subset of the constraints of the LP in the first iteration. Therefore, the partial solution* X_1' *must satisfy these constraints, and hence is a feasible solution to the LP in the second iteration. Since* X_2 *is the optimal solution for the LP in the second iteration,*

$$C(X_2) \leq C(X_1')$$

Substituting the value of $C(X_1')$,

$$C(X_2) \leq C(X_1) - \sum_{X_r \geq \frac{1}{2}} X_r$$

Readjusting the inequality and multiplying both sides by 2, we have

$$2C(X_1) \geq 2C(X_2) + 2 \sum_{X_r \geq \frac{1}{2}} X_r$$

Let W_2 *be the routers that are picked in the second iteration. By supermodularity,*

$$C(W_2) \leq 2C(X_2)$$

$$2 \sum_{X_r \geq \frac{1}{2}} X_r \geq \sum_{X_r \geq \frac{1}{2}} 1 = C(W_1)$$

Therefore, we have

$$2C(X_1) \geq C(W_2) + C(W_1)$$

The number of iterations required to obtain the final solution is upper bounded by the number of available routers. Let n *be the total number of iterations. Proceeding in the above fashion, we can show that after the* n^{th} *iteration,*

$$2C(X_1) \geq C$$

where C *is the cost of the final solution, and is given by*

$$C = C(W_1) + C(W_2) + \cdots C(W_n)$$

Now, since $C(X_1)$ *is the cost of the LP solution in the first iteration,*

$$C(X_1) \leq C(ilp)$$

where $C(ilp)$ *is the cost of the optimal solution to the ILP. Therefore,*

$$C \leq 2C(ilp)$$

The argument presented above demonstrates that the cost of the solution is at most twice the cost of the optimal ILP.

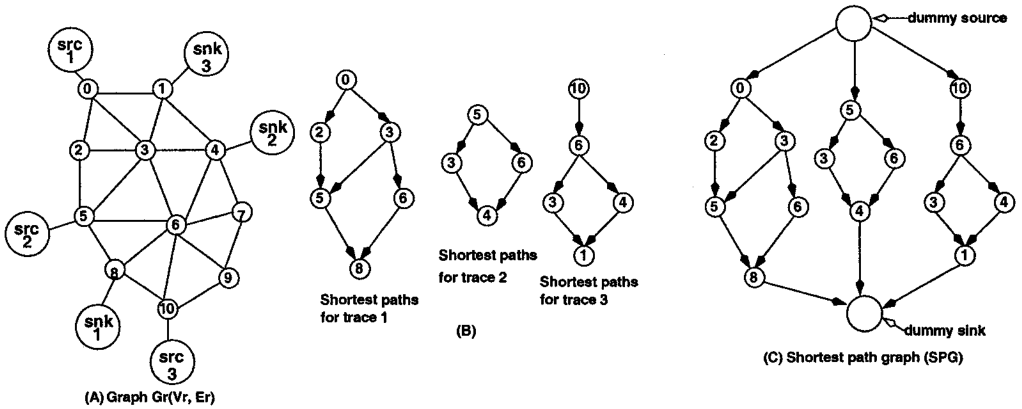

src 1
snk 3
snk 2
src 2
snk 1
src 3
0
1
2
3
4
5
6
7
8
9
10
(A) Graph Gr(Vr, Er)
Shortest paths for trace 1
Shortest paths for trace 2
Shortest paths for trace 3
(B)
dummy source
dummy sink
(C) Shortest path graph (SPG)

Fig. 6.5: Construction of SPG

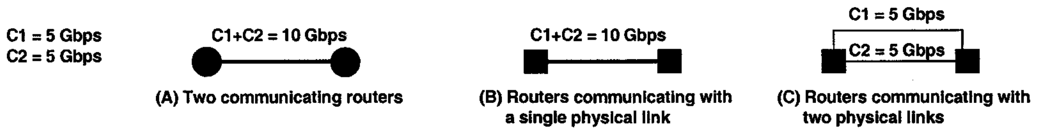

C1 = 5 Gbps
C2 = 5 Gbps
C1+C2 = 10 Gbps
(A) Two communicating routers
C1+C2 = 10 Gbps
(B) Routers communicating with a single physical link
C1 = 5 Gbps
C2 = 5 Gbps
(C) Routers communicating with two physical links

Fig. 6.6: Logical versus physical routing

6.3.2. NoC topologies with minimum power consumption

For a trace *tr*, we define a shortest path route to be one that consumes minimum power. The total power consumption for the route is given by the power consumption due to the routers as well as the physical links. The technique presented in the previous section does not ensure shortest path routes for the traces. Therefore, some traces may be routed by longer paths, resulting in higher power consumption. For example, consider the route for trace *tr* in Figure 6.4. In the figure, the darkened circles are already present in the topology, and the empty circles are not present in the topology. A traffic route for *tr* that consumes minimum number of routers is shown in part B of the figure. Clearly, this is not the shortest path for the trace. The shortest path is shown in part C of the figure, that consumes two extra routers. If the design objective is to minimize power consumption, the shortest path should be chosen at the expense of extra routers.

6.3.2.1. *Determination of shortest paths.* In order to determine the shortest paths, we again consider the routing graph. We associate a weight with each edge, which intuitively gives us the power consumption if the physical link corresponding to the edge is utilized in the final topology.

For each edge $e = \{r_i, r_j\}$, ρ_e denotes its edge weight, and is given by

$$\rho_e = \psi_i + \psi_o + l_e \times \psi_l$$

where l_e denotes the Manhattan distance between r_i and r_j. The edge weights are assigned such that they capture the link as well as the router power consumption.

Now, for each trace, we invoke Dijkstra's shortest path algorithm to determine the available shortest paths. Note that there can be multiple shortest paths from a source to a sink. The output of the call to the algorithm is a collection of subgraphs, each subgraph denoting the shortest paths for a particular trace. Consider the routing graph G_r depicted in Figure 6.5(A). In the example, we are required to route three traces, (src_1, snk_1), (src_2, snk_2), and (src_3, snk_3). The corresponding shortest path graphs are depicted in part (B) of the figure. Each node has a unique ID expressed as a double (r, tr) where r denotes the router in V_r, and tr denotes the trace to which the node belongs. For example, router $(2, 1)$ denotes router with ID 2 in V_r, and belonging to the graph corresponding to trace 1. Finally, we generate a connected graph by introducing a dummy source and sink node, and connecting the all the source nodes to the dummy source, and the sink nodes to the dummy sink. We denote this graph by *shortest path graph* or SPG.

We define $I(r, tr) = 1$ if node (r, tr) is present in the solution obtained by invoking our routing and topology generation technique on the SPG. Otherwise, $I(r, tr) = 0$. We now define a mapping function, $I(r \times tr) \rightarrow X_r$ as follows.

$$X_r = \begin{cases} 1, \ if \ \forall tr \in E, \ \exists I(r, tr) = 1 \\ 0, \ otherwise \end{cases}$$

X_r is set to 1 if any of the traces utilizes router r. Therefore, X_r denotes the number of routers in the topology. We obtain our topology with minimum routers and shortest paths by invoking our linear programming based technique on the SPG, with an objective of minimizing the the sum of X_r. The cut based formulation is formulated as

$$Minimize \sum_{r} X_r \quad such\ that$$

$$\forall \mathcal{C} \quad \sum_{\forall e((r,tr),(s,tr)) \in \delta(\mathcal{S})} X_r \geq F(\mathcal{S})$$

As before, we formulate an equivalent flow based formulation for the problem, and solve it in polynomial time with a 2-approximation guarantee.

6.3.3. Satisfying bandwidth constraints

The routing technique presented above does not address bandwidth constraints on the router ports. As a result, the bandwidth constraints on some ports may be violated. If there are bandwidth violations in router ports, traffic routing is performed by introducing new router ports in the routers as follows. Figure 6.6(A) depicts a logical link between two routers. The link carries two traffic traces C1 and C2, each with a bandwidth requirement of $5Gbps$. The total bandwidth flow on the link is the sum of the bandwidth flow due to C1 and C2. The routers are treated as black boxes, and the link conveys the information that the two routers communicate with the specified bandwidth. If a single physical link between two routers can satisfy the bandwidth requirement on it, the router to router connection is established by the physical link (see Figure 6.6(B)). On the other hand, if a single physical link is unable to meet the bandwidth requirements, the logical connection between two routers is established by introducing new ports in the routers, and corresponding links, as shown in 6.6(C). The traffic traces are distributed among the links such that the links can support the bandwidth flow in them. In the figure, traces C1, and C2 are routed through different links, thus resulting in $5Gbps$ traffic on each link, instead of the original $10Gbps$ traffic. The actual physical links, even when implemented in the top metal layer, consume only $0.3\mu m$ per wire [74]. Therefore, representing a logical link with multiple physical links does not cause an appreciable difference in link lengths, and corresponding link power consumption.

In order to minimize the complexity of the router architecture that is synthesized, the introduction of new ports should be such that the traffic traces are routed through minimum number of router ports. For each logical link, we determine the minimum number of ports and corresponding physical links required to route all the traces on the logical link such that the bandwidth constraints on the ports are satisfied. If traces t_1 to t_n are routed through the logical link, and each trace t_i has a bandwidth requirement of ω_i, the required number of ports (p) must be more than the ceil of the ratio of the total bandwidth flow due to the traces, and the capacity of the port. Mathematically,

$$p = \left\lceil \frac{\sum_{i=1,n} \omega_i}{\Omega} \right\rceil$$

where Ω is the bandwidth constraint on the port.

6.3.4. Satisfying port constraints

At the end of routing and topology generation stage and addition of router ports to satisfy bandwidth constraints, the topology may contain routers that have more ports than the maximum number of ports allowed per router. In this case, we introduce and additional constraints to the ILP formulation and utilize the technique of iterative rounding of the integer relaxation to arrive at the final solution. Due to the additional constraints, we cannot guarantee an approximation bound on the solution quality. In the following paragraphs, we present details of our technique to generate NoC topologies with both port and bandwidth constraints.

6.3.4.1. *Additional constraints for number of ports.* : Let $T_{r,s}$ be a binary variable that is set to 1 if physical link (r, s) in the routing graph is utilized to route some trace. If no trace is routed through (r, s), $T_{r,s}$ is set to zero. Therefore, the following constraint must hold.

$$\forall t \in E,\ T_{r,s} \geq Y_{r,s,t}$$

Since a physical link from r to s corresponds to a physical link from s to r, we have the following constraint.

$$\forall r, s\ T_{r,s} = T_{s,r}$$

The total number of ports for router r consists of all the inter-router physical links, and the cores to which the router is attached. In order to ensure that the number of ports is less than the maximum allowable number, we need the following constraint.

$$\forall r \sum_{r,s \in G_r} T_{r,s} \leq \eta_{max} - \sum_{m:m\mathcal{M}r} 1$$

6.3.4.2. *Presence of port and bandwidth constraints.* : Due to port constraints, the technique presented in Section 6.3.3 that assigns several physical links per logical link between routers to satisfy bandwidth constraints, is no longer applicable. Therefore, instead of satisfying bandwidth constraints as a post processing step, we introduce additional constraints in the formulation. The following constraint ensures that the bandwidth constraint on the router ports are not violated.

$$\forall (r, s) \sum_{t} (\omega(t) \times Y_{r,s,t}) \leq \Omega$$

Now, to enable multiple physical links, we transform the SPG to generate SPG2 as follows. For each edge e in the SPG, introduce η_{max} edges in SPG2. The edge e denotes a logical link between the two routers. By introducing η_{max} edges for e in SPG2, we enable generation of η_{max} physical links in the solution obtained on SPG2, for each logical link e in SPG. The LP formulation invoked with the additional port constraints on SPG2 thus ensures that in the final solution, the number of ports per router is less than η_{max}, and at the same time, multiple physical links between routers are explored to satisfy the bandwdith constraints.

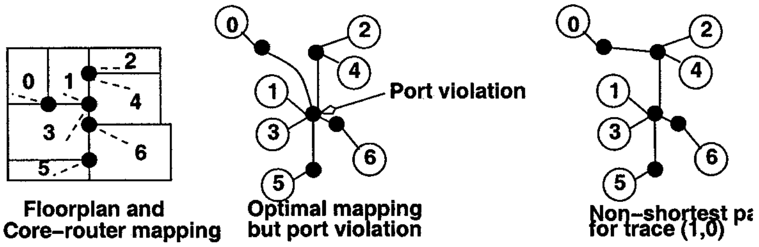

0
1
2
3
4
5
6
Floorplan and
Core-router mapping
Port violation
Optimal mapping
but port violation
Non-shortest pa
for trace (1,0)

Fig. 6.7: Resolving infeasible solutions due to port constraints

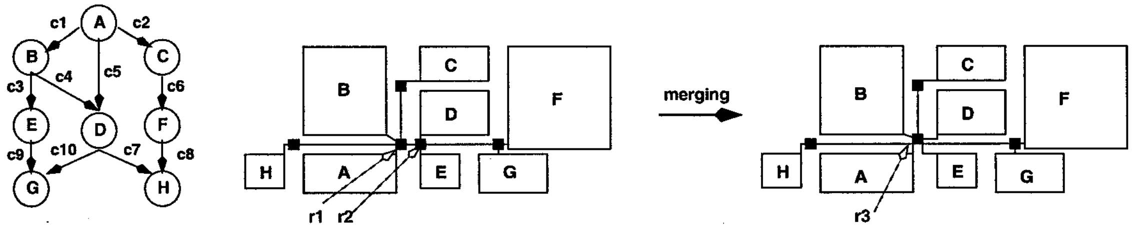

A
B
C
D
E
F
G
H
c1
c2
c3
c4
c5
c6
c7
c8
c9
c10
merging
r1
r2
r3

Fig. 6.8: Router merging

6.3.4.3. *Modification of iterative rounding technique.* : The iterative rounding technique presented in Section 6.3.1.2 cannot be directly applied under port constraints. This is because, the rounding technique assumes that at least one variable is set to a value greater than 0.5, and rounds it to 1. This assumption is not valid under the additional constraints. Therefore, in each iteration, we pick a variable with maximum value, and round it to 1. As before, since at least one variable is rounded to 1 in each iteration, the total number of iterations is upper bounded by the number of variables.

6.3.4.4. *Infeasible solutions.* :Due to the presence of port constraints, no feasible solution to the LP might exist. For example, consider Figure 6.7. In the figure, traces $(1,0)$, $(1,2)$, $(1,3)$, $(1,4)$, $(1,5)$, and $(1,6)$ have a unique shortest path from router 1 to the router to which each node is mapped. If the router is constrained to have at most 5 ports, at least one trace must be routed through a non-shortest path, such that the port constraints are satisfied.

We solve the infeasibility problem by relaxing the constraint of shortest path on certain traces. The selected traces to be routed through longer path should be such that

- unmapping the trace alleviates the port constraint on the router, and
- the trace has minimum bandwidth requirement, so that it through a longer path results in minimum additional power consumption.

We define a router to be an articulation point of a trace if any shortest path of the trace must be routed through the router. In Figure 6.7, the router mapping core 1 is the an articulation point for all the traces, and the routers mapping the other respective cores are articulation points for the traces originating or sinking at the cores. Our technique examines articulation points that cause a port violation, and selects the trace with least bandwidth requirement. The constraint of shortest path is relaxed for this trace. The LP is invoked again on the modified SPG. This step is repeated until a feasible solution is obtained.

6.3.5. Router merging

At the end of the mapping and routing stages, the architecture may have routers that are placed very close to each other. We merge pairs of routers that are less than distance D_{max} apart. Our merging technique checks all pairs of available routers. Two routers are candidates for merging if the following conditions hold.

- The distance between them is less than D_{max}.
- Merging the two routers does not violate the D_{max} constraint for any other router in the topology.
- The port and bandwidth constraints are not violated.
- The overall power consumption does not increase as a result of merging.

The merging step is continued as long as there are pairs of routers that satisfy the above criteria.

The merging step is depicted in Figure 6.8. In the figure, $r1$, and $r2$ are the candidate routers. Merging is performed by introducing a new router $r3$ at one of the locations corresponding to $r1$ or $r2$. The new router ($r3$) is placed at the location that results in lower power consumption.The location of $r3$ is determined as follows. If the location is chosen to be the location of $r1$, the following changes in power consumption take place.

- The power consumption due to traces $c3$ through $c8$ are reduced due to the reduction of one hop in their respective routes.
- The power consumption due to trace $c9$ and $c10$ are increased due to the increase in the corresponding link lengths.

Similarly, the change in power consumption due to the placement of $r3$ at the location of $r2$ is calculated. The location that results in greater reduction in power consumption is chosen as the location of $r3$. The introduction of the router is followed by mapping the cores mapped to $r1$ and $r2$ to $r3$, generating a router to router link between $r3$ and all routers to which $r1$ and $r2$ are connected, and updating the traffic routes for all traces through $r1$ and $r2$. Finally, $r1$ and $r2$ are removed from the topology.

6.4. Results

In this section, we present the results obtained by execution of our technique on several multimedia and network processing benchmark applications. We compare the results generated our technique with an optimal ILP based technique [12], and a recursive partitioning based heuristic [15] that addresses the same problem. In Section 6.4.1 we discuss the benchmark applications, in Section 6.4.2 we discuss the experimental setup, and finally in Section 6.4.3 we present and discuss the results.

Graph	Graph ID	Nodes	Edges
JPEG Encoder	G1	8	21
MPEG-4 decoder	G2	12	13
MWD	G3	12	13
VOPD	G4	12	13
Set-top Box	G5	25	27
AH Auth-IPv4	G6	9	8
Diffserv-IPv4	G7	11	10
NP1	G8	15	22
NP2	G9	17	24
NP3	G10	24	42

Table 12: Benchmarks

6.4.1. Benchmark applications

We generated custom NoC architectures for five multimedia, and five network processing benchmarks. The details of the benchmarks are presented in Table 12. Benchmarks G1 through G5 are multimedia applications, and benchmarks G6 through G10 are network processing applications. The size of the benchmarks varied from 8 nodes and 21 edges to 25 nodes and 27 edges. The following is a brief description of the benchmarks.

- *JPEG Encoder:* A VHDL implementation of the JPEG Encoder core was obtained from the opencores website [75]. We simulated the VHDL model and generated the flow graph and the corresponding CTG for the application.

- *MPEG-4 decoder, MWD, and VOPD:* The MPEG-4, Video Object Plane Decoder (VOPD) and Multi Window Display (MWD) applications were obtained from the work presented by Jalabert et al. [32].

- *AH Auth-IPv4, and Diffserv-IPv4:* The process flow graph and CTG for these applications were obtained by profiling them on a network processor [76].

- *NP1, NP2, and NP3:* NP1, NP2, and NP3 are industrial strength network-processing benchmarks obtained from the work presented by Pasricha et al. [77].

For the benchmarks G1 through G7, the size of the ARM core was estimated to be $3 \times 3 = 9mm^2$ [78]. For benchmarks G8 through G10, the size of the cores were provided in [77]. The network interface, whose area is estimated to be $0.2mm^2$ [79] was included in the calculation of the core area. The router architecture utilized in the work consists of 2 virtual channels, FIFO depth of 16, and width of 32 bits. For a 5 port router, the area was estimated to be $0.19mm^2$ in 65 *nm* technology. For a 9 port router, the area was estimated to be $0.37mm^2$. For routers with more ports, the area was correspondingly scaled.

6.4.2. Experimental setup

In this section, we describe the power model and floorplanner utilized in this work, the maximum number of ports per router architecture and the maximum allowable link lengths for which NoC topologies were generated, and the simulation environment utilized to simulate our NoC designs.

6.4.2.1. *Power models and floorplanner.* We characterized our router architecture on a 65 *nm* Synopsys low power library. In this technology, the power consumption for the input and output port was estimated to be $204nW/Mbps$ and $94nW/Mbps$, respectively. The link power consumption was estimated to be $89nW/Mbps/mm$. We utilized the Parquet floorplanner [70] to obtain our floorplans.

6.4.2.2. *Architecture specific settings.* As mentioned in Section 6.3, the NoC topology generation technique must take the router architecture parameters into account. Therefore, we experimented with the following four settings

- Case-1 : No restriction on the number of ports, and no restriction on the link lengths.
- Case-2 : No restriction on the number of ports, but link length restricted to $6mm$.
- Case-3 : Number of ports restricted to 5, and no restriction on link lengths.

	Power		Routers	
	$\frac{LP}{ILP}$	$\frac{LP}{ANOC}$	$\frac{LP}{ILP}$	$\frac{LP}{ANOC}$
CASE-1	1.04	0.90	1.21	1.31
CASE-2	1.04	0.93	1.12	1.21
CASE-3	1.22	NA	1.18	NA
CASE-4	1.22	NA	1.18	NA

Table 13: Average power and router consumption

Runtime (secs)			
Graph size	Tech	ILP	ANOC
5	< 1	~ 5	< 1
15	~ 1	> 42100	< 1
25	~ 5	> 42100	< 1

Table 14: Runtimes

- Case-4 : Number of ports restricted to 5, and link lengths restricted to $6mm$.

Case-1 depicts a low frequency operation that allows long link lengths, and a router library that can generate routers with any number of ports. Case-2 targets a high frequency operation where the link delays are significant, and a maximum link length must be imposed to enable single clock-cycle transfer. Case-3 denotes a low frequency operation, but with a router library that can only generate routers with at most 5 ports. Finally, Case-4 denotes a high frequency operation, along with a router library that can generate only 5 port routers.

6.4.3. Analysis and discussion

In accordance with the proof in Section 6.2, the core to router mapping stage produced optimal solutions. The solutions generated by the routing and topology generation stage also converged to optimal solutions. Figures 6.9 - 6.12 present the comparisons of power and router resource consumption between our technique and the existing techniques. The bars in the figures are normalized to the optimal solution generated by the ILP based technique. In the figures, the first bar denotes the power consumption of our technique, the second bar denotes the power consumption of ANOC, the third bar denotes the router resource consumption of our technique, and the fourth bar denotes the router resource consumption of ANOC. Since the ILP based technique optimizes power consumption in isolation, our technique consumed less routers than the ILP for some benchmarks.

6.4.3.1. *Summary of results.* Table 13 presents the summary of results for the comparison of our technique with existing techniques. Our technique generated close to optimal results for

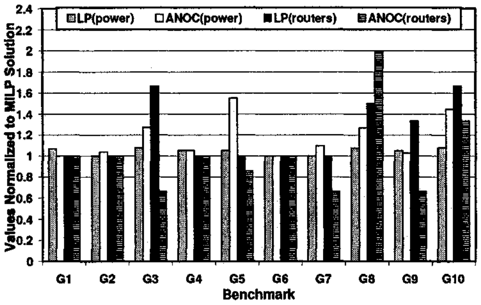


Fig. 6.9: Power and router comparisons with no port and no link constraints

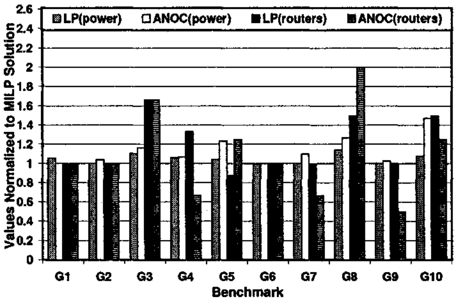


Fig. 6.10: Power and router comparisons with no port constraint but with link constraints

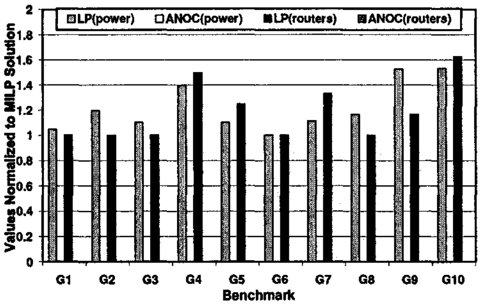


Fig. 6.11: Power and router comparisons with port constraints but no link constraints. ANOC violated port constraints for all benchmarks and is not plotted

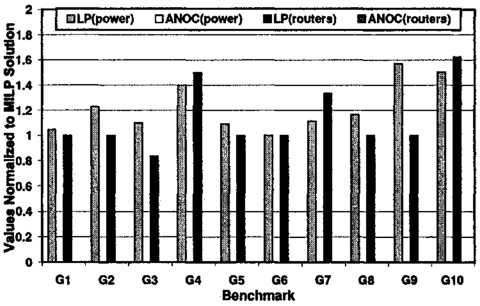


Fig. 6.12: Power and router comparisons with power and router constraints. ANOC violated port constraints for all benchmarks and is not plotted

all benchmarks. Table 14 presents the comparison of the runtimes of the techniques. While it took the ILP several hours to generate optimal solutions, our technique generated solutions in a few seconds. The ANOC technique being a low complexity heuristic, generated solutions within a second. In the following paragraphs, we summarize our results in the absence and presence of port constraints, respectively.

6.4.3.2. *Absence of port constraints:* In the absence of port constraints, our technique generated solutions with the approximation guarantee. As a result, our technique was able to generate results that matched very closely with the optimal solution. When compared with ANOC, our

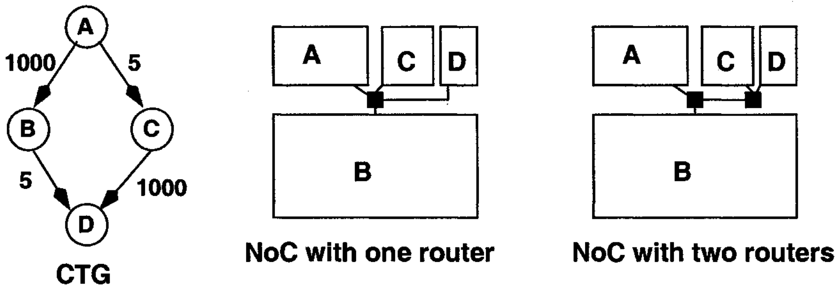


Fig. 6.13: Minimizing power consumption by introducing additional routers

ID	Set-top Box	NP	ID	Set-top Box	NP
0	ASIC	ARM	13	MEM	MEM
1	DSP	ARM	14	ASIC	MEM
2	DSP	ASIC	15	DSP	MEM
3	DSP	ASIC	16	DSP	MEM
4	ASIC	ASIC	17	DSP	MEM
5	CPU	DMA	18	CPU	MEM
6	MEM	Watchdog	19	MEM	SDRAM
7	DSP	UART	20	DSP	ACC1
8	DSP	ITCI	21	DSP	NI-1
9	DSP	SDRAM	22	DSP	NI-2
10	CPU	MEM	23	MEM	NI-3
11	ASIC	TIMER	24	ASIC	
12	ASIC	SlvAC			

Table 15: Node description for Set-top box, and NP

technique consistently performed in terms of power consumption. Since power consumption was the primary objective, our technique trade-off minimization of router resources in favor of lower power consumption. As a result, it generated solutions with higher number of routers compared to ANOC.

In the absence of port as well link length constraints, a possible solution is to connect all cores to a single router. However, this may not be the optimal solution. This is because, a single router solution may not be the best solution in terms of power consumption. This fact is illustrated in Figure 6.13. Placing just one router, and connecting all cores to that router results in long link lengths and link power consumption due to the edge (C, D), which has a high bandwidth requirement. This can easily be offset by introducing an additional router such that link lengths corresponding to the routes of highly communicating cores (edge (C, D)) are

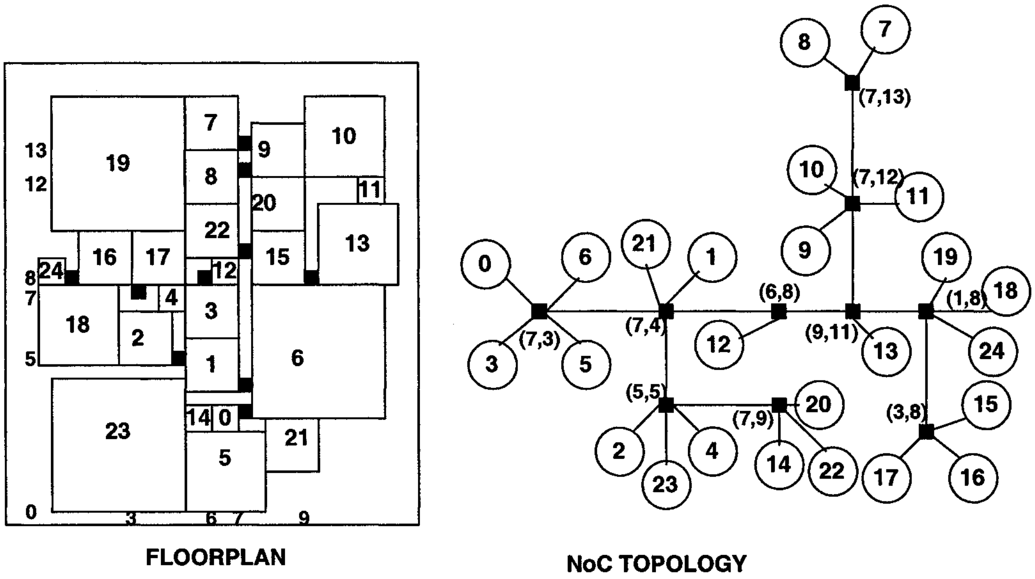

FLOORPLAN
13
12
19
7
8
22
9
20
10
11
13
16
17
24
15
12
8
7
4
18
2
3
1
6
5
0
14
23
21
5
0
3
6
7
9
NoC TOPOLOGY
8
7
(7,13)
10
(7,12)
11
9
0
6
21
1
19
(6,8)
18
(1,8)
(7,3)
(7,4)
(9,11)
3
5
12
13
24
(5,5)
20
(3,8)
15
(7,9)
2
4
14
22
17
16
23

Fig. 6.14: Floorplan and NoC topology for Set-top box

minimized. Please note that if router minimization was the sole criterion, our technique generates one router solutions.

6.4.3.3. *Presence of port constraints:* ANOC does not consider port constraints, and as a result, did not generate valid solutions for most of the cases. When port constraints were imposed, our technique invoked the heuristic technique to satisfy port constraints, and as a result, the results are slighlty worse. Nevertheless, the solutions generated by our technique are within 22% of the optimal.

6.4.3.4. *NoC designs.* We present NoC designs for two large benchmarks, a set-top box benchmark (benchmark G5), and a network processing benchmark (benchmark G10), respectively. The CTG for the benchmarks can be obtained from [52], and [77], respectively. Figure 15 presents the description of the nodes for the applications.

The custom topologies of the two benchmarks produced by our linear programming based technique with 5 port routers and a link length constraint of 6*mm* are shown in 6.14, and 6.15

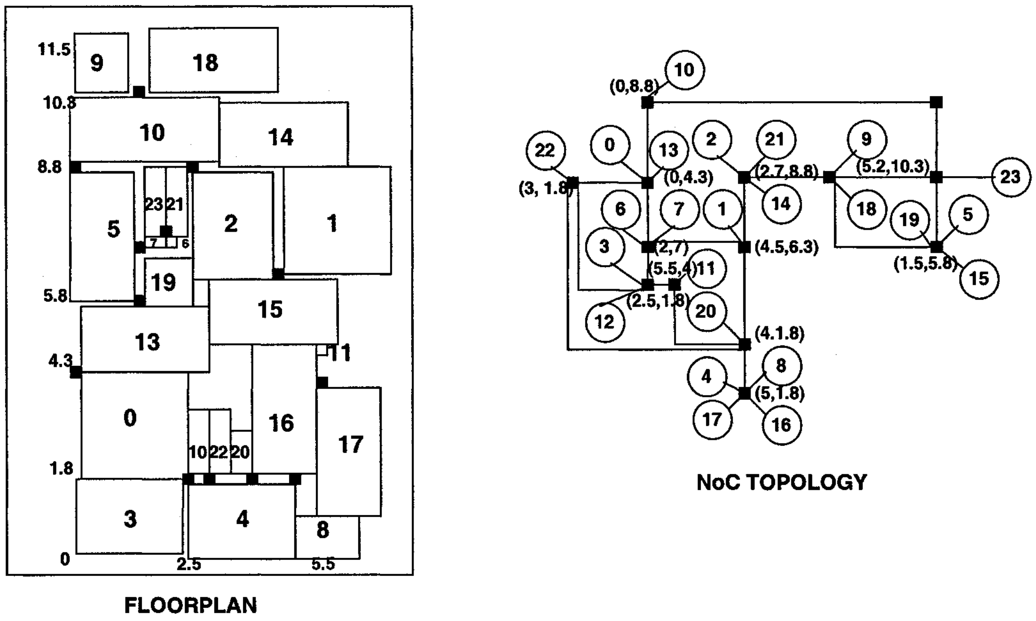

11.5
10.3
8.8
5.8
4.3
1.8
0
2.5
5.5
9
18
10
14
5
23
21
7
6
2
1
19
15
13
11
0
16
10
22
20
17
3
4
8
FLOORPLAN
(0,8.8)
10
22
0
13
2
21
9
(3, 1.8)
(0,4.3)
(2.7,8.8)
(5.2,10.3)
23
6
7
1
14
18
19
5
3
(2,7)
(4.5,6.3)
(5.5,4)
11
(1.5,5.8)
15
(2.5,1.8)
20
12
(4.1.8)
4
8
(5,1.8)
17
16
NoC TOPOLOGY

Fig. 6.15: Floorplan and NoC topology for NP

respectively. In the figures, the left hand side denotes the floorplan of the SoC, and the right hand side denotes the NoC topology. The black squares denote the routers. The floorplan is depicted by annotating the boundaries by X and Y co-ordinate values respectively. The routers in the NoC topology are annotated by the co-ordinates of their lower left hand side boundary. For example, if a router is annotated with values (2,2), the location of its lower left hand side corner is the co-ordinate $X = 2$ and $Y = 2$.

6.5. Conclusion

In this chapter, we presented approximation algorithms for the design of custom NoC architectures. We presented polynomial time optimal and factor-2 approximation algorithm for the core to router mapping, and topology generation stages, respectively. We also presented graph transformations and heuristic techniques such that the NoC consumes minimum power, minimum router resources, and satisfies the architecture specific port and bandwidth constraints on the routers. We demonstrated the superior quality of the solutions generated by our technique by experimenting with an optimal ILP formulation, and an existing heuristic called ANOC.

CHAPTER 7

GENETIC ALGORITHM BASED TECHNIQUE FOR NOC DESIGN

7.1. Introduction

In this chapter, we propose a genetic algorithm based iterative technique for routing and topology generation for NoCs. Please refer to Chapter 2 for a formal definition of the problem. A genetic algorithm searches the design space iteratively, and serves as a trade-off between the optimal ILP based technique that has a high runtime overhead, and low complexity heuristics that solutions in quick time, but at a lower quality. A GA models the problem as a set of genetic strings that undergo genetic operations namely, reproduction, crossover and mutation over successive generations to finally evolve into a final solution. The GA maintains several solutions in each generation (or iteration). Therefore, it handles multi-objective optimization problems well by generating a Pareto curve that provides the designer the ability to choose a solution that best suits his or her needs.

Our GA based algorithm models the problem as genetic strings in three levels of hierarchy. The genetic operators are applied probabilistically on the strings at each level of hierarchy over successive generations. On termination, the GA outputs a Pareto curve, where each point in the curve corresponds to a trade-off between power consumption, and corresponding routers in the NoC.

The results generated by the GA are compared with the ILP based technique, and the ANOC heuristic. The GA is able to generate solutions that are very close in quality to the optimal results generated by the ILP, in much less time than the ILP. Compared to the ANOC heuristic, although the GA has a higher runtime, it generates results that are consistently superior across all multimedia and network processing benchmarks. This chapter is organized as follows. Section 7.2 presents our GA based technique in detail, Section 7.3 presents results for multimedia and network processing benchmarks, and finally, Section 7.4 concludes that chapter.

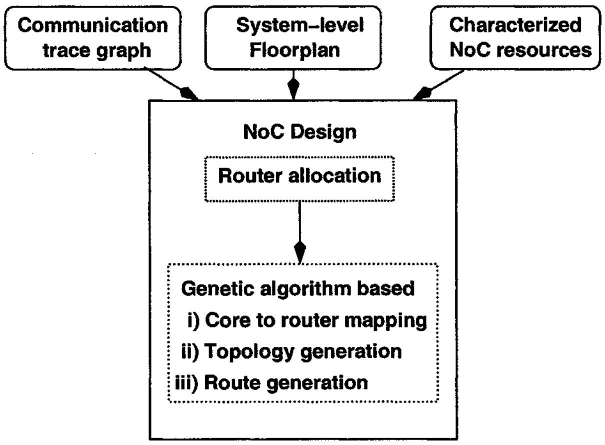

Communication trace graph
System–level Floorplan
Characterized NoC resources
NoC Design
Router allocation
Genetic algorithm based
i) Core to router mapping
ii) Topology generation
iii) Route generation

Fig. 7.1: Application specific NoC design flow

7.2. Genetic algorithm for NoC topology design and route generation

In this section we present our automated technique for topology design design and route generation of application specific NoC. The overall flow of application specific NoC design is shown in Figure 7.1 and an example is shown in Figure 7.2.

Our technique takes the communication trace graph, a system-level floorplan and interconnection architecture elements as input. As a first step, our technique allocates routers at physical locations in the floorplan. The selection of the router locations should reduce the design space of the GA, and thus its runtime. Similar to [12] the possible router locations are assumed to be at the corners of the computation cores as shown in Figure 7.2. As the possible physical locations of the routers are known, we can determine the shortest distance from any node to the routers. By the same argument, we can also determine all inter-router distances. Therefore, we can estimate the link lengths (and resulting link power consumption) in the NoC with a high degree of confidence. The physical link lengths of inter-router and node-router connections are constrained by the clock period of the core that is initiating the write operation. The designer can specify a maximum length of the physical link that permits the single clock cycle data transfer.

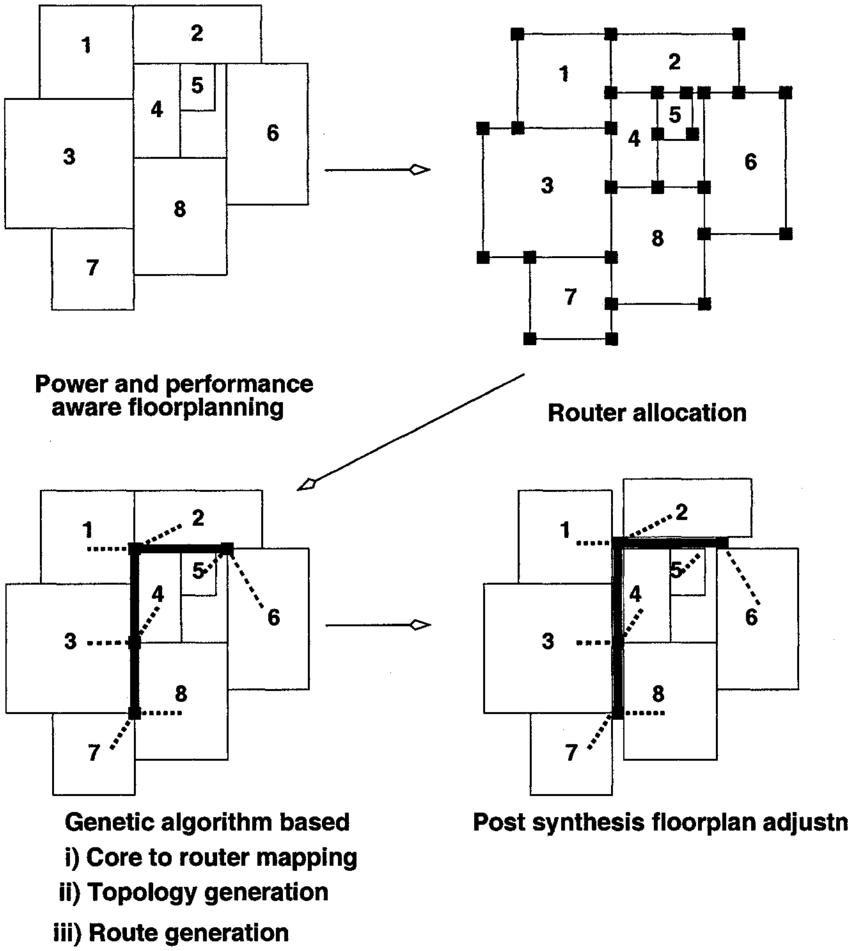

1
2
5
4
6
3
8
7
Power and performance aware floorplanning
Router allocation
Genetic algorithm based
i) Core to router mapping
ii) Topology generation
iii) Route generation
Post synthesis floorplan adjustn

Fig. 7.2: Example NoC design flow

As the next step, our GA based automated design technique selects the routers to be utilized in the NoC, maps the nodes to router ports, constructs the topology of the network, and routes the traffic traces on the interconnection architecture. The final layout in Figure 7.2 shows the node to router mapping by the dotted lines, and the NoC topology by the thick lines. During topology generation, the physical dimensions of the routers were neglected. At the end of the topology generation stage, we adjust the SoC floorplan such that the area of the routers is taken into account. However, since the router dimensions are small, we do not expect a significant variation between the floorplans before and after the router dimensions are taken into account.

The NoC architectures generated by our technique can result in deadlocks between various traffic traces. We address deadlock avoidance during NoC generation phase, as well as a post processing step. Our static routing algorithm utilizes the knowledge of the underlying router architecture to generate deadlock avoiding routes. In case deadlocks cannot be avoided, our post processing step adds virtual channels in suitable router ports to break the deadlocks [6] [43]. In the following section, we present our GA based interconnection architecture design technique. In the following sections we first give an overview of the GA, and then present the NoC topology generation, and deadlock avoidance techniques in detail.

7.2.1. Overview of GA

A GA is based on the biological phenomenon of genetic evolution. It maintains a set of solutions known as the population or a generation. GA operates in an iterative manner and evolves a new generation from the current generation by application of genetic operators. A new generation is created by first increasing the population by generating new individual solutions, and then selecting a constant number of solutions based on their fitness criteria. The fitness criteria is a cost function that captures the optimization goal. The selection of solutions based on their fitness criteria models the evolutionary behavior known as the survival of the fittest. A GA based technique typically applies three operators namely reproduction, crossover, and mutation to produce new members. Reproduction duplicates a solution across generations, crossover combines two solutions to generate two new solutions, and mutation modifies an existing solution to generate one new solution. The algorithm continues to operate in an iterative manner until the termination condition is reached.

المنارة للاستشارات

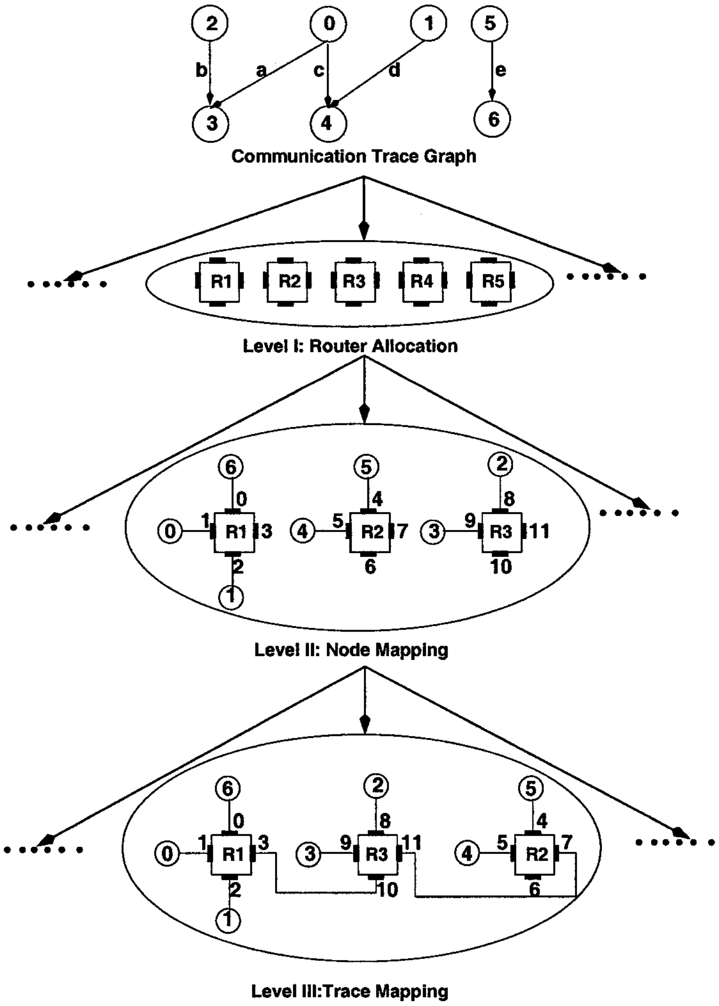

2
0
1
5
b
a
c
d
e
3
4
6
Communication Trace Graph
R1
R2
R3
R4
R5
Level I: Router Allocation
6
5
2
0
4
8
0
1
R1
3
4
5
R2
7
3
9
R3
11
2
6
10
1
Level II: Node Mapping
6
2
5
0
8
4
0
1
R1
3
3
9
R3
11
4
5
R2
7
2
10
6
1
Level III:Trace Mapping

Fig. 7.3: Hierarchical representation

7.2.2. Data structures for representation of solution

GA based optimization requires a representation of the population that supports efficient application of genetic operators. Our GA based technique models the population in a hierarchical manner consisting of three levels. The first level represents the number of routers, the second level denotes the mapping of the *CTG* nodes to the ports of the routers in the solution, and finally the third level specifies the routing of the communication traces from the source node to the sink node possibly via the ports of intermediate routers. Figure 7.3 shows the hierarchical representation of

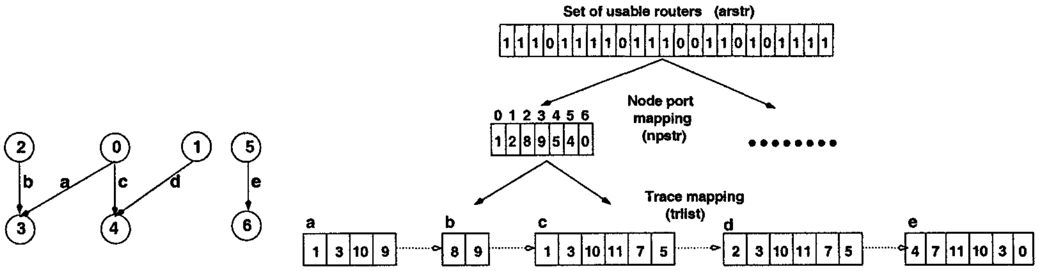


Fig. 7.4: Array based data-structure

the population in our GA based technique. At the first level, our technique maintains I different architectures with various number of routers in each architecture. At the second level, for each router specification at level 1, our technique saves J different node to port mappings. Finally, at the third level, our technique maintains K different mappings of the communication traces on the ports of the routers for every node to port mapping at level 2. Note that our technique does not explicitly model the physical links between the routers. Rather, the physical links are derived from the mapping of the communication traces. For the rest of the chapter we will refer to the 3 levels as router level or level 1, node level or level 2, and trace level or level 3, respectively.

7.2.2.1. *Router level data structure.* Application of the genetic operators requires the representation of the population as strings of chromosomes. Strings of chromosomes are represented by sets of arrays. Figure 7.4 depicts the chromosome representation of the solution shown in Figure 7.3. At the first level, the number of routers in a solution is represented by a binary array $arstr[MAXROUTER]$ where $MAXROUTER$ is the total number of routers in the architecture. Note that the location and number of routers are determined before the GA is invoked. $MAXROUTER$ is only the upper-limit on the routers that can be utilized. For the example shown in Figures 7.3 and 7.4, $MAXROUTER = 24$. We denote each router by r_i where i is an integer such that $0 \leq i \leq MAXROUTER - 1$. Each router that can be possibly utilized in the architecture is assigned a random location in the array given by $loc(r_i)$. For example, r_4 may be assigned to location *arstr*[0] ($loc(r_4) = 0$), r_2 may be assigned to location *arstr*[1] ($loc(r_2) = 1$), and so on. Finally, all ports of the maximum number of routers that can possibly be utilized in the architecture are assigned a unique number. Hence, the four ports of r_4 at location *arstr*[0] are numbered 0, 1, 2, and 3, the four ports of r_2 at location *arstr*[1] are numbered 4, 5, 6, and 7, and so on. The GA maintains I instances of array *arstr* at the first level. *arstr* is a binary array where a "1" in location "i" denotes that the router assigned to *arstr*[i] is possibly utilized in the architecture, and a "0" denotes that the router is not utilized. We say that the router is possibly utilized because even though the router is in the architecture, a communication trace may not be mapped to it.

7.2.2.2. *Node level data structure.* As mentioned earlier, for each instance of *arstr* array, the GA maintains J instances of node to port mapping. The node to port mapping at the second level is stored in an integer array given by *npstr*[$|V|$]. Location i contains the port number to which node $v_i \in V$ is assigned. In Figure 7.4, the string *npstr* represents one such node to port mapping where node 0 is mapped to port 1, node 1 is mapped to port 2, node 2 is mapped to port 8, and so on. Note that since each port can map only one node, a port is not repeated in the string.

Our technique can generate NoC topologies with heterogeneous routers that have variable number of input/output ports. The number of ports in a particular router are only known once the final solution has been generated. Therefore, our technique initially assumes a constant maximum number of ports at each router. The number of ports is given by the router with maximum number of ports in the router library. For example, if the maximum number of ports in a router that is available in the IP library are 8, the GA initially assumes that all routers have 8 ports. Once the topology has been generated, the required number of ports in each router is known, and the suitable IP-block can be selected from the component library. Thus, our NoC design topology supports heterogeneous router architectures.

7.2.2.3. *Trace level data structure.* Our GA maintains K instances of communication trace mappings for every instance of *npstr* array. The communication trace mapping is represented by a linked list *trlist* of integer arrays $trstr_i$, where the i^{th} array refers to the communication mapping of trace i. For example in Figure 7.4, the first $trstr$ array refers to trace a, second $trstr$ array refers to trace b and so on. Each $trstr$ array is an ordered set of ports indicating the flow of the traffic. For example, the communication trace a passing through ports 1, 3, 10, 9 in that order is represented by a array given by $[1, 3, 10, 9]$.

7.2.3. Legality criteria for solution representation

In this section, we present the necessary and sufficient legality criteria for solution representation at the router, node and trace level. A violation of the criteria results in an infeasible solution.

We note that a node is mapped to a unique port. Therefore, at the router level, the only legality criterion to be satisfied is that the total number of ports, which is given by the number of "1"s in the router level string multiplied by the number of ports per router, is greater than or equal to the number of nodes in the CTG.

At the node level, the following legality criteria must hold.

1. Since a port can map only one node, a port is assigned to one and only one location in *npstr*.
2. A port cannot belong to a router that is not included in the corresponding router level string. Therefore, all ports that are in *npstr* belong to a router r_i that is included in the corresponding router level string. That is, $arstr[loc(r_i)] = 1$.
3. The distance between the node to the port must be less than or equal to the maximum designer specified distance to enable single clock cycle data transfer.

The GA has to check for several legality conditions at the trace level. The conditions are enumerated below.

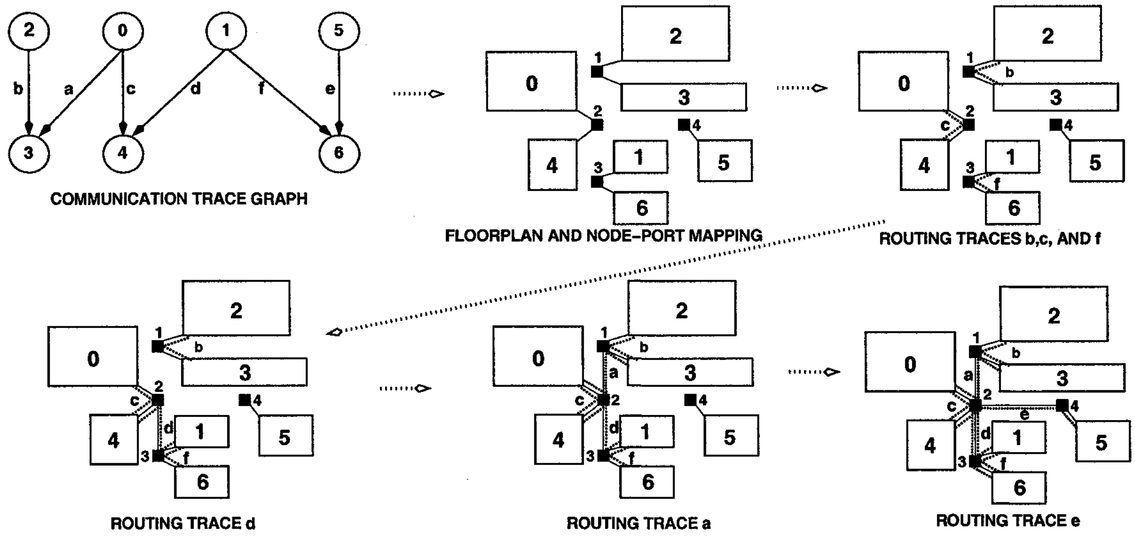
1. Since the traffic trace routes are represented by arrays, every traffic trace has an associated array.
2. The array starts at position 0 with the port to which the source of the trace is mapped, followed by a different port of the same router. The array ends with an integer that represents a port mapped to the sink node.
3. A port that is assigned to a node represents the origin or termination of traces that have that particular node as a source or sink, respectively. Therefore, no other port number in the array (except for the ports mapping source and sink nodes) represents a port mapped to a node.
4. If a trace enters a port of a router, it must leave the router through one of its output ports. In order to represent this constraint, we adopt the convention that even numbered positions in the array represent ports into which the trace enters, and odd numbered positions in the array represent ports from which the trace leaves.
5. From the previous constraint, it follows that if a trace enters a port, the next integer is the port number from which the trace leaves the same router.
6. In order to avoid cycles, a port number in the array is not repeated.
7. Since the route does not contain cycles, at most two ports of a router can appear in an array. Violation of the constraint will result in a router being visited more than once, thus resulting in a cycle. This constraint, along with constraints 4 and 5 makes sure that the number of ports belonging to the same router appearing in the trace array is either two or zero.

8. If a pair of ports adjacent to each other represent ports of two different routers, they should be consistent over all arrays. This pair represents two ports of different routers connected together by a physical link. This constraint forbids a port from being connected to multiple ports.
9. If a traffic is routed through a router, it must pass through two ports of that router. The two ports combined contribute toward one router hop. Therefore, the length of the array must be less than or equal to twice the latency constraint, where latency constraint is represented in number of hops.
10. The distance between two adjacent routers in the trace array must permit single clock cycle transfer of a word between the routers. As explained earlier the designer specifies the maximum distance between two routers for single clock cycle transfer. Therefore, this constraint can be satisfied if the length of the physical link between two routers is less than the designer specified distance.
11. The bandwidth constraint at the ports included in the array is not violated.

A communication trace mapping is legal if it contains legal arrays corresponding to all traffic traces.

7.2.4. Generation of initial population and modified shortest path algorithm (MSP)

In this section, we discuss the algorithms utilized for generation of initial population. An initial population is obtained by generating I router allocation arrays, J node to port mapping arrays for each router allocation, and K communication trace mapping arrays for each node mapping. The initial router allocation is generated by uniformly random assignment of “0”s and “1”s to all locations of every instance of *arstr* array. Similarly, the node to port mapping is obtained by uniformly random selection of a node of the CTG and mapping it to a port which



COMMUNICATION TRACE GRAPH

FLOORPLAN AND NODE-PORT MAPPING

ROUTING TRACES b,c, AND f

ROUTING TRACE d

ROUTING TRACE a

ROUTING TRACE e

Fig. 7.5: Communication trace mapping

is also selected by a uniformly random function. The node to port assignment is subject to the legality criterion described in the previous section.

The generation of an initial communication trace mapping is more involved. Initially, a trace is selected at random, and a modified shortest path algorithm (MSP) is applied to generate the traffic mapping from the source to the sink node, respectively. As mentioned earlier, a trace mapping is denoted by an array of port numbers. Every trace that is mapped through ports of multiple routers establishes a physical link between two consecutive ports of different routers. MSP differs from classic shortest path since it maps the traffic along the shortest route subject to the links established by already mapped traces, and the bandwidth constraints on the ports of the routers. Moreover, it generates traffic routes such that they do not cause a deadlock in the network. In the following paragraphs, we present the algorithm in detail. Deadlock avoidance will be discussed later in Section 7.2.9.

Since the MSP algorithm finds routes with minimum power consumption, the distance between any two routers is measured in terms of link and router power consumption. Given two

```
Modified Shortest Path
begin
  mark_all_traces_free()
  while free_traces_available()
    t = get_free_trace()
    $r(t)$ = shortest_path(t)
    if ($r(t) \neq$ valid)
      unmap($r(t)$)
    else
      update_physical_links()
    end-if
    tag(t)
  end while
end
```

Fig. 7.6: Pseudo code for MSP algorithm

routers $r1$ and $r2$, the cost of establishing a path from $r1$ to $r2$ for a trace t is given by

$$c_{(r1,r2)} = \begin{cases} (\Psi_i + \Psi_o + dist(r1, r2) \times \Psi_l) \times \omega(t), \\ if\ dist(r1, r2) \leq D_{max}\ and\ no \\ bandwidth\ or\ port\ violations\ in\ r1\ and\ r2 \\ \infty,\ otherwise \end{cases}$$

where $dist(r1, r2)$ is the Manhattan distance between routers $r1$ and $r2$, and D_{max} is the maximum allowable distance between two routers to ensure single clock cycle data transfer. Therefore, the cost function establishes the following two properties.

- A shortest path directly corresponds to a path with minimum power consumption.
- If routers $r1$ and $r2$ are further than the maximum allowed inter-router distance apart, or establishing the route results in bandwidth or port violations in $r1$ or $r2$, the cost is set to ∞ and hence, the trace is not routed through that link.

Consider the CTG and initial node to port mapping shown in Figure 7.5. The flow of the algorithm is denoted by the dotted arrows. The communication trace mapping is generated by selecting a trace at random, for example “b”. We show the trace “b” by a dotted curve since it

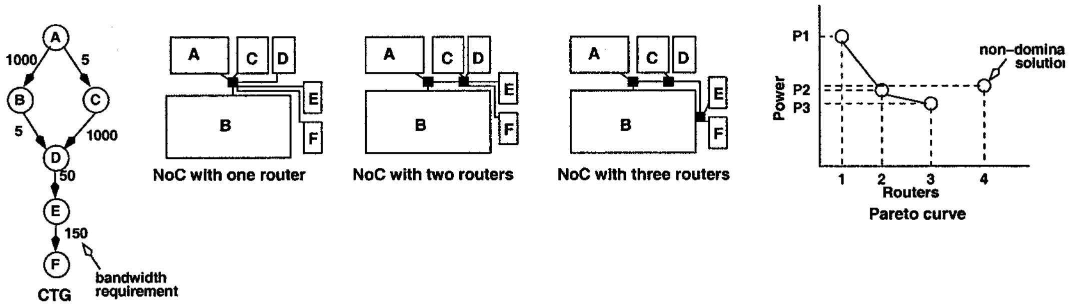

A
1000
5
B
C
5
1000
D
50
E
150
F
bandwidth requirement
CTG
A
C
D
E
B
F
NoC with one router
A
C
D
E
B
F
NoC with two routers
A
C
D
E
B
F
NoC with three routers
P1
P2
P3
Power
non-domina
solutio
1
2
3
4
Routers
Pareto curve

Fig. 7.7: Pareto points for NoC generation

is routed through just one router. Similarly, traces "c" and "f" are also shown by dotted curves. The next trace that is selected is "d". As the source (node 1) and sink (node 4) of trace "d" are mapped to different routers, we invoke the MSP. The distance between source and sink nodes of a communication trace is given as the path from source to sink with minimum power consumption, subject to the previously mapped traces, and bandwidth constraints on the ports. Since, no inter-router trace has been mapped, traffic "d" is mapped by the shortest path. Mapping of trace "d" effectively establishes a link between the routers to which nodes 1, and 4 are mapped. In other words, routers $r2$ and $r3$ are connected by a physical link. The next trace that is picked is "a", and its routing establishes a physical link between the routers mapping the corresponding nodes, which are routers $r2$ and $r1$. Finally, trace "e" is routed. First, it is routed from router $r4$ to router $r2$ since it is the only router that lies within the maximum allowed distance for single clock cycle data transfer. This establishes a link between routers $r2$ and $r4$. The remaining part of the route is generated by utilizing the already existing link between routers $r2$ and $r3$.

It is possible that MSP is unable to find a path from the source to sink, or the available path violates the latency constraint. In such a case, the traffic trace is left un-mapped, and a penalty is accrued as described in the discussion on the fitness function in the following section.

Figure 7.6 presents the MSP algorithm. Line 1 is the initialization phase where all traces are marked "free". The algorithm iterates until "free" traces are available. In each iteration of the loop, it obtains a random free trace (line 3), and attempts to obtain a shortest path for the trace. If a valid route is not found, the trace is unmapped (line 6). On the other hand, if a valid route is found, the corresponding physical links are updated to accommodate the route (line 8). At each iteration, the selected trace is tagged, such that it is not explored again.

7.2.5. Pareto points and fitness calculation

Our technique generates the Pareto points on the power consumption versus router requirement plot for a given application. For example, Figure 7.7 depicts three Pareto points corresponding to the solutions with least power consumption for NoC topologies with one, two and three routers, respectively. We observe that it is prudent to choose a solution with two routers, over a solution with one router that results in high power consumption, and a solution with 3 routers that results in a small reduction in power consumption. Given the Pareto curve for a particular application the designer can select the solution that offers the best trade-off between power consumption and router requirement.

In order to generate the Pareto curve, we define the well known concept of dominant solutions in the context of multi-objective optimization [8]. In the GA, a solution Z^* is said to dominate Z if Z^* is better than Z in all objectives. In Figure 7.7, solution 3 dominates solution 4, as 3 has lower power consumption, as well as less number of router resources. A solution is non-dominant if there exists a solution that dominates it. In the example shown in Figure 7.7, the solution with 4 routers consumes more power than the solution with 3 routers, and hence, is not part of the Pareto curve. At each generation, corresponding to each router, our GA maintains the legal solution (with legal routes for all traces) that consumes least power. On termination, the GA outputs the Pareto curve obtained from the set of solutions belonging to the last generation.

The size of the population is larger than the number of Pareto points. Therefore, in addition to the Pareto points, the GA also maintains non-dominant solutions for each generation. A non-dominant solution is selected on the basis of its fitness. The fitness calculation of each solution includes the area under the solution on the power consumption versus router requirement plot. The area under the solution is given by the product of the projections on the X and Y axes, respectively. Higher the area under the solution, lower the fitness of the solution, and vice versa.

In a particular generation not all members of the population represent solutions in which all traces are routed. Such solutions are denoted as illegal solutions. An illegal solution can occur if the MSP is unable to route all traces successfully. As mentioned earlier, a dominant solution is a legal solution. A non-dominant solution can be an illegal solution.

The overall fitness of a non-dominant solution is given by

$$fitness(solution) = \frac{1}{r \times p + \gamma * u}$$

where r is the number of routers in the topology, p is the power consumption, γ is the weight given to unmapped traces, u is the number of unmapped traces. The value $(r \times p)$ denotes the area of the solution on the power consumption versus router requirement plot. The value of γ is set very high to effectively pull down the fitness of illegal non-dominant solutions.

During the evolution of the next generation of a population, the GA first selects the dominant solutions and then selects the non-dominant solutions based on their fitness.

7.2.6. Overview of the optimization technique

Figure 7.8 shows the top level flowchart of our GA based optimization technique. The input to the technique is the set of router architectures, the communication trace graph $G(V, E)$, and the system-level floorplan. An initial population of solutions is generated using the algorithms described in Section 7.2.2, and the fitness of each trace level string (solution) is calculated. Initially, the exit criterion is set to false. Our technique applies genetic operators at the three levels of solution hierarchy with different probabilities. For each genetic operation at a higher level of

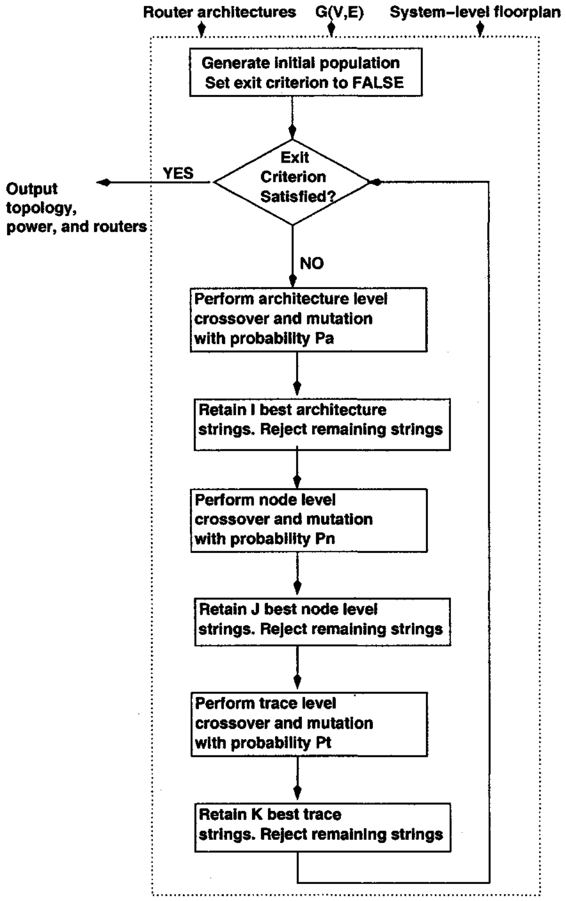

Router architectures
G(V,E)
System-level floorplan
Generate initial population
Set exit criterion to FALSE
Exit
Criterion
Satisfied?
YES
Output
topology,
power, and routers
NO
Perform architecture level
crossover and mutation
with probability Pa
Retain I best architecture
strings. Reject remaining strings
Perform node level
crossover and mutation
with probability Pn
Retain J best node level
strings. Reject remaining strings
Perform trace level
crossover and mutation
with probability Pt
Retain K best trace
strings. Reject remaining strings

Fig. 7.8: GA for custom NoC design

hierarchy, the GA explores several operations at the lower levels. This approach aids in a structured design space exploration for the problem. At each level the number of solutions produced by crossover is much larger than those produced by mutation. At each hierarchical level new individual solutions are produced by the application of the genetic operators. A new generation is produced by selection of the M fittest members among the current generation and the new individual solutions, where $M = I$ for level 1, $M = J$ for level 2, and $M = K$ for level 3. Selection of members of current generation for the next generation models the reproduction operation. At all three levels of the hierarchy, the fitness is given by the strongest complete solution at the trace level. Since each router allocation has J instances of node mapping and every node mapping has

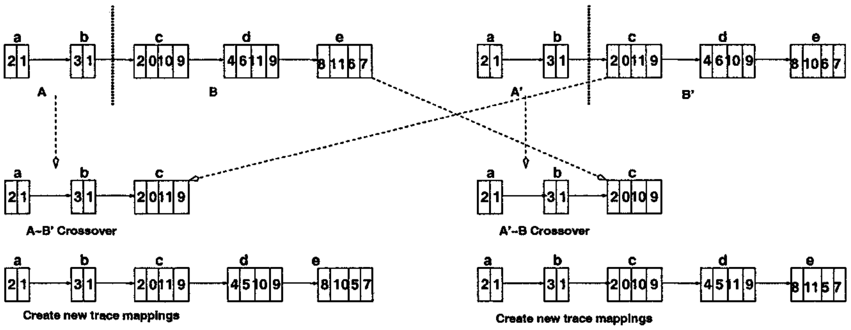

a b c d e
2 1 | 3 1 | 2 0 10 9 | 4 6 11 9 | 8 11 6 7
A
B
a b c d e
2 1 | 3 1 | 2 0 11 9 | 4 6 10 9 | 8 10 6 7
A'
B'
a b c
2 1 | 3 1 | 2 0 11 9
A–B' Crossover
a b c
2 1 | 3 1 | 2 0 10 9
A'–B Crossover
a b c d e
2 1 | 3 1 | 2 0 11 9 | 4 5 10 9 | 8 10 5 7
Create new trace mappings
a b c d e
2 1 | 3 1 | 2 0 10 9 | 4 5 11 9 | 8 11 5 7
Create new trace mappings

Fig. 7.9: Trace level crossover

K instances of trace mapping, the fitness of a particular router allocation (at level 1) is given by the strongest trace among the $J \times K$ possible traces level mappings. Termination condition is reached if the N successive generations do not result in any change in the pareto curve. We set N to summation of the number of nodes and edges in the CTG, that is $N = |V| + |E|$.

7.2.7. Genetic operators

In this section, we discuss the crossover and mutation operators that are utilized to produce new individuals from an old generation. The reproduction operator is implicitly applied during the creation of the new generation from the set of current generation, and new members. Since the solution is modeled at three levels, we will consider the application of these operators at three levels.

7.2.7.1. *Crossover operation.* The crossover operator selects two solutions or parents from the previous generation and produces two new solutions or children. In this section, we discuss the crossover operation of strings at the trace, node, and router levels.

Trace level crossover. The trace level crossover operation is applied to every set of K traces to generate K_{cross} new individuals. The trace level crossover operation is illustrated in Figure 7.9. Two trace mapping link lists belonging to the same node mapping are chosen randomly from the

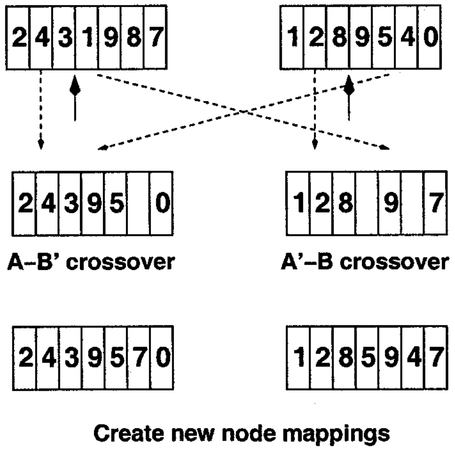


Fig. 7.10: Node level crossover

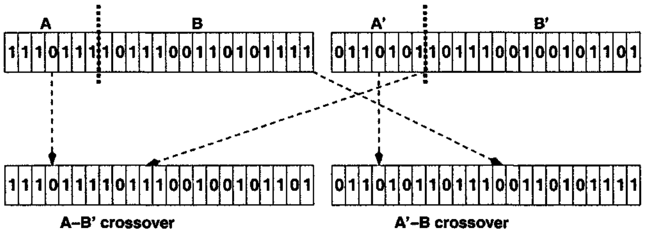


Fig. 7.11: Router level crossover

K linked lists. Each of the two linked lists is partitioned into two at the same randomly selected cut point. The cut point for the two lists is shown by upward arrow in the top of Figure 7.9. In the figure the linked lists are divided into $A-B$, and $A'-B'$ sub-lists, respectively. The crossover operator proceeds to create two new individual solutions by appending the lists as $A-B'$, and $A'-B$. As a convention, during the concatenation, the first half of the list (A or A') dominates the second half(B' or B). Therefore, based on the communication trace mapping of A, some trace mappings of B' may violate the legality criteria presented in Section 7.2.3. The violations that can occur are:

- It may not be possible to generate a physical route for the trace due to the previously imposed physical connections by traces belonging to A.
- Mapping the trace on the links may lead to a bandwidth violation in one or more ports of the routers.

If a communication trace mapping in B' is not legal due to a trace mapping in A, the particular communication trace is re-mapped by invoking the MSP algorithm. For example, in the left part of Figure 7.9, trace "d" is remapped as the route for trace "c" is modified by crossover. A communication trace is left un-mapped if the MSP is unable to generate a route from source to destination.

Node level crossover. The node level crossover operation is applied to every set of J mappings to generate J_{cross} new individuals. The node level crossover operation is illustrated in Figure 7.10. Two node level arrays belonging to the same router allocation are chosen at random from the J mappings. The two arrays are divided into two ($A - B$, and $A' - B'$ in the figure) at the same randomly selected position. The two arrays are concatenated as $A - B'$, and $A' - B$ such that a port is mapped to at most one node. The second legality condition for node level crossover (see Section 7.2.3) is automatically satisfied as the children and parent strings always belong to the same router level string. As in the case of trace level crossover, the mapping in A or A' dominates the mapping in B' or B. It is possible that the some of the nodes belonging to B (or B') cannot be mapped as the corresponding port in the parent string A (or A' respectively) maps a different node. In such cases, a new mapping for the node is generated randomly. For example, as shown in the figure, the mapping of node 5 in array $A - B'$ is initially left blank since port 4 is occupied by node 1. Hence, the mapping of node 5 is generated randomly. Similarly, the mapping of nodes 4 and 5 are randomly selected in the array $A' - B$.

Once the node level mappings are defined, the communication trace mappings are generated for the two new individual solutions. The technique duplicates as many communication trace mappings as possible from the parents' trace mappings to the children. The technique picks the first trace level strings of A and B, and duplicates as many routes as possible on the first trace level string of the child, such that the trace level legality constraints are not violated. The procedure is repeated for the remaining trace level strings of the parents. Since the node mapping of the children is not identical to that of the parents, some of the duplicated routes (trace mappings) for the traces may be illegal. These traces are not duplicated, and are instead regenerated by invoking the MSP algorithm.

Router level crossover. The router level crossover operation is applied to every set of I router allocations to generate I_{cross} new individuals. The mechanism is illustrated in Figure 7.11. Two router level allocations are chosen randomly from I arrays, and a random point is selected to divide each array into two. In the figure, the two parts have been shown as $A - B$, and $A' - B'$, respectively. Two new solutions are created by concatenating the arrays as $A - B'$ and $A' - B$, respectively. If the number of ports in the new string is less than $|V|$, the crossover is rejected, and the process is repeated with re-selection of router allocation arrays. After the crossover, node level and communication trace mappings are generated for the two individual solutions. As before, as many node level mappings and corresponding trace routes are duplicated on the children as possible. All node level mappings of sub-array A (and A') are duplicated. All node level mappings of sub-array B' (and B) are also duplicated subject to legality criteria for node mapping, presented in Section 7.2.3. Nodes whose mapping violated the legality criteria are randomly assigned to open ports. The trace mappings are generated as in the node level crossover.

7.2.7.2. *Mutation operation.* In this section, we discuss the mutation operation at the trace, node, and router levels, respectively. At each level, the mutation operator randomly selects a parent solution from the current generation, and randomly causes a small local change to generate a new individual solution.

Trace level mutation. The trace level mutation operation is applied to every set of K traces to generate K_{mutate} new individuals. The mutation operator is applied to every set of trace level mappings for every node level mapping in the population. The trace level mutation operator first selects a trace mapping at random from the K traces assigned to a particular node. At the trace level, there are some traces that are mapped to the architecture, and some are left unmapped due to violation of legality criteria. The mutation operator then selects a mapped trace at random and adds it to the set of unmapped traces. Next, it proceeds to randomly select an unmapped trace and map it to the architecture by invocation of the MSP algorithm. The process continues until as many unmapped traces are mapped as possible subject to the legal mapping constraints.

Node level mutation. The node level mutation operation is applied to every set of J mappings to generate J_{mutate} new individuals. The node level mutation is applied to every set of node mappings for every router allocation in the current generation. Two ports (u, v) in a particular router allocation are selected at random. A node mapping for the same router allocation is also selected at random. If the node mapping has either one or two nodes assigned to u or v, their mapping is exchanged. If the node mapping has only one node mapped to a port, say u, its mapping is changed to v. If the node mapping has no nodes assigned to either u or v, the process is repeated by selection of two new ports. Once a new node mapping is generated, all communication traces associated with the moved nodes are added to the set of unmapped traces in all trace mappings. The MSP algorithm is invoked for each communication trace mapping to map unmapped traces.

Router level mutation. The router level mutation operation is applied to every set of I router allocations to generate I_{mutate} new individuals. The router level mutation is applied to every router allocation in the current generation. As mentioned before the router allocation is specified by an array of binary digits. Router level mutation is applied by the selection of a random location in the array, and the inversion of the corresponding bit. If a '0' is inverted to '1' a router is added and no change is applied to the lower levels. On the other hand, if a '1' is inverted to '0' a router is removed. Hence, all node level mappings that contain any ports belonging to the removed router and associated communication traces are re-generated similar to the initial population creation.

7.2.8. Post-synthesis floorplan adjustment

During the interconnection architecture stage, we assume that the routers are located at the corners of the cores of the floorplan. After the NoC topology generation stage, we adjust the floorplan such that the actual size of the routers are taken into account. As mentioned before, the area occupied by the routers is very small, and our router architecture with 5 ports, FIFO depth of 16, width of 32, and 2 virtual channels consumed only 0.19 mm^2 in 65 nm technology. Therefore, this adjustment of the floorplan after NoC synthesis does not cause significant difference between the pre-synthesis and post-synthesis floorplans.

7.2.9. Deadlock avoidance

In a NoC, depending on the router architecture and the routing scheme, it is possible that deadlocks occur. In a deterministic routing scheme, deadlocks can be avoided by either designing the routing protocol to avoid deadlocks, or by the addition of virtual channels as a post processing step [65].

We exploit the underlying router architecture to modify our MSP router to find deadlock avoiding paths. If deadlocks cannot be avoided, the MSP nevertheless takes that path, and the deadlock is broken during post processing by the introduction of addition virtual channels. In the following sections, we first introduce the concept of channel dependency graphs (CDG), and their relationship with deadlocks [65]. This is followed by our technique for deadlock avoidance, and deadlock breaking.

7.2.9.1. *Deadlock avoiding routing.* Figure 7.12 depicts the router architecture utilized in this work. For simplicity, we depict a four port router. The detailed explanation of the architecture can be obtained from [12]. The router consists of separate input and output ports to route traffic to and from the router, thus isolating traffic entering and leaving a router port.

Fact 2 If traces are routed through only one path between two routers in the NoC topology in the forward as well as the return directions, the forward and return paths utilize separate input/output buffers, and therefore, no cycles are introduced in the channel dependency graph (CDG). Consequently, deadlocks do not occur. On the other hand, if traces are routed through different paths between the same pair of routers, there is a possibility of deadlock.

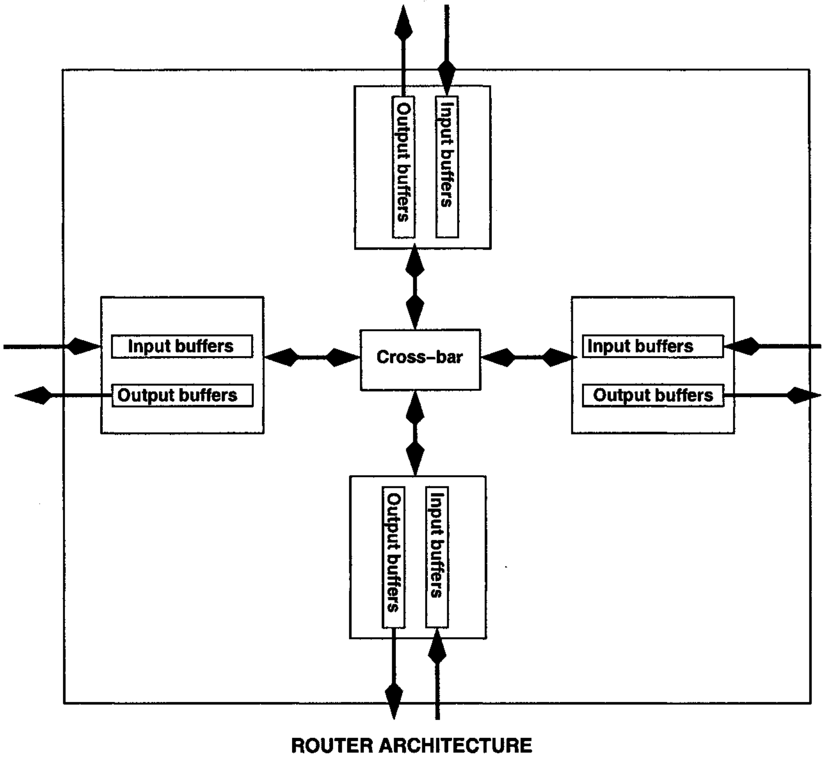

Output buffers
Input buffers
Input buffers
Output buffers
Cross-bar
Input buffers
Output buffers
Output buffers
Input buffers
ROUTER ARCHITECTURE

Fig. 7.12: Router architecture

This is illustrated in Figure 7.13 (A) and (B). The first routing technique uses different paths for routes to route trace $t2$ from $R2$ to $R4$, and trace $t4$ from $R4$ to $R2$, respectively. This can cause a deadlock as follows:

- Trace $t1$ holds link $R1 - R2$ and waits on link $R2 - R3$.
- Trace $t2$ holds link $R2 - R3$ and waits on link $R3 - R4$.
- Trace $t3$ holds link $R3 - R4$ and waits on link $R4 - R1$.
- Trace $t4$ holds link $R4 - R1$ and waits on link $R1 - R2$.

On the other hand, the second routing technique has the same forward and return path between two routers. Hence, it routes trace $t4$ along the path $R4, R3, R2$ as the $R2, R3, R4$ path was established while routing trace $t2$. Similarly, trace $t3$ is routed through the path $R3, R2, R1$.

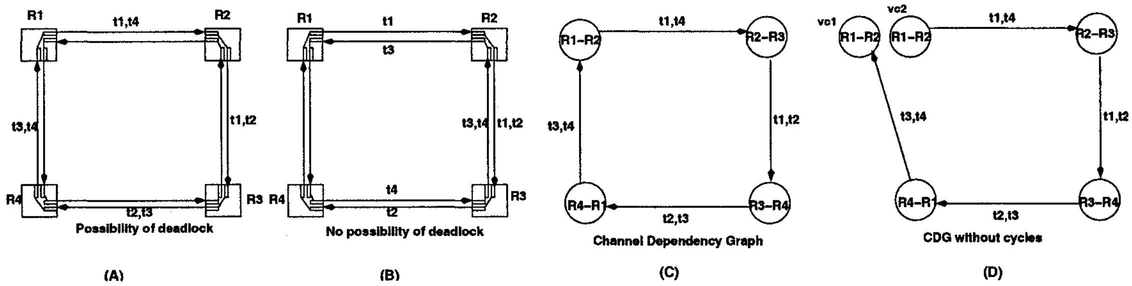

R1
R2
R3
R4
t1,t4
t3,t4
t1,t2
t2,t3
Possibility of deadlock
(A)
R1
R2
R3
R4
t1
t3
t3,t4
t1,t2
t4
t2
No possibility of deadlock
(B)
R1-R2
R2-R3
R3-R4
R4-R1
t1,t4
t3,t4
t1,t2
t2,t3
Channel Dependency Graph
(C)
vc1
vc2
R1-R2
R1-R2
R2-R3
R3-R4
R4-R1
t1,t4
t3,t4
t1,t2
t2,t3
CDG without cycles
(D)

Fig. 7.13: Breaking deadlocks by adding virtual channels

Since the forward and return paths utilize separate input/output buffers, the routes do not cause a deadlock.

The MSP algorithm utilizes the knowledge of already routed traces during route discovery for the current trace. Once a path between two routers is established as a result of routing of a previous trace, the MSP router tries to utilize the same path for all other routes in forward as well as in the reverse direction, subject to minimum power consumption, and bandwidth constraints. Alternative paths are explored only when the path causes a violation in the bandwidth constraints of the routers, or the path results in a higher power consumption. Therefore, in its effort to generate the shortest path trees, the MSP rejects as many deadlock causing routes as possible, and instead finds other deadlock free routes with the same power consumption.

7.2.9.2. *Breaking deadlocks with virtual channels.* Even though the MSP attempts to avoid deadlocks, in some cases they may not be avoidable due to bandwidth violations on the router ports. This is because, the bandwidth violations restrict the MSP from utilizing certain paths to route the traces. Since MSP is a deterministic custom routing algorithm, it provides us the opportunity to statically inspect the routes to determine which router ports can cause deadlocks. Additional virtual channels can be added to these ports such that the routing is deadlock free.

7.3. Results

In this section, we present the results obtained by execution of our technique on several multimedia benchmark applications. We compare the results generated by the GA with an optimal ILP based technique [12], and a recursive partitioning based heuristic called ANOC [15] that address the same problem. In Section 7.3.1 we discuss the benchmark applications, in Section 7.3.2 we discuss the experimental setup, in Section 7.3.3 we describe the GA tuning parameters, and finally in Section 7.3.4 we present and discuss the results.

7.3.1. Benchmark applications

We generated custom NoC architectures for five multimedia, and five network processing benchmarks. The details of the benchmarks are presented in Table 16. Benchmarks G1 through G5 are multimedia applications, and benchmarks G6 through G10 are network processing applications. The size of the benchmarks varied from 8 nodes and 20 edges to 25 nodes and 27 edges. The following is a brief description of the benchmarks.

- *JPEG Encoder:* A VHDL implementation of the JPEG Encoder core was obtained from the opencores website [75]. We simulated the VHDL model and generated the flow graph and the corresponding CTG for the application.
- *MPEG-4 decoder, MWD, and VOPD:* The MPEG-4, Video Object Plane Decoder (VOPD) and Multi Window Display (MWD) applications were obtained from the work presented by Jalabert et al. [32].
- *AH Auth-IPv4, and Diffserv-IPv4:* The process flow graph and the corresponding CTG for these applications were obtained from network processor implementations [76].
- *NP1, NP2, and NP3:* NP1, NP2, and NP3 are industrial strength network-processing benchmarks obtained from the work presented by Pasricha et al. [77].

For the benchmarks G1 through G7, the size of the ARM core was estimated to be $3 \times 3 = 9mm^2$ [78]. For benchmarks G8 through G10, the size of the cores were provided in [77]. The network interface, whose area is estimated to be $0.2mm^2$ [79] was included in the calculation of the core area. The router architecture utilized in the work consists of 2 virtual channels, FIFO depth of 16, and width of 32 bits. In 65 nm technology, the area of the router was estimated to be $0.19mm^2$ and $0.37mm^2$ for 5 port and 9 port routers, respectively.

7.3.2. Experimental setup

7.3.2.1. *Power models and floorplanner.* We characterized our router architecture in a 65 nm TSMC low power library. In this technology, the power consumption for the input and output port was estimated to be $204nW/Mbps$ and $94nW/Mbps$, respectively. The link power consumption was estimated to be $89nW/Mbps/mm$. We utilized the Parquet floorplanner [70] to obtain our floorplans.

7.3.2.2. *Number of ports per router and maximum link length.* Our experiments were guided by two parameters namely, maximum number of ports per router that can be synthesized such that the timing constraints are met, and the maximum link length that ensures a single clock cycle transmission. The IP library may either provide a hard router architecture core with a fixed number of ports, or a soft core with parameterizable number of router ports. If the router architecture is a hard IP, the NoC synthesis tool must take the number of ports of the router as a constraint. On the other hand, if the IP is a soft core, the constraint on the number of ports is not applicable. We present results when the number of ports per router is limited to 5 (hard router IP core), as well as when the number of ports is parameterizable (soft IP core).

For $65nm$ technology, we estimated the link delay to be $0.02ns/mm$. At an operating frequency of $333MHz$, this delay does not contribute significantly to the overall delay, and can be ignored. Benini et al. [7] predict that in the future, NoCs will be clocked at $5GHz$ or more. At

Graph	Graph ID	Nodes	Edges
JPEG Encoder	G1	8	21
MPEG-4 decoder	G2	12	13
MWD	G3	12	13
VOPD	G4	12	13
Set-top Box	G5	25	27
AH Auth-IPv4	G6	9	8
Diffserv-IPv4	G7	11	10
NP1	G8	15	22
NP2	G9	17	24
NP3	G10	24	42

Table 16: Benchmarks

this frequency, the router delay will be of the order of a tenth of a nano-second. The link delay is expected to stay at almost the same value of $0.02 ns/mm$ [7]. Thus, the link and router delay are comparable, and this puts a constraint on the maximum allowable link length between two routers. In order to exercise both the above mentioned cases, we experimented with and without link length constraints. For the constrained case, we set a maximum link length of $6mm$, and a corresponding link delay of 0.12 ns.

Table 17 assigns a unique name for the instances of our technique with and without link length constraints, and with and without port constraints, respectively. For example, an experiment with router architectures without any link length constraints and with maximum number of ports being 5 is given a name $GA_{\infty,5}$.

7.3.3. GA tuning parameters

We experimented with several population sizes of the GA. At the end of each generation, the population sizes were varied from 125 to 1000. A small population size limits the searchable solution space, and generates sub-optimal results albeit with lower runtimes. On the other hand, a large population size provides a large design space for exploration and therefore, generates better solutions at the cost of larger runtimes. Our experiments indicated that a population size of 1000 served as a good trade-off between design space available for exploration, and the runtime of the

Max. link length	Max. ports	Technique
6	5	$GA_{6,5}$
6	∞	$GA_{6,\infty}$
∞	5	$GA_{\infty,5}$
∞	∞	$GA_{\infty,\infty}$

Table 17: Technique nomenclature

GA. To enable an unbiased exploration of the design space at the three levels of hierarchy, the population sizes were maintained equally at each level. Therefore, a population size of 1000 was divided into 10 router level strings, 10 node level strings per router level string, and 10 traffic level strings per node level string.

In our experiments, we observed that if the pareto curve did not change for $|V|+|E|$ iterations, it did not change further even if the GA was executed for several more iterations. Therefore, we set a termination condition for our technique to be such that the the pareto curve does not change for $|V|+|E|$ iterations.

We experimented with different architecture, node and traffic level crossover probabilities. We obtained best results when the trace level crossover probability was higher than the node level crossover probability, which in turn was higher than router level crossover probability. For our experiments, we set the router level crossover probability at 0.1, node level probability at 0.5, and trace level probability at 1. The value of γ used in the calculation of fitness of the solution was set to a very high value. For our experiments, we set the value to 1000000. Hence, invalid solutions had a very poor fitness, and were rejected in subsequent generations of the GA.

7.3.4. Results and discussion

In this section, we first present a comparison of our technique with the two existing techniques, present NoC designs produced by our technique, and finally present simulation results for the JPEG encoder benchmark.

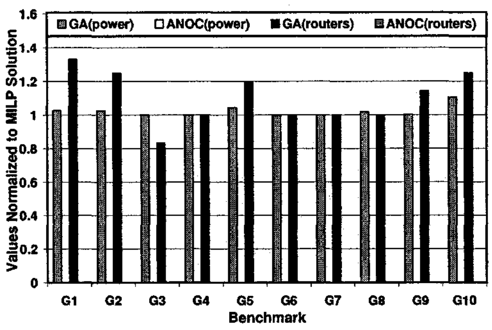


Fig. 7.14: Comparison between $GA_{6,5}$, $ILP_{6,5}$ and $ANOC_{6,5}$ corresponding to minimum power consumption. ANOC caused port violations for all benchmarks

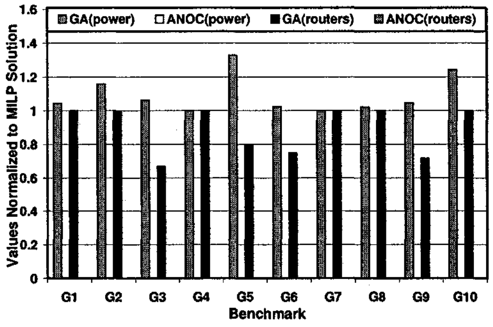


Fig. 7.15: Comparison between $GA_{6,5}$, $ILP_{6,5}$ and $ANOC_{6,5}$ corresponding to solution with minimum router resource consumption. ANOC caused port violations for all benchmarks

7.3.4.1. *Comparison with existing techniques.*

Description of existing techniques. We compared the results produced by our GA with an optimal ILP based technique [12], and a heuristic technique called ANOC [15] that address the same problem. The ILP based technique formulates the problem as an objective function to be minimized under a set of constraints. The technique has an exponential runtime complexity, and takes several hours to generate optimal solutions for many benchmarks. We obtained optimal results by letting the ILP run until completion.

ANOC is a technique that operates on the given system-level floorplan, and invokes a recursive bipartitioning based heuristic to generate the final NoC. Both ILP and ANOC are single objective techniques that minimize power consumption. Further, the ANOC technique does not consider constraints on the number of ports in the router.

Comparisons for $GA_{6,5}$, $GA_{6,\infty}$, $GA_{\infty,5}$ and $GA_{\infty,\infty}$. Figures 7.14 - 7.21 present the results for the four different cases illustrated in the experimental setup section. The bars in the figures are normalized to the corresponding results produced by the optimal ILP based technique. The first

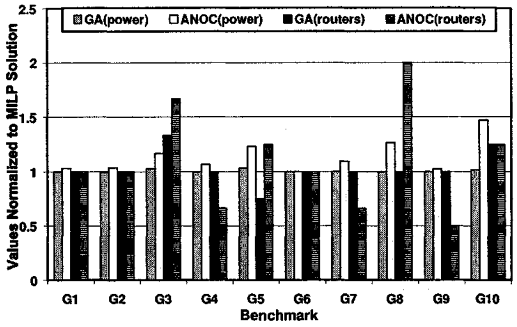


Fig. 7.16: Comparison between $GA_{6,\infty}$, $ILP_{6,\infty}$ and $ANOC_{6,\infty}$ corresponding to minimum power consumption

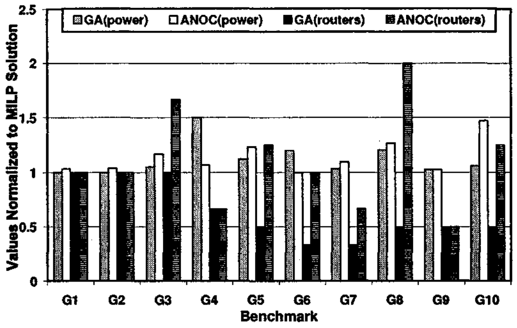


Fig. 7.17: Comparison between $GA_{6,\infty}$, $ILP_{6,\infty}$ and $ANOC_{6,\infty}$ corresponding to solution with minimum router resource consumption

bar denotes the power consumption of the GA, the second bar denotes the power consumption of ANOC, the third bar denotes the router resource consumption of GA, and finally, the fourth bar denotes the router resource consumption of ANOC.

Figures 7.14 and 7.15 present the comparison of our technique with ILP and ANOC based techniques under a port constraint of 5, and a maximum link length of $6mm$. Figure 7.14 presents the case when the solution picked from the pareto curve corresponds to the one with least power consumption. Figure 7.15 presents the case when the solution corresponds to the one with least number of routers. Since the ILP based technique optimizes power consumption and does not optimize router resource consumption, our technique consumes lesser number of routers than the ILP based technique for some cases. Since ANOC does not consider port constraints, it failed to generate valid solutions for many cases. The bars corresponding to the benchmarks for which ANOC failed are not plotted in the figures.

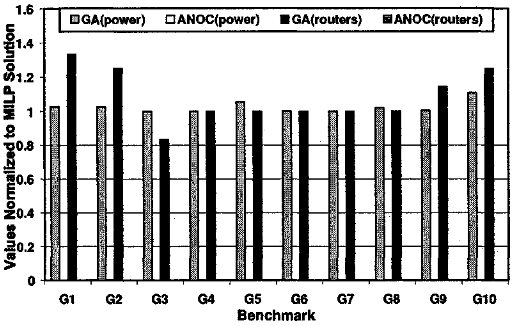


Fig. 7.18: Comparison between $GA_{\infty,5}$, $ILP_{\infty,5}$ and $ANOC_{\infty,5}$ corresponding to minimum power consumption. ANOC caused port violations for all benchmarks

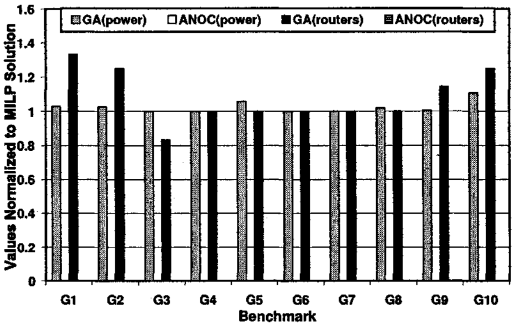


Fig. 7.19: Comparison between $GA_{\infty,5}$, $ILP_{\infty,5}$ and $ANOC_{\infty,5}$ corresponding to solution with minimum router resource consumption. ANOC caused port violations for all benchmarks

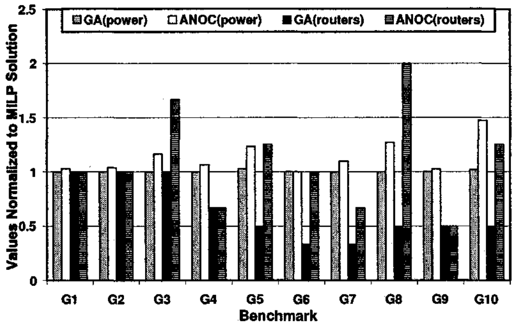


Fig. 7.20: Comparison between $GA_{\infty,\infty}$, $ILP_{\infty,\infty}$ and $ANOC_{\infty,\infty}$ corresponding to minimum power consumption

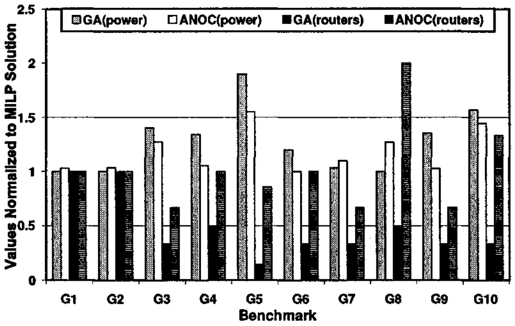


Fig. 7.21: Comparison between $GA_{\infty,\infty}$, $ILP_{\infty,\infty}$ and $ANOC_{\infty,\infty}$ corresponding to solution with minimum router resource consumption

	Least power solution				Least routers solution			
	Power		Routers		Power		Routers	
	$\frac{GA}{ILP}$	$\frac{GA}{ANOC}$	$\frac{GA}{ILP}$	$\frac{GA}{ANOC}$	$\frac{GA}{ILP}$	$\frac{GA}{ANOC}$	$\frac{GA}{ILP}$	$\frac{GA}{ANOC}$
$GA_{6,5}$	1.02	NA	1.08	NA	1.08	NA	0.89	NA
$GA_{6,\infty}$	1.00	0.89	1.03	1.09	1.11	0.99	0.63	0.65
$GA_{\infty,5}$	1.02	NA	1.13	NA	1.05	NA	0.93	NA
$GA_{\infty,\infty}$	1.00	0.87	0.96	1.12	1.27	1.05	0.48	0.5

Table 18: Average power and router consumption

Runtime (secs)			
Graph size	GA	ILP	ANOC
5	$\sim$ 5	~ 5	< 1
15	$\sim$ 100	> 42100	< 1
25	$\sim$ 1000	> 42100	< 1

Table 19: Runtimes

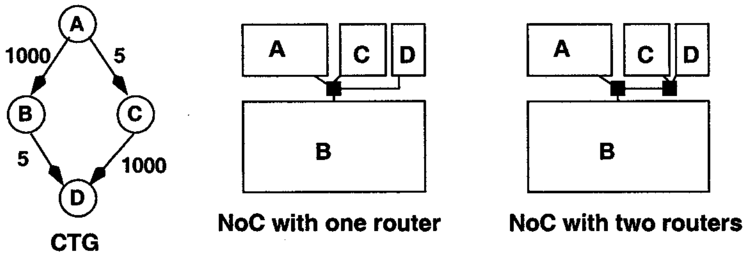


Fig. 7.22: Minimizing power consumption by introducing additional routers

Figures 7.16 and 7.17 present similar results for no port constraints and a link length constraint of $6mm$, Figures 7.18 and 7.19 present the corresponding results for a port constraint of 5, and no link constraints, and Figures 7.20 and 7.21 present results for no port constraint and no link length constraints, respectively.

When no port constraints and no link length constraints are imposed, it is possible to generate a solution trivially by connecting all the cores to a single router. However, due to the significant contribution of link power consumption, a solution with only one router may not always be the optimal solution. Consider an example shown in Figure 7.22. Placing just one router, and connecting all cores to that router results in long link lengths and link power consumption due to the edge (C, D), which has a high bandwidth requirement. This can easily be offset by introducing an additional router such that link lengths corresponding to the routes of highly communicating cores (edge (C, D)) are minimized.

Summary of results for comparison with existing techniques. Table 18 presents the average power, and router resource consumption comparisons for our technique versus the ILP and ANOC techniques respectively, for each of the four cases. In the table, some of the comparisons of GA with ANOC have been marked as N.A indicating that ANOC failed to generate valid solutions, and hence, the comparison is not applicable.

The low overall deviation in power consumption reinforces our claim on the quality of the solutions generated by the GA. The slightly lower power consumption of the ILP comes at a much higher runtime.

ANOC is a low complexity heuristic, and does not explore a large design space before arriving at the final solution. On the other hand, the GA explores several solutions as it evolves through successive generations. As a result, the GA is able to out-perform ANOC for all benchmarks.

Table 19 presents the comparison of runtimes of the three techniques. It took the ILP several hours to converge to the optimal solution. On the other hand, the GA converged to its best solution in a few minutes. Due to its low complexity, ANOC was able to generate solutions within one second for all benchmarks. Although the GA takes more time to generate solutions, its solution quality is better than ANOC.

In order to demonstrate the scalability of the technique, we also experimented with synthetic benchmarks whose sizes were varied between 30 nodes edges to 50 nodes and edges. Our technique was able to generate solutions to 50 node benchmarks in about one hour.

7.3.4.2. *Comparison of pre and post synthesis floorplans.* In this section, we compare our approach of synthesizing the NoC on the system-level floorplan with the traditional approach that synthesizes the NoC and then invokes the floorplanner to generate the layout. We compared the results for the four cases, $GA_{6,5}$, $GA_{6,\infty}$, $GA_{\infty,5}$ and $GA_{\infty,\infty}$. In each of the four cases, we utilized the Parquet floorplanner to generate the system-level layout, and our GA based technique

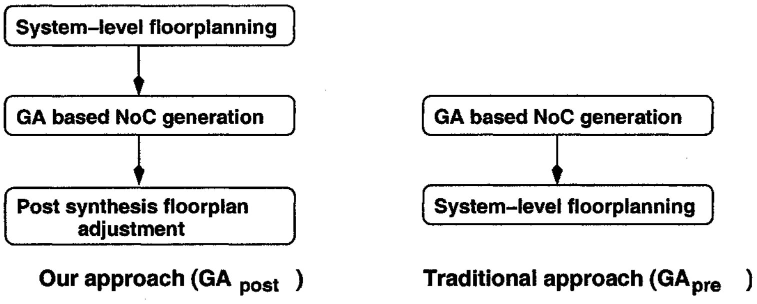

System–level floorplanning
GA based NoC generation
Post synthesis floorplan adjustment
Our approach (GA post)
GA based NoC generation
System–level floorplanning
Traditional approach (GApre)

Fig. 7.23: Our approach versus existing approach

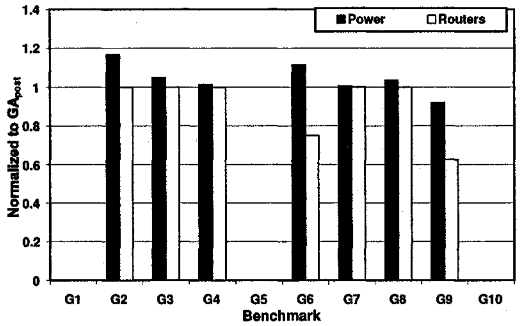


Fig. 7.24: Comparing GA_{pre} and GA_{post} for $GA_{6,5}$. Existing approach caused link length violations for benchmarks 1, 5 and 10.

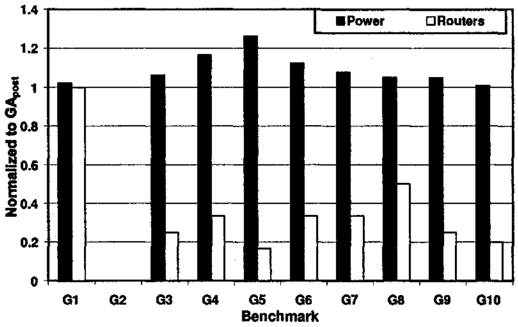


Fig. 7.25: Comparing GA_{pre} and GA_{post} for $GA_{6,\infty}$. Existing approach caused link length violations for benchmark 2.

for NoC architecture generation.

Figure 7.23 contrasts our approach with existing approach. Our approach invokes a system-level floorplanner, followed by the GA for NoC topology generation, and finally, floorplan adjustment to generate the final NoC with floorplan. The traditional approach generates the NoC topology by invoking the GA, followed by a call to the floorplanner to obtain the final NoC with floorplan.

Comparison for $GA_{6,5}$. Figure 7.24 presents the comparison of power consumption and router resources between our approach and existing approach. The bars in the figure are normalized to the solutions generated by our approach. The existing approach generates the interconnection

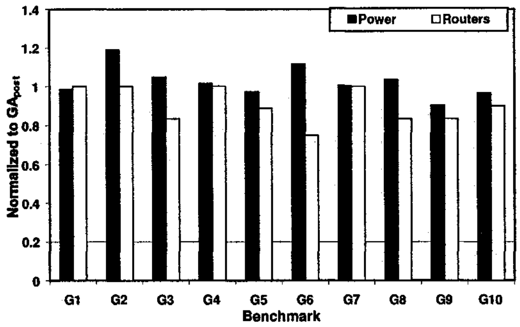


Fig. 7.26: Comparing GA_{pre} and GA_{post} for $GA_{\infty,5}$

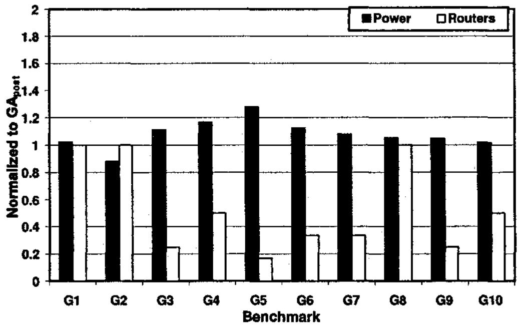


Fig. 7.27: Comparing GA_{pre} and GA_{post} for $GA_{\infty,\infty}$

architecture without any knowledge of the floorplan, and then floorplans the NoC such that the link power consumption is minimized. Since the NoC architecture generation phase has no information about the link lengths, the final solution may have long links that violate the link length constraints. In our experiments, the link lengths were violated for benchmarks G1, G5 and G10. On the other hand, our approach generates the NoC with the knowledge of the link lengths, and therefore, is able generate valid solutions for all benchmarks.

Comparison for $GA_{6,\infty}$. Figure 7.25 presents the comparison of power consumption and router resources when no port constraints are imposed. As before, the link length constraint is maintained at $6mm$. Since link lengths and corresponding link power consumption was not taken into consideration during the synthesis stage, the traditional approach trivially generated solutions with just one router. This resulted in 1.09 times the power consumption compared to our approach. Moreover, upon floorplanning, the traditional approach violated the link length constraints for the MPEG-4 decoder benchmark.

Comparison for $GA_{\infty,5}$. Figure 7.26 presents the comparison of power consumption and router resources when port constraint of 5 was imposed, and no link length constraints were imposed. In this case, the existing approach was able to perform comparably to our approach. The imposition of port constraints clusters the cores such that highly communicating cores communicate through one router in the cluster. The absence of link length constraints allows long links and hence generates valid solutions.

Comparison for $GA_{\infty,\infty}$. In this case, neither port constraints nor link length constraints were applied. As a result, the existing approach generated solutions with just one router, which is trivially the best solution when link power consumption is not considered. But when the floorplanner was invoked, long link lengths resulted in larger power consumption. On the other hand, due to the knowledge of the floorplan, our approach was able to optimize the link power consumption, and introduce additional routers to generate solutions with lower overall power consumption. Figure 7.27 presents the results for this case. Our approach consumed less power than the existing approach in 9 of the 10 cases.

Summary of results for comparison with post synthesis floorplanning approach. The comparisons presented in the above paragraphs demonstrate the need for integrating system-level floorplanning in the NoC design flow. The results for $GA_{6,5}$ and $GA_{6,\infty}$ demonstrate that a floorplan agnostic NoC synthesis stage followed by a call to an existing floorplanner results in long link lengths that violate the link length constraint. When neither port constraint nor link length constraints are imposed, NoC synthesis without link length information results in a trivial solution consisting of only one router. We demonstrated that due to increasing contribution of link power consumption, a design with only one router results in large overall communication consumption.

7.3.4.3. *NoC designs.* We present NoC designs produced by our GA for three benchmarks, a set-top box application (benchmark G5), network processing application (benchmark G10), and JPEG encoder application (benchmark G1), respectively. Figure 7.28, depicts the CTG for JPEG encoder application. We refer the reader to [52] for the CTG of the set-top box application, and to [77] for the CTG of the network processing application. Figure 7.29 presents the description of the nodes in the three CTGs.

The custom topologies of the three benchmarks produced by our GA based technique with

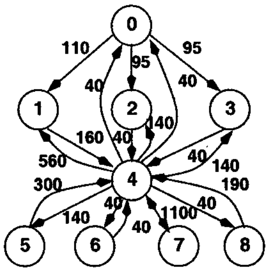


Fig. 7.28: CTG for JPEG encoder

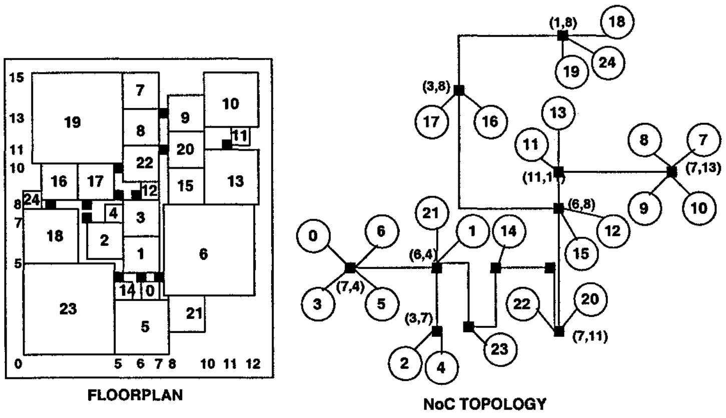
ID	Set-top Box	NP	JPEG	ID	Set-top Box	NP
0	ASIC	ARM	RGB	13	MEM	MEM
1	DSP	ARM	Y Buf	14	ASIC	MEM
2	DSP	ASIC	CR Buf	15	DSP	MEM
3	DSP	ASIC	CB Buf	16	DSP	MEM
4	ASIC	ASIC	JPEG	17	DSP	MEM
5	CPU	DMA	DCT	18	CPU	MEM
6	MEM	Watchdog	Q-hdr	19	MEM	SDRAM
7	DSP	UART	Huf.buf	20	DSP	ACC1
8	DSP	ITCI	Q-buf	21	DSP	NI-1
9	DSP	SDRAM		22	DSP	NI-2
10	CPU	MEM		23	MEM	NI-3
11	ASIC	TIMER		24	ASIC	
12	ASIC	SlvAC				

Fig. 7.29: Node description for set-top box, NP, and JPEG

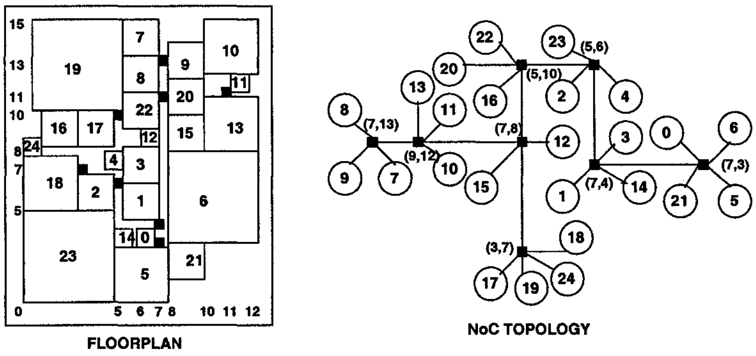
5 port routers and a link length constraint of $6mm$ are shown in 7.30, 7.31, and 7.32 respectively. The figures depict the NoC topologies corresponding to least power solution, and least router solution of the pareto curve, respectively. In the figures, the left hand side denotes the floorplan of the SoC, and the right hand side denotes the NoC topology. The black squares denote the routers. The floorplan is depicted by annotating the boundaries by X and Y co-ordinate values respectively. The routers in the NoC topology are annotated by the co-ordinates of their lower left hand side boundary. For example, if a router is annotated with values (3,4), the location of its lower left hand side corner is the co-ordinate $X = 3$ and $Y = 4$.

Figure 7.33 depicts the pareto curve for the three benchmarks. The axis of the figure denotes the number of router resources in the solution, and the Y axis denotes the corresponding power consumption. The JPEG application generated a unique solution with three routers. The Network Processor application generated three pareto points corresponding to 8, 9 and 10 routers, and the set-top box application generated four pareto points corresponding to 8, 9, 10, and 12 routers, respectively.

As can be observed from the Pareto curve for the set-top box and network processor applications, the best trade-off between power consumption and the corresponding router resources


FLOORPLAN
15
13
11
10
8
7
5
0
5 6 7 8 10 11 12
19
7
8
22
12
9
20
15
10
11
13
16
17
24
4
3
18
2
1
6
14
0
23
5
21
NoC TOPOLOGY
(1,8)
(3,8)
(11,1)
(7,13)
(6,8)
(6,4)
(7,4)
(3,7)
(7,11)

Least power solution

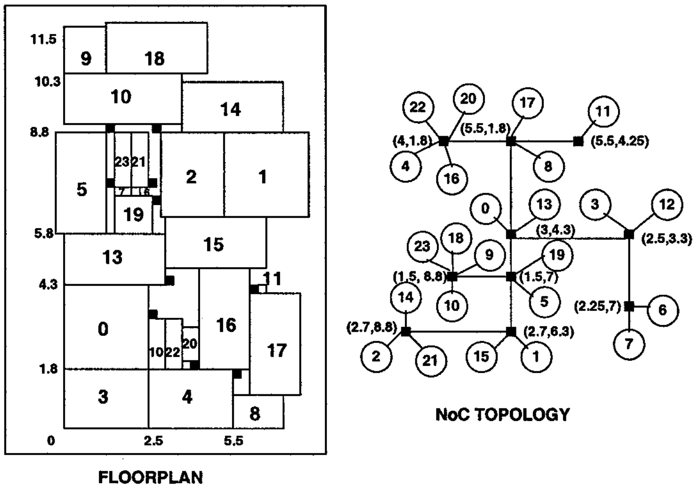

FLOORPLAN
15
13
11
10
8
7
5
0
5 6 7 8 10 11 12
NoC TOPOLOGY
(5,6)
(5,10)
(7,13)
(9,12)
(7,8)
(7,4)
(7,3)
(3,7)

Least router solution

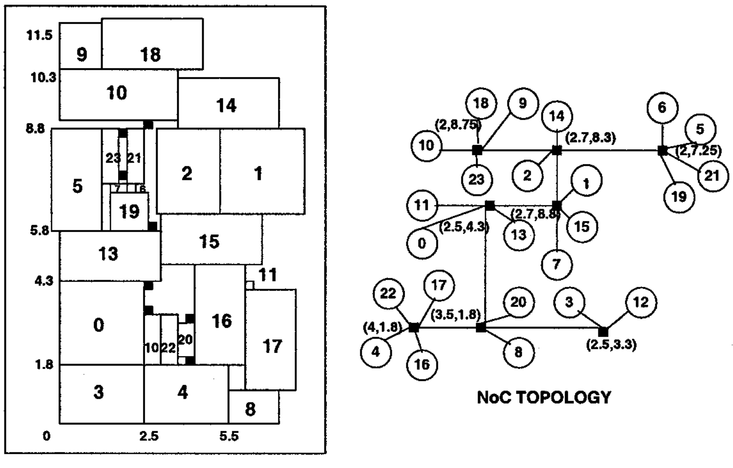
Fig. 7.30: Floorplan and NoC topology for set-top box

required was offered by the solution with 9 routers. This particular design and observation can only be obtained by generating a Pareto curve, thus substantiating our technique.

Figure 7.34 plots the variation power consumption in successive generations for the three pareto points of the set-top box benchmark. The plot is normalized to the highest power consumption among all generations. The least power solution improved six times for 9 router solutions, four times for 10 router solutions, and four times for 12 router solutions. The figure is annotated with regions when solutions corresponding to each router were dominant. The 8 and 9 router solutions remained dominant over all generations. The 10 router solution was initially dominant


11.5
10.3
8.8
5.8
4.3
1.8
0
2.5
5.5
9
18
10
14
5
23
21
2
1
19
13
15
11
0
16
17
10
22
20
3
4
8
FLOORPLAN
22
20
17
11
(4,1.8)
(5.5,1.8)
(5.5,4.25)
4
16
8
0
13
3
12
(3,4.3)
(2.5,3.3)
23
18
9
19
(1.5, 8.8)
(1.5,7)
14
10
5
(2.25,7)
6
(2.7,8.8)
(2.7,6.3)
7
2
21
15
1
NoC TOPOLOGY

Least power solution


11.5
10.3
8.8
5.8
4.3
1.8
0
2.5
5.5
9
18
10
14
5
23
21
2
1
19
13
15
11
0
16
17
10
22
20
3
4
8
FLOORPLAN
18
9
14
6
5
(2,8.75)
(2.7,8.3)
10
(2,7.25)
23
2
1
21
19
11
(2.7,8.8)
(2.5,4.3)
15
0
13
7
22
17
(3.5,1.8)
20
3
12
(4,1.8)
4
16
8
(2.5,3.3)
NoC TOPOLOGY

FLOORPLAN

Least router solution

Fig. 7.31: Floorplan and NoC topology for NP

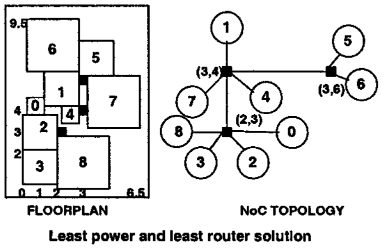

9.5
6
5
1
7
0
4
2
3
8
4
3
2
0 1 2 3 6.5
FLOORPLAN
1
5
(3,4)
(3,6)
6
7
4
(2,3)
8
0
3
2
NoC TOPOLOGY
Least power and least router solution

Fig. 7.32: Floorplan and NoC topology for JPEG encoder

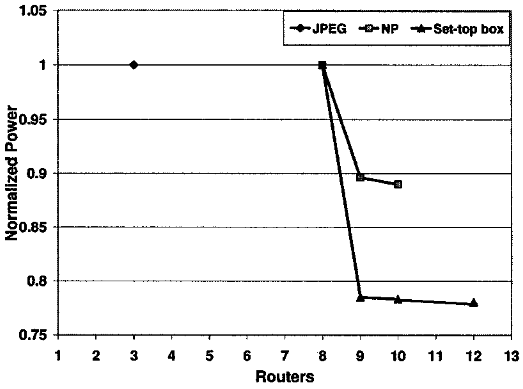


Fig. 7.33: Pareto curve for the JPEG, NP and set-top box applications

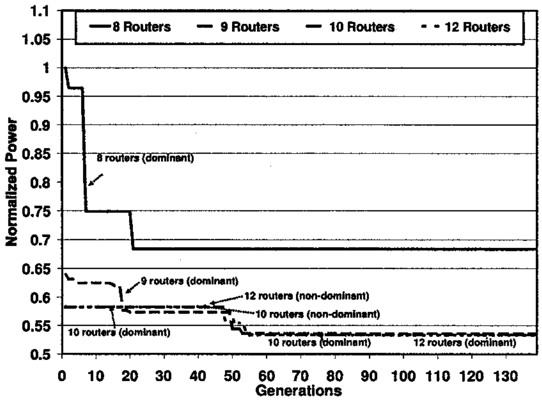


Fig. 7.34: Power consumption across generations

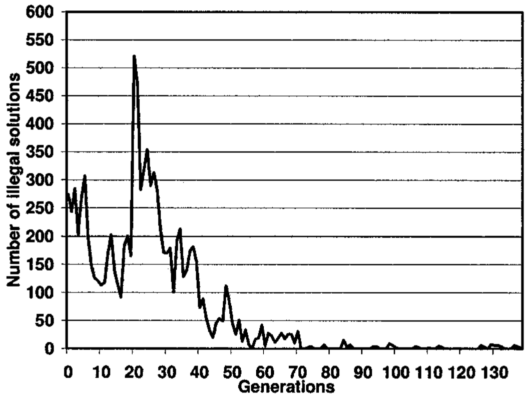


Fig. 7.35: Number of illegal solutions across generations

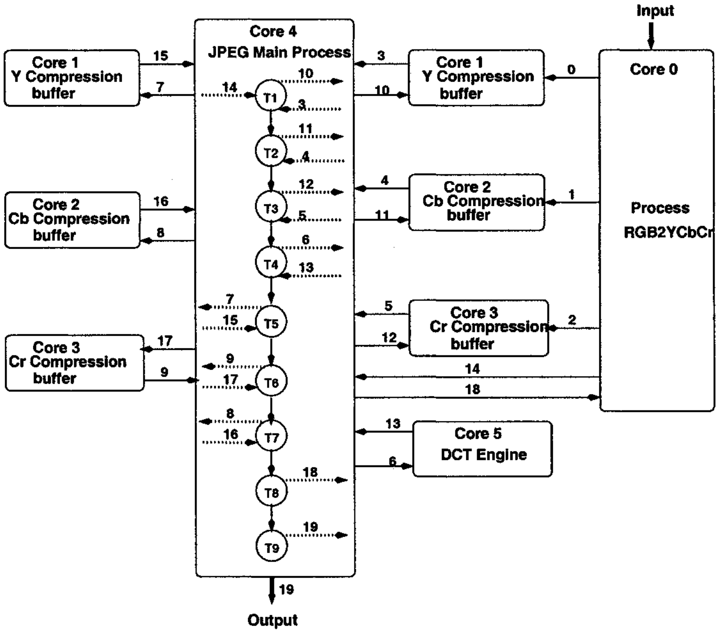

Input
Core 4
JPEG Main Process
Core 1
Y Compression
buffer
Core 1
Y Compression
buffer
Core 0
Core 2
Cb Compression
buffer
Core 2
Cb Compression
buffer
Process
RGB2YCbCr
Core 3
Cr Compression
buffer
Core 3
Cr Compression
buffer
Core 5
DCT Engine
T1
T2
T3
T4
T5
T6
T7
T8
T9
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
Output

Fig. 7.36: JPEG process graph

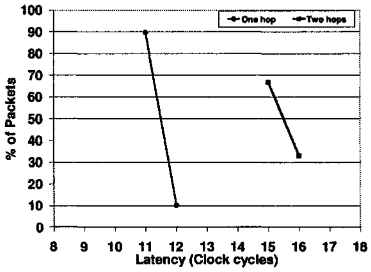


Fig. 7.37: Latency for JPEG encoder

with the least power consumption. Between generations 15 and 55, it was non-dominant as the 9 router solution consumed lesser power. Between 55 and the end, it again became dominant due to lower power consumption. The 12 router solution became dominant in the last few generations due to lower power consumption.

Figure 7.35 plots the number of illegal solutions at the end of each generation. At the end of the first generation, there were several illegal solutions in the population. However, due to their low fitness value, illegal solutions were filtered away in successive generations.

7.3.4.4. *Deadlocks.* Our router architecture included two virtual channels per input port, and two virtual channels per output port. For this architecture, we determined that deadlocks do not occur in any of our designs. The two virtual channels are able to successfully break any cycles in the channel dependency graph.

7.3.4.5. *Simulation with JPEG Encoder benchmark.* We validated our design approach by simulating a case study the ISO and ITU-T81(JPEG) standard compliant implementation available at the Opencores website [75]. We also compared the power consumption and router requirement for the custom topology generated by our technique against the mesh topology for the JPEG encoder. The mesh topology was generated from the floorplan of the JPEG encoder by manual optimization. The CTG and the custom network topology for the application are shown in Figures 7.28 and 7.32, respectively.

Similar to [12], we characterized the one hop latency across a router with 4 injecting ports and one receiving port that was close to congestion. Notice in this experiment there is contention among multiple packets from the injecting ports. The average one hop network latency was found to be 24 clock cycles for the entire packet. For our router architecture, the multiple hop latency was found to fit the curve given by $(hops \times 17) + 7$.

Cycle accurate simulation framework. The process graph for the JPEG encoder [75] is shown in Figure 7.36. The main process graph has two master cores namely Core 0 and 4. The other cores are slaves. Core 0 fills the Y (Core 1), Cb (Core 2), and Cr (Core 3) buffers and sends Core 4 a start signal. Core 4 obtains data from the Y, Cb and Cr buffers, performs DCT on the data (Core 5) , followed by Quantization (Core 6 and 8) and finally Huffman encoding (Core 7). Core 4 then outputs the compressed block and sends Core 0 the finish signal which then places the next set of data into the buffers. The arrows between the various cores denote the communication traces. The various stages within Core 4 denote the sequence of input/output transactions initiated by it.

A cycle accurate register transfer level VHDL model of the topology along with the JPEG cores was used to validate our approach. The JPEG cores were interfaced with the NoC through dedicated packetization and de-packetization modules.

Validation of power consumption model. We utilized a cycle accurate power consumption model (similar to [22]) of the router architecture and physical links to estimate the NoC power consumption. We observed a variation of about 10% between the power consumption values estimated from NoC synthesis, and the values obtained from the cycle accurate simulator.

Validation of latency model. We studied the network latency variations of the JPEG encoder application. Network latency is measured from the time the first flit of the packet enters the network, to the time the last flit leaves the network. Therefore, a constant packet length offset of 8 clock cycles corresponding to a packet size of 8 flits is added to the network latency.

Figure 7.37 presents the latency of packet forwarding in one-hop, and two hops, respectively. Since the network is operated in the un-congested mode, the latency of packet forwarding remained almost constant for packets routed in one hop as well as two hops.

Comparison with mesh based topology. The mesh based NoC design for the JPEG encoder consumed 2.58 times the power consumption of the custom topology. All traces in the mesh based topology were routed through two hops. Consequently, the average latency of forwarding the traffic traces was found to be 1.26 times the corresponding value.

7.4. Conclusion

In this chapter, we presented a novel GA based technique for application specific custom NoC design. We compared our technique with the optimal ILP formulation presented in Chapter 4, and the ANOC heuristic presented in Chapter 5. While ILP and ANOC suffer from high runtime and low solution quality respectively, experimental results on several multimedia and network processing benchmarks demonstrate that our GA based technique is able to generate close to optimal solutions in reasonable time. The scalability of the technique was demonstrated by experimentation with large industrial benchmarks.

CHAPTER 8

INTEGER LINEAR PROGRAMMING AND HEURISTICS FOR LOW POWER MAPPING AND SCHEDULING FOR MULTIPROCESSOR ARCHITECTURES, UNDER THROUGHPUT CONSTRAINTS

8.1. Introduction

In this chapter, we address the second part of the thesis, which pertains to system-level low power optimization. Embedded system applications in multimedia and network processing domains have witnessed an increase in complexity and performance requirements. These two factors coupled with the need for shorter design turn around times and ease of future upgrades have led to the advent of programmable multiprocessor architectures for design of such applications. Examples of commercial multiprocessor architectures aimed at these embedded applications include System-on-Chip (SoC) designs such as Intel IXP series (IXP1200, IXP2400, IXP2800) processors [80], TI TMS320C8x [81], Motorola C-port C-5 [82], and board level implementations such as Sun SX2500 board [83], Alacron FastImage 1500 [84], Synergy microsystem's MantaQX [85]. These architectures are deployed in portable devices (such as DVD players, digital cameras), set top boxes (HDTV), edge and backbone routers.

All these implementations have low power consumption as a key design requirement. The portable devices are constrained by battery lifetime. The set top boxes, edge and back bone routers have thermal budgets which in turn translate into power consumption constraints. Consequently, innovative system level low power design techniques are required for the implementation of these embedded applications on multiprocessor architectures.

System-level low power optimization is enabled by utilization of the dynamic power management (DPM) [86] and dynamic voltage scaling (DVS) [87] (also known as dynamic voltage frequency scaling) capabilities of a processor. Both, DPM and DVS were developed to address the challenge of increased power consumption in CMOS devices. DPM exploits the idle times in application behavior and turns off the power supply to several sub-systems of the processing element. Therefore, DPM reduces the stand-by power or leakage power consumption of the application. For example, the Intel SA1100 StrongARM processor [88] has two low power DPM modes namely idle and sleep in addition to the normal run mode. In the idle mode the CPU clock is switched off and in the sleep mode the power supply to majority of the chip is turned off.

DVS trades-off the active power consumption with performance while executing the non-timing critical portions of the application. In CMOS technologies the active power consumption of a chip can be specified as $P = C_{switch} \times V^2 \times f$ where C_{switch} is the average capacitance switched per clock cycle, V is the operating voltage and f is the frequency of operation. The propagation delay through a CMOS inverter can be approximated as $t_p = \frac{C_L}{2V}\left(\frac{1}{k_p} + \frac{1}{k_n}\right)$ [89] where C_L is the load capacitance, V is the operating voltage, and k_p and k_n are the process gain factors of the p- and n-type devices, respectively. The frequency of the device can then be calculated as $f = \frac{1}{t_p} = \frac{2V}{C_L}\left(\frac{k_p k_n}{k_p + k_n}\right) = KV$, where $K = \frac{2}{C_L}\left(\frac{k_p k_n}{k_p + k_n}\right)$. Thus, the frequency of a processor in CMOS technology is linearly dependent on the operating voltage. Therefore reduction in the supply voltage results in a cubic reduction in power consumption at the expense of a linear slow down in the processor speed. DVS exploits this relationship to provide variable operating voltages and corresponding frequencies for the processor. For example, the Intel SA1100 StrongARM processor [88] supports supply voltages that range from 0.8 V to 1.5 V with corresponding operating frequencies ranging from 59 MHz to 206 MHz.

DPM and DVS can be currently applied to multiprocessor board level architectures. In SoC based architectures, the emerging globally synchronous locally asynchronous design methodology [3] with multiple voltage and clock islands would also include DPM and DVS.

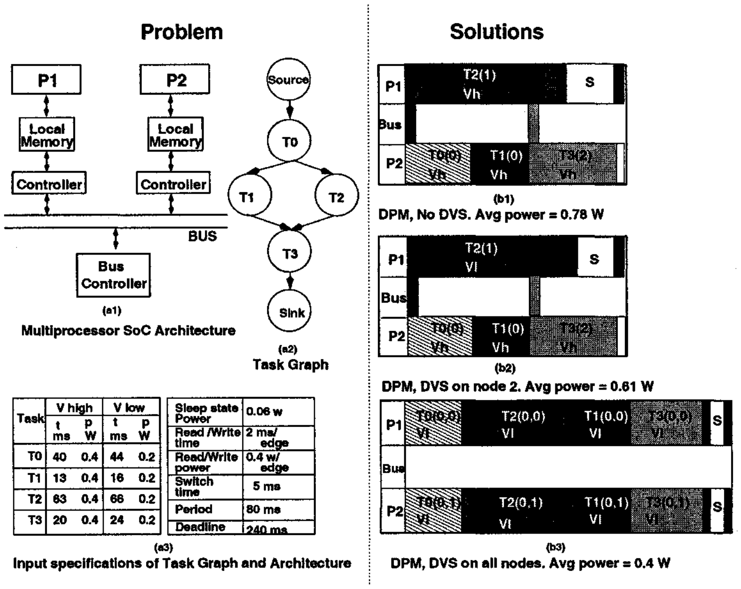
Embedded system applications in the multimedia and network processing domains demonstrate periodic behavior. Real life implementations of these applications allow a greater time to process each data block than the period (period = 1/throughput). Time to process each data block is typically denoted by latency or deadline. For example, the MPEG-1 video decoder is required to support a stream of upto 1.5 Mbps. If we consider that the decoder in each execution generates a macroblock of 8x8 (64 values) of 8 bits each, then the throughput requirement can be specified as 2,929 macroblocks/s (approximately) or a period of 345 μs/block. If the stream is intended for human viewing, an initial latency of 100 μs will not result in detectable degradation. In other words, an MPEG-1 decoder that generates a steady stream of 2,929 macroblocks/s(period = 345 μs/block) with a latency of 445 (100 + 345) μs/block will satisfy the performance requirements. Performance constraints with deadline greater than period coupled with multiprocessor target architecture enable the application of system level loop transformations such as pipelined scheduling and unrolling.

Pipelined scheduling and loop unrolling are two powerful loop transformations for throughput maximization of applications. Pipelined scheduling constructs a steady state that overlaps instances of tasks belonging to different iterations of the original specification. In the steady state there is only one instance of every task in the application. Unrolling as the name suggests transforms the original specification by replicating successive iterations. Thus, the transformed specification includes more than one instance of each task in the original loop. The two transformations can also be applied in an integrated manner where the specification is first unrolled and then scheduled into a pipeline.

This chapter presents system level design techniques that integrate loop transformations with DPM and DVS to minimize the power consumption of applications mapped to multiprocessor architectures. In the following two sections we present a motivating example.

8.1.1. Embedded multiprocessor architecture

The techniques that are presented in this work are applicable to a large class of embedded multiprocessor System-on-Chip (SoC) architectures. In a typical SoC architecture each processing


Problem
Solutions
P1
P2
Local Memory
Local Memory
Controller
Controller
BUS
Bus Controller
Source
T0
T1
T2
T3
Sink
P1
Bus
P2
T2(1) Vh
S
T0(0) Vh
T1(0) Vh
T3(2) Vh
T2(1) Vl
T0(0,0) Vl
T2(0,0) Vl
T1(0,0) Vl
T3(0,0) Vl
T0(0,1) Vl
T2(0,1) Vl
T1(0,1) Vl
T3(0,1) Vl

(a1) Multiprocessor SoC Architecture

(a2) Task Graph

Task	V high		V low	
	t ms	p W	t ms	p W
T0	40	0.4	44	0.2
T1	13	0.4	16	0.2
T2	63	0.4	66	0.2
T3	20	0.4	24	0.2

Sleep state Power	0.06 w
Read /Write time	2 ms/ edge
Read/Write power	0.4 w/ edge
Switch time	5 ms
Period	80 ms
Deadline	240 ms

(a3) Input specifications of Task Graph and Architecture

(b1) DPM, No DVS. Avg power = 0.78 W

(b2) DPM, DVS on node 2. Avg power = 0.61 W

(b3) DPM, DVS on all nodes. Avg power = 0.4 W

Fig. 8.1: Motivating Example

engine has its own code and data memory. The SoC architectures (such as Intel IXP series processors) provide sufficient on-chip memory to map the code and data segments of typical tasks. Embedded system application domains such as multimedia and network processing demonstrate clear demarcation of producer and consumer tasks with well defined inter-task communication behavior. Due to these characteristics and the application specific nature of the design, the designer can statically allocate data and code memory for each task, and data memory for inter-processor communication. Support for mutual exclusion during memory accesses is provided by the memory controller in hardware.

Existing SoC architectures implement a multitude of memory configurations to support inter-processor communication. The memory configurations for inter-processor communication can be broadly classified into two categories, namely centralized shared memory (such as scratch pads) and distributed memory (special transfer memory that is local) similar to uniform (UMA) and non-uniform memory access (NUMA) schemes, respectively of multi-computer systems [90] (which are not SoC). However, in contrast to multi-computer architectures, the SoC devices do not support cache coherence schemes. The techniques presented in this work be applied to both schemes of inter-processor communication.

For the purposes of discussing the motivating example and without loss of generality, we consider the target multiprocessor architecture shown in Figure 8.1(a1). The architecture supports special transfer memory which is local to each processor for inter-processor communication. The programmable bus controller implements a pre-defined bus protocol. Based on the static schedule and the underlying architecture, the controller assigns bus access rights to individual processing elements on every initiation of task execution. The time overhead due to a task reading (writing) from (to) its local memory is included in the run time specification of task. A task always writes to its local memory. Inter-processor communication takes place whenever a task is required to read data that is not available in its local memory.

8.1.2. Motivating Example

Figure 8.1(a2) shows a task graph (TG) consisting of 4 tasks that are to be mapped onto a homogeneous (identical) two processor architecture. *Note that although the example is shown for homogeneous architecture, our techniques are applicable to heterogeneous architectures as well.* The specifications of the architecture and the task graph are given in the tables in Figure 8.1(a3).

The figure shows three pipelined schedules in Figures 8.1(b1), 8.1(b2) and 8.1(b3) respectively. Figure 8.1(b1) depicts a pipelined schedule that does not apply DVS, Figure 8.1(b2) shows a pipelined schedule that applies DVS, and finally Figure 8.1(b3) depicts a schedule that applies combined DVS, unrolling and pipelined scheduling. All three schedules apply DPM whenever the processor idle time is more than the time overhead (order of ms, 5 ms is a typical value) incurred by switching to and from the low power state. In Figures 8.1(b1) and 8.1(b2), each box contains the task name, its pipeline stage in brackets, and the voltage at which the processor is operating during the execution of task. In Figure 8.1(b3), each box contains the task name, its pipeline stage and its instance in brackets, and the voltage at which the processor is operating during the execution of the task. The black boxes represent switching overhead. The white boxes with "s" indicate that the processor is in a sleep state. Empty white boxes indicate that the idle time is less than the switching overhead and hence, the processor does not go into a sleep state. The shaded portion on the bus indicates that it is busy and the white portion indicates it is idle.

The average power consumption of a design can be calculated by dividing the total energy required to execute the steady state by the period. The energy required to execute the steady state is the summation of the energy required to perform each task, the communication overheads and processor voltage switching overheads. Therefore, the average power consumption is given by:

$$P_{avg} = \frac{1}{\mathcal{T}_{period}} \left[\sum_{\forall v_i} \mathcal{T}_{i,j,k} \cdot \mathcal{P}_{i,j,k} + \sum_{\forall s_i^{dpm}} \mathcal{T}_i^{dpm} \cdot \mathcal{P}_i^{dpm} + \right.$$

$$\left. \sum_{\forall c_k \in C} \mathcal{T}_{c_k} \cdot \mathcal{P}_{c_k} + \sum_{\forall v_i} \mathcal{TC}_i \cdot (\mathcal{P}_{mem} + \mathcal{P}_{bus}) \right]$$

where $\mathcal{T}_{i,j,k}$ is the total execution time of a task v_i scheduled on a processor pe_j at an operating voltage state s_k^{norm}, $\mathcal{P}_{i,j,k}$ is the average power consumption of that task, $\mathcal{T}_i^{dpm}$ is the time spent in sleep state s_i^{dpm}, $\mathcal{P}_i^{dpm}$ is the power consumed in that sleep state, $\mathcal{P}_{c_k}$ is the power overhead due to switching between states, $\mathcal{T}_{c_k}$ is the time overhead due to switching between states, $\mathcal{P}_{mem}$ is the average power consumption of the memory, $\mathcal{P}_{bus}$ is the average power consumption of the bus, $\mathcal{TC}_i$ is the communication overhead of task v_i, and $\mathcal{T}_{period}$ is the period constraint. If the task graph is unrolled to K degrees, the average power is computed on the entire task graph that has been replicated K times and has a period constraint $\mathcal{T}_{period} * (K + 1)$.

The pipelined design with DPM and without DVS (Figure 8.1(b1)) consumes an average power of 0.78 W, the pipelined design with DPM and DVS (Figure 8.1(b2)) has an average power consumption of 0.61 W, and finally the unrolled and pipelined design with DPM and DVS (Figure 8.1(b3)) has an average power consumption of 0.4 W. It is evident that the design that consumes least power not only has all tasks running at a low power DVS state, it also has minimum communication overhead. Unrolling of the task graph gives us more design space to explore and hence, we can obtain a design that consumes lower power.

8.1.3. Our Contributions

The work makes the following contributions:

1. The chief contribution of the work is in the realm of design techniques for system-level power optimization of throughput constrained applications on multi-processor architectures.
2. We present an optimal MILP formulation that integrates DVS with pipelined scheduling and unrolling, and applies DPM as a final stage to generate system level low power designs on embedded multiprocessor architectures.
    - We present three modifications to the MILP formulation that trade-off design quality for reduced run times.
    - We present linearization techniques that generate linear programming formulations for non-linear constraints encountered in the integration of system low power techniques with loop transformations.
3. We also present a low run time heuristic technique that integrates DVS, DPM, pipelined scheduling and unrolling for designing low power embedded system implementations. Although, the MILP formulation generates an optimal solution, it has exponential run times. The heuristic technique requires much lower run times and generates close to optimal results.

- We present deterministic and simulated annealing based low power optimization strategies that can be integrated with the heuristic technique.

We also present results of extensive experimentation with realistic multimedia applications and synthetic task graphs to evaluate the quality and run-times of our techniques.

The chapter is organized as follows: Section 8.2 discusses the previous work, Section 8.3 presents the MILP formulations, Section 8.4 presents the heuristic techniques, Section 8.5 discusses the experimental results, and finally Section 8.6 concludes the chapter.

8.2. Previous Work

8.2.1. Power minimization in uni-processor systems

Kwon et al. [91] presented two voltage allocation techniques for variable voltage uni-processor architectures. They presented an optimal voltage allocation technique for the case when the power consumption of all the tasks was equal, and an MILP formulation for the case when the power consumption was different. Gang et al. [92] presented a fixed priority energy efficient scheduling technique on variable voltage processors. They determine a fixed minimum constant speed for all tasks, and then find a better voltage schedule that further minimizes power consumption. Hong et al. [93] addressed the problem of jointly scheduling periodic and aperiodic tasks to achieve power minimization. Shin et al. [94] presented an integrated scheduling, DVS and DPM technique for hard real time tasks that consistently run at lesser time than their worst case execution times. Flautner et al. [95] described low power scheduling techniques based on performance factor predictors that are computed dynamically. Rao et al. [96] [97] obtained analytical results for energy optimal speed control of a single device that supports DVS. In [98], they presented analysis of a two device chain that communicate through intermediate buffers. In contrast to our work, they do not address the mapping and scheduling problem. Jianli et al. [99] present task scheduling algorithms for a single processor capable of operating at multiple voltages, and a set of devices.

All the above mentioned techniques are aimed at power minimization in uni-processor systems. Our work is distinguished from these techniques due to its focus on multiprocessor systems. Embedded system implementations on multiprocessor architectures give an opportunity to explore loop transformations together with DVS and DPM for power minimization.

8.2.2. Power minimization in multiprocessor systems

Zhang et al. [100] proposed a two phase technique for energy minimization of task graphs with data dependence. The first phase is a heuristic task scheduling technique, and the second phase is an integer programming (IP) formulation for voltage selection. The formulation is constrained by a pre-defined schedule from the first phase. As a result, it explores only a subset of the entire range of possible solutions. Therefore, the technique (task scheduling and voltage selection) does not guarantee an optimal solution. In contrast, we present a MILP formulation that searches the entire solution space by integrating scheduling and voltage selection.

Luo et al. [101] presented a low power heuristic design technique to schedule independent aperiodic and periodic task graphs jointly in real time embedded systems. A list based scheduling technique with energy sensitive priority function on task graphs with data dependence constraints was presented by Gruian et al. [102]. Liu et al. [103] presented a power aware scheduling algorithm on data dependent task graphs for mission critical applications. A hybrid global/local search optimization framework on task graph with data dependence constraints was given by Bambha et al. [104]. Yang et al. [105] presented a two phase (offline and online) energy aware scheduling technique for data dependent task graphs operating on multiprocessor SoC architectures. Manzak et al. [106] presented a two pass technique for scheduling data dependent task graphs that distribute the slack to achieve power minimization.

All the above mentioned techniques do not consider trading off performance gains by pipelining and loop unrolling for power optimization. Many realistic implementations of multimedia and network processing applications support pipelined processing of incoming data. Multiprocessor mapping of such applications offers an opportunity to apply algorithmic transformations to minimize power. Our work is distinguished from the existing approaches due to its application of integrated loop transformations (pipelining and unrolling), DVS and DPM to optimize the throughput and power consumption of an embedded multiprocessor architecture.

8.2.3. Low power scheduling in high-level synthesis

The problem of resource constrained low power scheduling on multi-component systems has also been addressed in the context of high level synthesis of application specific integrated circuits (ASIC). In particular, Lin et al. [107], Johnson et al. [108] and Mohanty et al. [109] [110] have addressed the problem of power minimization under latency constraints by sequential scheduling on variable voltage resources. In other words, similar to the multiprocessor techniques mentioned above, they have not addressed the problem in the context of power optimization by pipelining and unrolling. Further, as their techniques are aimed at ASIC synthesis, they do not account for processor voltage switching and inter-processor communication overheads (both in terms of time and energy) that are crucial for system-level design.

8.2.4. Algorithmic transformations for system-level low power design

Existing approaches that apply loop transformations for power minimization concentrate on memory power reduction in uni-processor systems. Shiue et al. [111] use loop transformations for memory design and exploration of low power embedded systems. Esakkimuthu et al. [112] present ways to reduce memory energy by applying loop transformations. This thesis, on the other hand, integrates DVS and DPM techniques with loop transformations to minimize processor power in a multiprocessor architecture.

8.3. MILP Formulations for System Level Low Power Design

In this section, we first present the optimal MILP constraint formulation for unrolled and pipelined scheduling of the given directed acyclic graph, on the target multiprocessor architecture. We also present three modified formulations that restrict the solution space by altering one or more constraints in the optimal formulation and hence require lower run time to generate their solutions. In Section 8.3.1 we define the problem, Section 8.3.2 presents the optimal constraint formulation ($MILP$), and in Sections 8.3.3, 8.3.4 and 8.3.5, we present each of the three modified MILP formulations, respectively.

8.3.1. Problem Definition

The problem that we address consists of mapping and obtaining a pipelined schedule of a periodic application on a DVS and DPM enabled embedded multiprocessor architecture such that the throughput and latency are satisfied, and power is minimized. Formally, the problem can be stated as follows.

Given:

- An application specified as a directed acyclic task graph (DAG), $G(V, E)$ where $v_i \in V$ denotes a task and $e_k = (v_i, v_j) \in E$ denotes data dependency between v_i and v_j. For every edge $e_k = (v_i, v_j) \in E$, $\omega(e_k)$ denotes the total bytes of data transferred from v_i to v_j.
- An embedded multiprocessor architecture with a set of processing elements PE. For every processing element $pe_i \in PE$, the DVS and DPM capabilities are given by a graph $I(pe_i) = \{S, C\}$ where $s_j \in S$ denotes a particular voltage state of the processor and $c_k = (s_i, s_j)$

is an edge between two states. For every edge $c_k = (s_i, s_j) \in C$, $\mathcal{T}_{c_k}$ and $\mathcal{P}c_k$ denote the time and power overheads to switch between the two states s_i and s_j. The subset of S that includes the execution states is given by S_{norm}. The various shut down states are given by $S_{shut} = S/S_{norm}$. For each shut down state $s_i^{dvm} \in S_{shut}$, the average power consumption is given by $\mathcal{P}_i^{dvm}$.

- A characterization for every task $v_i \in V$, executing on processor $pe_j \in PE$ at voltage state $s_k \in S_{norm}$, in terms of its execution time (including communication with local memory) and power consumption denoted by $\mathcal{T}_{i,j,k}$ and $\mathcal{P}_{i,j,k}$, respectively.
- The average time and power per (read or write) access for the bus denoted by $\mathcal{T}_{bus}$ and $\mathcal{P}_{bus}$, respectively.
- The average time and power per (read or write) access for the shared memory denoted by $\mathcal{T}_{mem}$ and $\mathcal{P}_{mem}$, respectively.
- A period $\mathcal{T}_{period}$ and deadline constraint $\mathcal{T}_{deadline}$ on the execution of the task graph ($\mathcal{T}_{deadline} > \mathcal{T}_{period}$).

The objective is to map and obtain a static, unrolled, and pipelined design of the task graph on the processing elements such that the average power required to execute the steady state is minimized while satisfying performance constraints. Many applications at the system level do not contain strongly connected components. For example, in this work, we modeled the JPEG decoder, MPEG-1 decoder, and MP3 encoder algorithms as directed acyclic graphs. Hence, our techniques accept DAGs as input. Pipelined scheduling under resource, period and deadline constraints is an NP complete problem [71].

8.3.2. The MILP formulation for optimal solution

In this section, we present the MILP formulation that gives an optimum solution for the problem defined in Section 8.3.1. In the following description we first discuss the formulation for the distributed memory architecture utilized in the motivating example in Section 8.1.2, and then present modifications for the shared memory architecture. Section 8.3.2.1 derives constants from the problem definition, Section 8.3.2.2 defines base and derived variables, Section 8.3.2.3 states the objective function, Section 8.3.2.4 states the constraints that the objective function must satisfy, Section 8.3.2.5 discusses the modifications for shared memory architecture, and finally Section 8.3.2.6 presents techniques to linearize the non-linear equations. Table 20 gives the nomenclature of various variables utilized in the formulation and acts as a quick reference for the reader.

8.3.2.1. *Constants.* The following constants are derived from the problem definition:

- *Unrolling degree*: Let K be the unrolling degree of the task graph. Therefore, task graph $G(V,E)$ will be replicated K times, such that, $G(V,E)$ is transformed to $G^K(V^K,E^K)$, where V^K represents V replicated K times, and E^K represents E replicated K times. Clearly, $|V^K| = (K+1)|V|$ and $|E^K| = (K+1)|E|$. The schedule table will therefore have $K+1$ instances of the task graph. Multiple task graphs are scheduled by unrolling them over the hyper-period of the graphs. Let an instance of the task graph be denoted by a number k such that $0 \leq k \leq K$.

- *Deadline and Period*: We calculate the deadline $\mathcal{T}^k_{deadline}$ of each instance of the task graph as follows: $\forall k, 0 \leq k \leq K$,

$$\mathcal{T}^k_{deadline} = \mathcal{T}_{deadline} + k * \mathcal{T}_{period}$$

  We calculate the period of the unrolled graph as follows:

$$\mathcal{T}^K_{period} = (K+1) * \mathcal{T}_{period}$$

Variable	Description
K	Unrolling degree
$MAXVAL$	A very large number
$\mathcal{T}_{deadline}$	Deadline of task graph
$\mathcal{T}^k_{deadline}$	Deadline of k^{th} instance of task graph
$\mathcal{T}_{period}$	Period of the task graph
$\mathcal{T}^K_{period}$	Period of the unrolled task graph
ps^k_i	Pipeline stage of task v_i of k^{th} instance of task graph
$\mathcal{M}_{i,j}$	(0,1) variable, 1 iff task v_i is mapped to processor pe_j
$\mathcal{S}_{i,j}$	(0,1) variable, 1 iff task v_i executes at operating state s_j
T^r_i	Time when task v_i is ready to start execution
T^{sr}_i	Time when task v_i starts reading incoming edges
T^{se}_i	Time when task v_i starts execution
T^w_i	Time when task v_i is ready to write to shared memory
T^{ee}_i	Time when task v_i has finished execution
T^{es}_i	Time after possible switching for task v_i
$D_{T_{t_i,t_j}}$	(0,1) variable, 1 iff $t_i \leq t_j$
$\mathcal{MV}_{i,j}$	(0,1) variable, 1 if tasks v_i and v_j are mapped on same processor
$\mathcal{C}_i$	(0,1) variable, 1 if task v_i utilizes the bus
$\mathcal{CR}_i$	(0,1) variable, 1 iff task v_i reads from the shared memory
$\mathcal{CW}_i$	(0,1) variable, 1 iff task v_i writes from the shared memory
$\mathcal{SV}_{i,j}$	(0,1) variable, 1 if tasks v_i and v_j are at same voltage
$\mathcal{TC}_{i,j}$	Time overhead for communication between v_i and v_j
$\mathcal{TC}_i$	Total communication time for v_i
$\mathcal{TCR}_{i,j}$	Time overhead for reading data due to edge (v_i, v_j) from the shared memory
$\mathcal{TCW}_{i,j}$	Time overhead for writing data due to edge (v_i, v_j) from the shared memory
$\mathcal{TCR}_i$	Total read overhead for task v_i
$\mathcal{TCW}_i$	Total write overhead for task v_i
$\mathcal{MS}_{i,j,k}$	(0,1) variable,1 iff task v_i mapped to processor pe_j at operating state s_k
$\mathcal{TR}_i$	Runtime of task v_i
$\mathcal{PR}_i$	Power consumption of task v_i
$\mathcal{N}_{i,j}$	(0,1) variable, 1 iff task v_j follows v_i on same processor
$\mathcal{NI}_{i,j}$	(0,1) variable, 1 iff task v_j follows v_i in the same iteration
$\mathcal{FT}_i$	(0,1) variable, 1 iff task v_i is the first task on the processor
$\mathcal{LT}_i$	(0,1) variable, 1 iff task v_i is the last task on the processor
$\mathcal{NW}_{i,j}$	(0,1) variable, 1 iff task v_j follows v_i across iterations
$\mathcal{SO}_i$	Switching overhead of task v_i
$\mathcal{EO}_{i,j}$	(0,1) variable, 1 iff task v_i starts before v_j and both are mapped to same processor
$\mathcal{BC}_{i,j}$	(0,1) variable, 1 iff task v_j accesses the bus, and task v_i starts before v_j
$\mathcal{BC}^{r,r}_{i,j}$	(0,1) variable, 1 iff task v_j accesses the bus for reading, before v_j accesses for reading
$\mathcal{BC}^{w,r}_{i,j}$	(0,1) variable, 1 iff task v_j accesses the bus for writing, before v_j accesses for reading
$\mathcal{BC}^{w,w}_{i,j}$	(0,1) variable, 1 iff task v_j accesses the bus for writing, before v_j accesses for writing
$\mathcal{BC}^{r,w}_{i,j}$	(0,1) variable, 1 iff task v_j accesses the bus for reading, before v_j accesses for writing

Table 20: Nomenclature of variables

- We define $MAXVAL$ to be some big number such that any number used in the formulation is less than $MAXVAL$.

Henceforth, all instances of the same task $v_i \in V^K$ will be treated as separate tasks. That is, if v_i and v_j are two different instances of the same task, $v_i \neq v_j$.

8.3.2.2. *Variables.* In this section, we define the independent and the dependent variables that are used in the formulation.

Base variables: We define the following base (independent) variables.

- *Pipeline stage variable*: Let the pipeline stage of a task $v_i \in V^K$, belonging to the task graph whose instance is k be denoted by ps_i^k.

- *Task-Processor mapping matrix*: We define a (0,1) variable $\mathcal{M}_{i,j}$ such that $\forall v_i \in V^K$ and $\forall pe_j \in PE$,

$$\mathcal{M}_{i,j} = \begin{cases} 1, \; if \; v_i \, is \, mapped \, onto \, pe_j \\ 0, \; otherwise \end{cases}$$

- *Task-Voltage State mapping matrix*: We define a (0,1) variable $\mathcal{S}_{i,j}$ such that $\forall v_i \in V^K$ and $\forall s_j \in \mathcal{S}_{norm}$,

$$\mathcal{S}_{i,j} = \begin{cases} 1, \; if \; v_i \; executes \; at \; s_j \\ 0, \; otherwise \end{cases}$$

- *Task runtime instance variables*: We define $\mathcal{T}_i^r$ to denote the time instance when task v_i is ready to start execution, $\mathcal{T}_i^{sr}$ to denote the time instance when v_i starts reading incoming edges, $\mathcal{T}_i^{se}$ to denote the time instance when v_i starts execution, $\mathcal{T}_i^{ee}$ to denote the time instance when v_i has finished execution and written data to its local memory, and $\mathcal{T}_i^{es}$ to denote the time instance when a possible switching of states after the execution of v_i has occurred.

- *Time instance comparison variable*: Given two time instances $\mathcal{T}_i$ and $\mathcal{T}_j$, we define a (0,1) variable $\mathcal{D}_{\mathcal{T}_i,\mathcal{T}_j}$ such that

$$\mathcal{D}_{\mathcal{T}_i,\mathcal{T}_j} = \begin{cases} 1, \; if \; T_i \leq T_j \\ 0, \; otherwise \end{cases}$$

We define an auxiliary (0,1) variable $\mathcal{D}^c_{\mathcal{T}_i,\mathcal{T}_j}$ such that,

$$\mathcal{T}_j - \mathcal{T}_i - \mathcal{D}_{\mathcal{T}_i,\mathcal{T}_j} * MAXVAL \leq -1$$

$$\mathcal{T}_i - \mathcal{T}_j - \mathcal{D}^c_{\mathcal{T}_i,\mathcal{T}_j} * MAXVAL \leq 0$$

$$\mathcal{D}_{\mathcal{T}_i,\mathcal{T}_j} + \mathcal{D}^c_{\mathcal{T}_i,\mathcal{T}_j} = 1$$

Derived Variables: We define the following derived variables.

- *Multiple tasks on the same processor*: For each pair of tasks v_i and $v_j \in V^K$, we define a (0,1) variable $\mathcal{MV}_{i,j}$ that is 1 if v_i and v_j are scheduled on the same processor, else 0. This can be represented as follows:

$$\mathcal{MV}_{i,j} = \Sigma_{\forall pe_k \in PE} \mathcal{M}_{i,k} * \mathcal{M}_{j,k}$$

- *Bus access variable*: A task $v_i \in V^K$ utilizes the bus if at least one of its parents is scheduled on a different processor. We define a (0,1) variable $\mathcal{C}_i$ such that $\forall v_i \in V^K$,

$$\mathcal{C}_i = \begin{cases} 1, \; if \; v_i \; utilizes \; the \; bus \\ 0, \; otherwise \end{cases} \tag{8.1}$$

The variable C_i can be derived as follows: $\forall v_i \in V^K$,

$$\forall (v_j, v_i) \in E^K, \mathcal{C}_i \geq (1 - \mathcal{MV}_{i,j}) \tag{8.2}$$

$$\mathcal{C}_i \leq \Sigma_{\forall (v_j,v_i) \in E^K} (1 - \mathcal{MV}_{i,j}) \tag{8.3}$$

The first equation sets C_i to 1 if v_i and any parent v_j are scheduled on different processors, and the second equation sets C_i to 0 if v_i and all its parents are scheduled on the same processor.

- *Multiple tasks at the same voltage state*: For each pair of tasks v_i and $v_j \in V^k$, we define a (0,1) variable $\mathcal{SV}_{i,j}$ that is 1 if v_i and v_j are scheduled at the same voltage state, else 0. This can be represented as follows:

$$\mathcal{SV}_{i,j} = \Sigma_{\forall s_k \in S_{norm}} \mathcal{S}_{i,k} * \mathcal{S}_{j,k}$$

- *Edge communication overhead*: Let the inter-processor communication overhead of each edge $e_k = (v_j, v_i) \in E^K$ be denoted by $\mathcal{TC}_{i,j}$. $\mathcal{TC}_{i,j}$ will be the time overhead associated with transfer of data from the local memory of the producer processor to the local memory of the consumer processor, such that

$$\mathcal{TC}_{i,j} = \begin{cases} 0,\ if \mathcal{MV}_{i,j} = 1 \\ (\mathcal{T}_{bus} + \mathcal{T}_{mem}) \cdot \omega(e_k),\ otherwise \end{cases} \tag{8.4}$$

- *Total communication overhead per task*: Let the total communication overhead of task v_i be denoted by $\mathcal{TC}_i$. Therefore, $\mathcal{TC}_i$ will be the sum of the communication overhead due to all incoming edges of v_i. Mathematically, $\forall i \in V^K$

$$\mathcal{TC}_i = \Sigma_{\forall v_j, (v_j, v_i) \in E^K} (\mathcal{TC}_{i,j}) * (1 - \mathcal{MV}_{i,j}) \tag{8.5}$$

- *Task mapped to a processor operating at a voltage state*: We define a (0,1) variable $\mathcal{MS}_{i,j,k}$ such that $\forall v_i \in V^K$, $\forall pe_j \in PE$, and $\forall s_k \in S_{norm}$,

$$\mathcal{MS}_{i,j,k} = \mathcal{M}_{i,j} * \mathcal{S}_{i,k}$$

$\mathcal{M}_{i,j} * \mathcal{S}_{i,k}$ gets a value 1 if task v_i executes on processing element pe_j operating at a state s_k, else 0.

- *Task execution time variable*: We define a variable $\mathcal{TR}_i$ to denote the runtime of the task. $\mathcal{TR}_i$ is computed as follows:

$$\mathcal{TR}_i = \Sigma_{\forall pe_j \in PE} \Sigma_{\forall s_k \in S_{norm}} \mathcal{MS}_{i,j,k} * \mathcal{T}_{i,j,k}$$

- *Task power consumption variable*: We define a variable $\mathcal{PR}_i$ to denote the power consumed by the task. $\mathcal{PR}_i$ is computed as follows:

$$\mathcal{PR}_i = \Sigma_{\forall pe_j \in PE} \Sigma_{\forall s_k \in S_{norm}} \mathcal{MS}_{i,j,k} * \mathcal{P}_{i,j,k}$$

- *Switching overhead and related variables*: For two tasks $v_i, v_j \in V^K$, we define a (0,1) variable $\mathcal{N}_{i,j} = 1$ if v_j follows v_i immediately on the same processor, else 0. v_j can immediately follow v_i in two ways:

  - In the same iteration of the schedule
  - Across two iterations of the schedule

  Obviously, v_j can follow v_i immediately in only one of the two ways.

  For v_j to follow v_i immediately in the same iteration, the following two conditions have to hold true:

  - v_i and v_j should be mapped to the same processor
  - No task that is mapped to the same processor as v_j (and v_i) should be scheduled between v_i and v_j.

  The second will be violated if any task $v_k \in V^K$ starts before v_j ($\mathcal{D}_{\mathcal{T}^r_k, \mathcal{T}^r_j} = 1$), starts after v_i ($\mathcal{D}_{\mathcal{T}^r_i, \mathcal{T}^r_k} = 1$), and is mapped onto the same processor as v_i (and v_j) ($\mathcal{MV}_{i,k} = 1$). Therefore, v_j immediately follows v_i in the same iteration if, $\mathcal{D}_{\mathcal{T}^r_i, \mathcal{T}^r_j} = 1$, $\mathcal{MV}_{i,j} = 1$, and a (0,1) variable $\mathcal{NS}_{i,j} = 1$ where,

$$\mathcal{NS}_{i,j} = \begin{cases} 1,\ if \\ \Sigma_{\forall v_k, v_k \neq v_i \neq v_j}(\mathcal{D}_{\mathcal{T}_k^r,\mathcal{T}_j^r} * \mathcal{D}_{\mathcal{T}_i^r,\mathcal{T}_k^r} * \mathcal{MV}_{i,k}) = 0 \\ 0,\ otherwise \end{cases}$$

We define a (0,1) variable $\mathcal{NI}_{i,j} = 1$ if v_j follows v_i immediately in the same iteration, else 0. Therefore,

$$\mathcal{NI}_{i,j} = \mathcal{D}_{\mathcal{T}_i^r,\mathcal{T}_j^r} * \mathcal{MV}_{i,j} * \mathcal{NS}_{i,j}$$

For v_j to immediately follow v_i across iterations, the following three conditions must hold true: i) v_j must be the first task scheduled on its processor, ii) v_i must be the last task mapped on its processor, and iii) v_i and v_j should be mapped onto the same processor. The conditions can be mathematically formulated as follows: let $\mathcal{FT}_j$ be a (0,1) variable that is 1 if v_j is the first task mapped onto its processor, else 0. Therefore,

$$\mathcal{FT}_j = \begin{cases} 1,\ if\ \Sigma_{\forall v_k \in V^K, v_k \neq v_j} \mathcal{D}_{\mathcal{T}_k^r,\mathcal{T}_j^r} * \mathcal{MV}_{j,k} = 0 \\ 0,\ otherwise \end{cases}$$

We define a (0,1) variable $\mathcal{LT}_i = 1$ if v_i is the last task mapped onto its processor, else 0. Therefore,

$$\mathcal{LT}_i = \begin{cases} 1,\ if\ \Sigma_{\forall v_k \in V^K, v_k \neq v_j} \mathcal{D}_{\mathcal{T}_i^r,\mathcal{T}_k^r} * \mathcal{MV}_{i,k} = 0 \\ 0,\ otherwise \end{cases}$$

We define a (0,1) variable $\mathcal{NW}_{i,j} = 1$ if v_j immediately follows v_i across iterations, else 0. Therefore,

$$\mathcal{NW}_{i,j} = \mathcal{LT}_i * \mathcal{FT}_j * \mathcal{MV}_{i,j}$$

As stated above, for v_j to immediately follow v_i, either $\mathcal{NI}_{i,j}$ or $\mathcal{NW}_{i,j}$ should be 1. Therefore,

$$\mathcal{N}_{i,j} = \mathcal{NI}_{i,j} + \mathcal{NW}_{i,j}$$

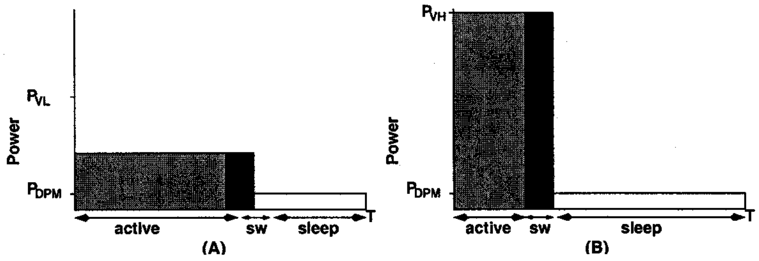


Fig. 8.2: Comparison of DVS policies: period constraint greater than task run time

Now, we can calculate switching overhead $\mathcal{SO}_i$ associated with each task $v_i \in V^K$ as follows: assuming that v_j immediately follows v_i, there will be a switching time overhead at the completion of v_i if v_i and v_j operate in different voltage states. Therefore,

$$\mathcal{SO}_i = \Sigma_{\forall v_j \in V^K, v_j \neq v_i} (\mathcal{N}_{i,j} * (1 - \mathcal{SV}_{i,j})) * \mathcal{T}_{c_k}$$

where $c_k = (s_i, s_j)$.

8.3.2.3. *Objective function.* The objective is to reduce the average power consumed by one steady state iteration of the schedule. Let TE represent energy associated with task execution, and CE represent the energy associated with communication.

- $TE = \Sigma_{\forall v_i \in V^K} \mathcal{TR}_i * \mathcal{PR}_i$
- $CE = \Sigma_{\forall v_i \in V^K} (\mathcal{TC}_i) * (\mathcal{P}_{mem} + \mathcal{P}_{bus})$

It is well known that between DVS and DPM, DVS should be exploited before DPM [87]. It can be explained with the examples shown in Figure 8.2. It shows the execution of a single task at two voltages V_L and V_H. The y-axis represents the power consumption, and x-axis denotes the time. In both cases, the task has the same period constraint T. The energy consumed by the task to perform useful computation at the higher voltage, $E(V_H) = P(V_H) \times t(V_H)$, is greater than the energy consumed by the task at the lower voltage, $E(V_L) = P(V_L) \times t(V_L)$. This follows

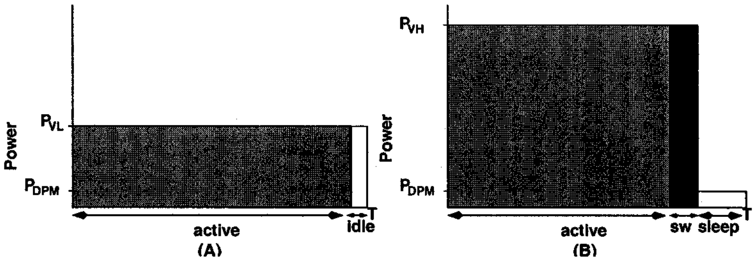


Fig. 8.3: Comparison of DVS policies: period constraint close to task run time

from the quadratic relation between operating voltage and the corresponding energy consumption ($P\,\alpha V^2 f, f\,\alpha V \Rightarrow t\,\alpha \frac{1}{V}, E = Pt \Rightarrow E\,\alpha\,V^2$). Consequently, the total energy consumption which includes the sleep mode energy at V_H is higher than V_L. Thus, DVS should be applied before DPM. The exception can occur when the task period is too tight that operating at the lowest voltage does not leave any time to go to the sleep mode (as shown in Figure 8.3). However, such cases are expected to be rare as the period should be just smaller than the execution time of the task at the lowest voltage plus the switching overhead. Hence, we always apply DVS first, and then DPM as a post optimization step.

Therefore, the objective is to minimize

$$\frac{TE + CE}{T^{K}_{period}}$$

DPM power is calculated as a post processing step. The input to the problem are the power consumption and run times of the tasks at different voltages on each processor. The objective of the techniques is to minimize the power consumption of the pipelined scheduling subject to the throughput and latency constraints. We calculate the power consumption of a schedule as the ratio of the total energy consumption of all the tasks over the throughput constraint.

8.3.2.4. *Constraints.*

- *Task-Processor mapping constraint*: A task should be mapped to only one processor. Therefore, for each task $v_i \in V^K$,

$$\Sigma_{\forall pe_j \in PE} \mathcal{M}_{i,j} = 1$$

- *Task-State mapping constraint*: A task should execute in only one state. Therefore, for each task $v_i \in V^K$,

$$\Sigma_{\forall s_j \in \mathcal{S}_{norm}} \mathcal{S}_{i,j} = 1$$

- *Task execution constraints*: Each task waits for the bus to be free, reads, executes and writes to the local memory, and then may or may not switch to a different state.

$$\mathcal{T}_i^{sr} \geq \mathcal{T}_i^{r} \tag{8.6}$$

$$\mathcal{T}_i^{se} = \mathcal{T}_i^{sr} + \mathcal{TC}_i \tag{8.7}$$

$$\mathcal{T}_i^{ee} = \mathcal{T}_i^{se} + \mathcal{TR}_i \tag{8.8}$$

$$\mathcal{T}_i^{es} = \mathcal{T}_i^{ee} + \mathcal{SO}_i \tag{8.9}$$

- *Dependence constraint*: Pipeline stage of a task is an indicator of the iteration to which the task belongs. A pipeline stage of 0 indicates that the task belongs to the current iteration. A pipeline stage of n indicates that the task belongs to n^{th} previous iteration from the current iteration of the schedule. Consider tasks v_i and v_j, $(v_i, v_j) \in E^K$. In the steady state execution of the schedule, v_j cannot start execution until v_i belonging to the same iteration of the schedule, has ended. Therefore, for each edge $(v_i, v_j) \in E^K$ belonging to the k^{th} instance of the task graph, $0 \leq k \leq K$, $ps_i^k \leq ps_j^k$,

$$\mathcal{T}_j^r + \mathcal{T}_{period}^K * (ps_j^k - ps_i^k) - \mathcal{T}_i^{ee} \geq 0$$

- *Latency constraints*: Every task $v_i \in V^K$ should end before its deadline. Hence, $\forall v_i \in V^K$,

$$\mathcal{T}_i^r + \mathcal{T}_{period}^K * ps_i^k \leq \mathcal{T}_{deadline}^k$$

- *Execution overlap constraints*: Given two tasks v_i and $v_j \in V^K$, if v_i starts before v_j, and both v_i and v_j are scheduled on the same processor, v_i should end before v_j.

  We define a (0,1) variable $\mathcal{EO}_{i,j}$ that is 1 if $\mathcal{D}_{\mathcal{T}_i^r,\mathcal{T}_j^r} = 1$ and $\mathcal{MV}_{i,j} = 1$. Therefore,

  $$\mathcal{EO}_{i,j} = \mathcal{D}_{\mathcal{T}_i^r,\mathcal{T}_j^r} * \mathcal{MV}_{i,j}$$

  $\mathcal{T}_i^{es}$ and $\mathcal{T}_j^r$ are related as follows: $\forall v_i, \forall v_j, v_i \neq v_j$,

  $$\mathcal{T}_i^{es} - \mathcal{T}_j^r \leq MAXVAL * (1 - \mathcal{EO}_{i,j})$$

  Therefore, if $\mathcal{EO}_{i,j} = 1$, task v_i has to end before task v_j starts reading. If $\mathcal{EO}_{i,j} = 0$ (task v_i starts after v_j, or they are mapped onto different processing elements), the equation will always hold.

- *Bus contention constraint*: Given two tasks v_i and $v_j \in V^K$, if v_i starts before v_j, and $\mathcal{C}_j = 1$, v_i should relinquish control of the bus before v_j starts reading. We define $\mathcal{BC}_{i,j} = 1$ if $\mathcal{C}_j = 1$ and $\mathcal{D}_{\mathcal{T}_i^r,\mathcal{T}_j^r} = 1$, else 0. Therefore,

  $$\mathcal{BC}_{i,j} = \mathcal{D}_{\mathcal{T}_i^r,\mathcal{T}_j^r} * \mathcal{C}_j \tag{8.10}$$

  Bus contention can thus be avoided by the following equation. $\forall v_i, \forall v_j, v_i \neq v_j$

  $$\mathcal{T}_i^{se} - \mathcal{T}_j^{sr} \leq MAXVAL * (1 - \mathcal{BC}_{i,j}) \tag{8.11}$$

Therefore, if $\mathcal{BC}_{i,j} = 1$, task v_i has to finish its read before task v_j starts reading. If $\mathcal{EO}_{i,j} = 0$ (task v_i starts after v_j, or v_j does not use the bus), the equation will always hold.

8.3.2.5. *Modifications for shared memory architectures.* The shared memory architecture in contrast to the distributed memory architecture utilizes a centralized shared memory for inter-processor communication. Thus, for each inter-processor communication operation, the producer task must first write to the shared memory, and the consumer task reads from the shared memory.

- *Task runtime instance variables for shared memory*: We define $\mathcal{T}_i^r$ to denote the time instance when task v_i is ready to start execution, $\mathcal{T}_i^{sr}$ to denote the time instance when v_i starts reading incoming edges, $\mathcal{T}_i^{se}$ to denote the time instance when v_i starts execution, $\mathcal{T}_i^w$ to denote the time instance when v_i has finished execution and written data to its local memory, $\mathcal{T}^{ee}$ to denote the time instance when v_i has written to the shared memory, and $\mathcal{T}_i^{es}$ to denote the time instance when a possible switching of states after the execution of v_i has occurred.

- *Bus access variable for shared memory*: We replace Equations 8.1-8.3 in the original formulation with the following equations. A task $v_i \in V^K$ utilizes the bus if at least one of its parents is scheduled on a different processor. We define a (0,1) variable $\mathcal{CR}_i$ such that $\forall v_i \in V^K$,

  $$\mathcal{CR}_i = \begin{cases} 1, & if\ v_i\ reads\ from\ the\ bus \\ 0, & otherwise \end{cases}$$

  The variable $\mathcal{CR}_i$ can be derived as follows: $\forall v_i \in V^K$,

  $$\forall (v_j, v_i) \in E^K, \mathcal{CR}_i \geq (1 - \mathcal{MV}_{i,j})$$

  $$\mathcal{CR}_i \leq \sum_{\forall (v_j, v_i) \in E^K} (1 - \mathcal{MV}_{i,j})$$

  The first equation sets $\mathcal{CR}_i$ to 1 if v_i and any parent v_j are scheduled on different processors, and the second equation sets $\mathcal{CR}_i$ to 0 if v_i and all its parents are scheduled on the same processor.

- *Read overhead per edge for shared memory*: We replace the Equation 8.4 in the original formulation with the following equations. Let the read overhead of each edge $e_k = (v_j, v_i) \in E^K$ from shared memory be denoted by $\mathcal{TCR}_{i,j}$. $\mathcal{TCR}_{i,j}$ will be the time overhead associated with transfer of data from the shared memory to the local memory of the consumer processor.

Thus,

$$\mathcal{TCR}_{i,j} = \begin{cases} 0,\ if \mathcal{MV}_{i,j} = 1 \\ (\mathcal{T}_{bus} + \mathcal{T}_{mem}) \cdot \omega(e_k),\ otherwise \end{cases}$$

- *Read and write overhead per task for shared memory*: We replace the Equation 8.5 in the original formulation with the following equations. Let the total shared memory read overhead of task v_i be denoted by $\mathcal{TCR}_i$. Therefore, $\mathcal{TCR}_i$ will be the sum of the read overhead due to all incoming edges of v_i. Mathematically, $\forall i \in V^K$

  $$\mathcal{TCR}_i = \sum_{\forall v_j, (v_j, v_i) \in E^K} (\mathcal{TCR}_{i,j}) * (1 - \mathcal{MV}_{i,j})$$

  Similarly, total shared memory write overhead per task can also be established. For task v_i let it be denoted by $\mathcal{TCW}_i$. We define the shared memory overhead for a task v_i as

  $$\mathcal{TC}_i = \mathcal{TCR}_i + \mathcal{TCW}_i$$

- *Task execution constraints for shared memory*: We replace Equations 8.6-8.9 in the original formulation with the following equations. Each task waits for the bus to be free, reads, executes and writes to the local memory or shared memory, and then may or may not switch to a different state. This can be represented as follows: $\forall v_i \in V^K$,

  $$\mathcal{T}_i^{sr} \geq \mathcal{T}_i^{r}$$

  $$\mathcal{T}_i^{se} = \mathcal{T}_i^{sr} + \mathcal{TCR}_i$$

  $$\mathcal{T}_i^{w} = \mathcal{T}_i^{se} + \mathcal{TR}_i$$

  $$\mathcal{T}_i^{ee} \geq \mathcal{T}_i^{w} + \mathcal{TCW}_i$$

  $$\mathcal{T}_i^{es} = \mathcal{T}_i^{ee} + \mathcal{SO}_i$$

- *Bus contention constraint for shared memory*: We replace Equations 8.10-8.11 in the original formulation with the following equations. Given any two tasks v_i and $v_j \in V^K$, if v_i accesses the bus before v_j, and $\mathcal{CR}_j = 1$, v_i should relinquish control of the bus before v_j obtains access of the bus. We define a variable $\mathcal{BC}_{i,j}^{r,r}$ to denote if task v_i accesses the bus for reading before task v_j. Thus,

$$\mathcal{BC}_{i,j}^{r,r} = \begin{cases} 1, \; if \; \mathcal{CR}_j = 1 and \, \mathcal{T}_i^r, \mathcal{T}_j^r = 1 \\ 0, \; otherwise \end{cases}$$

  Similarly, we can define $\mathcal{BC}_{i,j}^{w,r}$, $\mathcal{BC}_{i,j}^{w,w}$ and $\mathcal{BC}_{i,j}^{r,w}$. We have the following cases:

  - If v_i begins reading from the shared memory before v_j begins reading from the shared memory, the bus contention can be avoided by the following equation:

$$\forall v_i, \forall v_j, v_i \neq v_j, \mathcal{T}_i^{se} - \mathcal{T}_j^{sr} \leq MAXVAL * (1 - \mathcal{BC}_{i,j}^{r,r})$$

  - If v_i begins writing to the shared memory before v_j begins reading from the shared memory, the bus contention can be avoided by the following equation:

$$\forall v_i, \forall v_j, v_i \neq v_j, \mathcal{T}_i^{ee} - \mathcal{T}_j^{sr} \leq MAXVAL * (1 - \mathcal{BC}_{i,j}^{w,r})$$

  - If v_i begins writing to the shared memory before v_j begins writing to the shared memory, the bus contention can be avoided by the following equation:

$$\forall v_i, \forall v_j, v_i \neq v_j, \mathcal{T}_i^{ee} - \mathcal{T}_j^{w} \leq MAXVAL * (1 - \mathcal{BC}_{i,j}^{w,w})$$

  - If v_i begins reading from the shared memory before v_j begins writing to the shared memory, the bus contention can be avoided by the following equation:

$$\forall v_i, \forall v_j, v_i \neq v_j, \mathcal{T}_i^{se} - \mathcal{T}_j^{w} \leq MAXVAL * (1 - \mathcal{BC}_{i,j}^{r,w})$$

8.3.2.6. *Techniques to linearize relevant nonlinear equations.* The above given formulation incorporates non-linear equations in many places. In this subsection, we provide methods to linearize such equations.

- Given that $a_0, a_1, \cdots a_{n-1}$ are n (0,1) elements, we can linearize a non linear equation ($z = a_0 * a_1 ... a_{n-1}$) by applying the following constraints.

  $$0 \leq z \leq 1$$

  $$a_0 + a_1 + a_2 .. + a_{n-1} \geq n * z$$

  $$a_0 + a_1 + a_2 .. + a_{n-1} \leq z + n - 1$$

  This equivalence of the non-linear and linear equations can be easily proved by mathematical induction.

- Given an integer x, we can linearize $y = 1$ if $x > 0$, else 0 as follows:

  $$0 \leq y \leq 1,$$

  $$-x - (1 - y) * MAXVAL \leq -1,$$

  $$x - y * MAXVAL \leq 0$$

- Given two variables x and y where x is a (0, 1) variable and $y \geq 0$, we can linearize a non linear equation $z = x * y$ as follows:

  $$0 \leq z \leq y$$

  $$z - (x - 1) * MAXVAL \geq y$$

  $$-z + x * MAXVAL \geq 0$$

8.3.3. MILP formulation that assumes a constant switching overhead ($MILP_{so}$)

We relaxed the switching overhead conditions and assumed that every time a task finishes execution, the processor switches from one state to another. Hence, completion of every task is accompanied by a time overhead (> 0) for switching. In mathematical terms,

$$\forall v_i \in V^K, \mathcal{SO}_i = \mathcal{T}_{c_k}$$

Since assumption of constant switching overhead reduces the solution space to be searched, the run time of the formulation is reduced. System level power minimization techniques are applied in conjunction with minimization of communication power. Average power consumption of the schedule is calculated during post processing with re-adjustment and elimination of extra switching overhead. The drawback of the formulation is that it may invalidate certain solutions due to violation of timing constraints because of the assumption of switching overhead.

8.3.4. MILP formulation that maximizes task run time ($MILP_{lat}$)

In this formulation, instead of minimizing power, we maximize the cumulative task run time. The objective function is:

$$maximize \sum_{\forall v_i \in V^K} \mathcal{TR}_i$$

This formulation does not search solutions related to calculation of power and hence, has a lesser run time compared to the optimal formulation. Average power consumption of the solution (schedule) is calculated as a post processing step. The motivation for this heuristic is that maximizing task execution time implies application of DVS, which is a powerful method for minimizing system level power consumption. The formulation does not minimize communication power. Therefore, the resulting solution might consume more power due to communication.

8.3.5. MILP formulation combining $MILP_{lat}$ and $MILP_{so}$ ($MILP_{ls}$)

In this formulation, we combined the $MILP_{lat}$ and $MILP_{so}$ formulations. This formulation maximizes task run times and assumes a constant switching time overhead on completion of each task. The formulation does not use either power related variables or switching overhead related variables. The correct power consumption of the final schedule is calculated as a post processing step.

8.4. Heuristic Techniques for System-level Low Power Design

In this section, we present our heuristic techniques for system-level low power design. We first discuss the base algorithms that we utilize for pipelined scheduling and loop unrolling and follow it up with a detailed presentation of our heuristic optimization techniques.

8.4.1. Overview of the loop transformation algorithms

Our techniques are built upon well known functional pipelining [113] and loop unrolling [90] algorithms.

8.4.1.1. *Functional Pipelining:* Functional pipelining [113] is a well known resource constrained pipelined scheduling technique that has been utilized for operation scheduling in very large instruction word (VLIW) processors. It is aimed solely at throughput optimization and does not consider power minimization. It utilizes list-based scheduling and a fixed period schedule table to obtain a pipelined schedule. If a task during list based scheduling violates the period constraint, the task is "wrapped" around the schedule table to an earlier time slot that is free, thus introducing a pipeline stage. The scheduling algorithm accounts for data dependency and latency constraints that were described in Section 8.3. Our heuristic strategy integrates a design space exploration stage (aimed at processor mapping and voltage selection) with the functional pipelining technique to produce low power pipelined designs.

8.4.1.2. *Loop Unrolling:* The loop unrolling transformation replicates the task graph multiple times [90]. Similar, to functional pipelining loop unrolling is not particularly aimed at power minimization. The sole objective of loop unrolling is to increase the design space that can then be exploited to achieve the optimization goal of the problem at hand. Our technique integrates the unrolling transformation with the goal of minimizing the power consumption. Unrolling the task graph leads to transformation of the original task graph $G(V, E)$ to a new task graph $G'(V', E')$ (as described in Section 8.3) that includes multiple instances of the original graph.

8.4.2. Overall heuristic for system-level low power design

The flowchart for the overall heuristic technique for system-level low power design is shown in Figure 8.4. The shaded boxes in the figure denote deterministic optimization or simulated annealing specific operations that are addressed in following sections. As shown in the figure the technique consists of two basic stages that generate low power pipelined designs (shown in the left hand side of the figure) and unroll the task graph (shown in the right hand side of the figure), respectively. The system-level low power design heuristic applies an iterative strategy to design space exploration for generating optimized implementations. The outer loop applies low power pipelined scheduling and unrolling. The inner loop which is part of low power pipelined scheduling consists of functional pipelining and task operating voltage selection. In the following few paragraphs we discuss the various steps of the iterative design space exploration in further detail.

We address the processor mapping and voltage selection for all tasks in a graph as a partitioning problem. We consider each alternative processor and operating voltage as a partition that a particular task can be assigned to. We evaluate a particular design by applying functional pipelining, followed by DPM. We perform design space exploration by modifying an existing partition by application of moves on tasks. A move changes the processor and/or operating voltage

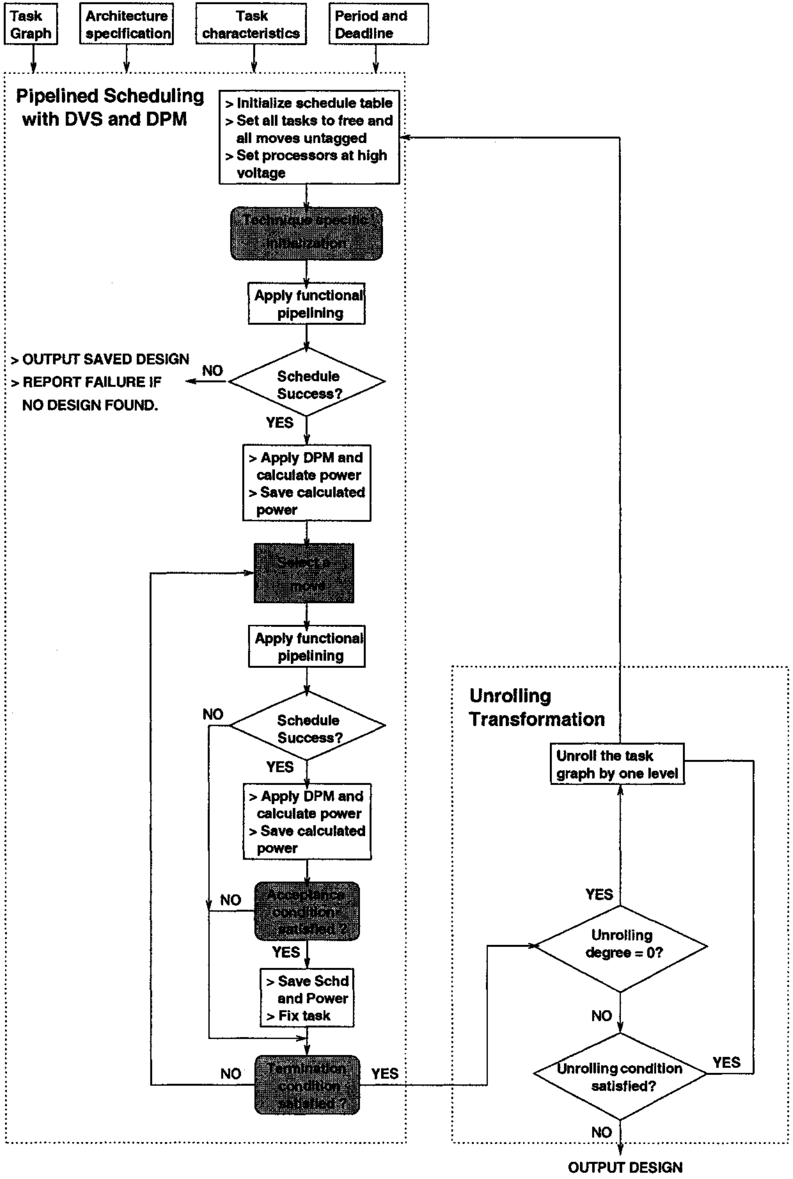

Task Graph
Architecture specification
Task characteristics
Period and Deadline
Pipelined Scheduling with DVS and DPM
> Initialize schedule table
> Set all tasks to free and all moves untagged
> Set processors at high voltage
Technique specific initialization
Apply functional pipelining
Schedule Success?
NO
> OUTPUT SAVED DESIGN
> REPORT FAILURE IF NO DESIGN FOUND.
YES
> Apply DPM and calculate power
> Save calculated power
Select a move
Apply functional pipelining
Schedule Success?
NO
YES
> Apply DPM and calculate power
> Save calculated power
Acceptance condition satisfied ?
NO
YES
> Save Schd and Power
> Fix task
Termination condition satisfied ?
NO
YES
Unrolling Transformation
Unroll the task graph by one level
YES
Unrolling degree = 0?
NO
Unrolling condition satisfied?
YES
NO
OUTPUT DESIGN

Fig. 8.4: Heuristic Technique for System-level Low Power Design

of a task. In the case of deterministic heuristic strategy (discussed in a following section) we consider several moves, and select the best move which results in the largest reduction in power consumption. In the simulated annealing based heuristic (discussed in a following section) we also accept moves which result in increased power consumption so that we can climb out of local minima.

During the execution of our heuristic optimization strategy several moves could be applied to a task. A "move" denotes modification of the operating voltage and/or processor mapping of the task from the current solution. Therefore, a move specifies the new processor and corresponding operating voltage. A move for a particular task during execution of our heuristic technique could be either "free", "tagged" or "fixed". A move is said to be free if it can be applied to the task. A move is said to be tagged if it cannot be applied to the task. If a task is tagged for a particular move, an alternative free move can be explored for the task. Finally, a move is said to be fixed if the processor and corresponding operating voltage have been established for the task. If a task is fixed for a particular move it is always scheduled on the specified processor and operating voltage by the functional pipelining algorithm.

Initially, the time constraint on the schedule table is set equal to the reciprocal of the target throughput multiplied by the number of instances of the task graph. The number of instances of the task graph are determined by the unrolling degree which is initially set to zero. The number of instances of the task graph increase as the unrolling degree is increased in subsequent iterations of the outer loop. All processors are set to the highest operating voltage (peak performance) and all moves for all tasks are free. The technique then applies list scheduling based functional pipelining [113] to evaluate the throughput. The list-based scheduling algorithm selects the task with the maximum urgency from the ready list. The urgency of a task v_i is given by

$$urgency(v_i) = \mathcal{T}_i + max_{\forall (v_i,v_j)\in E}(urgency(v_j))$$

where $\mathcal{T}_i$ is the execution time of task v_i. The urgency heuristic has been widely utilized in literature [113]. Initially all tasks are free (not fixed). The functional pipelining technique schedules the selected task on a (any) processor that permits the execution of the task at the lowest possible pipeline stage (first priority), and earliest possible schedule time (second priority) subject to the data dependence constraints. If the period and deadline constraints are not satisfied, the technique declares that the task graph cannot be scheduled and exits. If the timing constraints are satisfied, the technique applies DPM to the schedule and calculates the average power consumption.

The technique then performs automated design space exploration by application of moves. A move changes the processor and the operating voltage that a task is mapped to by selecting (possibly) another processor at a different operating voltage. The technique selects a move (based on deterministic optimization or simulated annealing strategy) and verifies if the timing constraint is satisfied by functional pipelining. During functional pipelining, the task is scheduled on the processor and operating voltage state specified by the move. If the performance constraints are not satisfied, the task is marked as tagged for that particular move and another move is selected. The iterative pipelined scheduling technique is repeated. If constraints are satisfied, the technique goes ahead and computes the total power consumed by the schedule. At this stage, the technique checks if the move acceptance conditions are satisfied. The conditions vary based on whether we utilize a deterministic or simulated annealing based optimization strategy and are discussed in the following sections. If the move is accepted, the task is fixed on the selected processor at the chosen voltage state. Otherwise, the task is tagged for that move and a new move is explored. Once the task has been fixed, the functional pipelining technique always schedules the task on the assigned processor and operating voltage state. The termination condition of the inner low power pipeline design loop also depends on the optimization strategy.

On termination of the inner loop of iterative functional pipelining, the outer loop consisting of unrolling transformation is initiated. Unrolling transformation is applied the first time when the unrolling degree is zero. If the unrolling degree is greater than zero, we check if the current schedule obtained due to the unrolling transformation resulted in a power reduction of greater than 1% over the previous schedule. If the condition is true another unrolling transformation is applied. Alternatively, the technique exits and outputs the current schedule as the final design.

8.4.3. Deterministic optimization for system-level low power design

The flowchart for deterministic optimization for system-level low power design is shown in Figure 8.5. The refinements in comparison to the overall heuristic discussed in the previous section are denoted by shaded boxes. The deterministic heuristic selects a "move" based on the gain function. Assume that in the current schedule a task v_i is assigned to processor pe_j operating at a voltage state $s_k^{norm} \in S_{norm}$, and executes in $\tau(v_i, pe_j, s_k^{norm})$ time units. The power consumption of the task is $\rho(v_i, pe_j, s_k^{norm})$ power units. Let *cschd* denote this schedule. Assume that the task is moved to a processor $pe_m \in PE$ operating at voltage state $s_n^{norm} \in S_{norm}$, such that the task has an execution time of $\tau(v_i, pe_m, s_n^{norm})$ time units and power consumption of $\rho(v_i, pe_m, s_n^{norm})$ power units. The anticipated schedule is a schedule that incorporates this move, while keeping all other task-processor-operating state mappings as in the current schedule. Let *aschd* denote the anticipated schedule. The gain between the two schedules is given by the difference between the power consumption of the current schedule and the anticipated schedule. The magnitude of the gain due to the move depends upon two factors: (i) the change in power consumption due to the execution of the task at a different processor-operating voltage, and (ii) the overhead due to communication if the move results in the task being mapped to a different processor. Let $M_{i,j,schd}$ be 1 if tasks v_i and v_j are mapped to different processors in a schedule *schd*, and $(v_i, v_j) \in E$ else

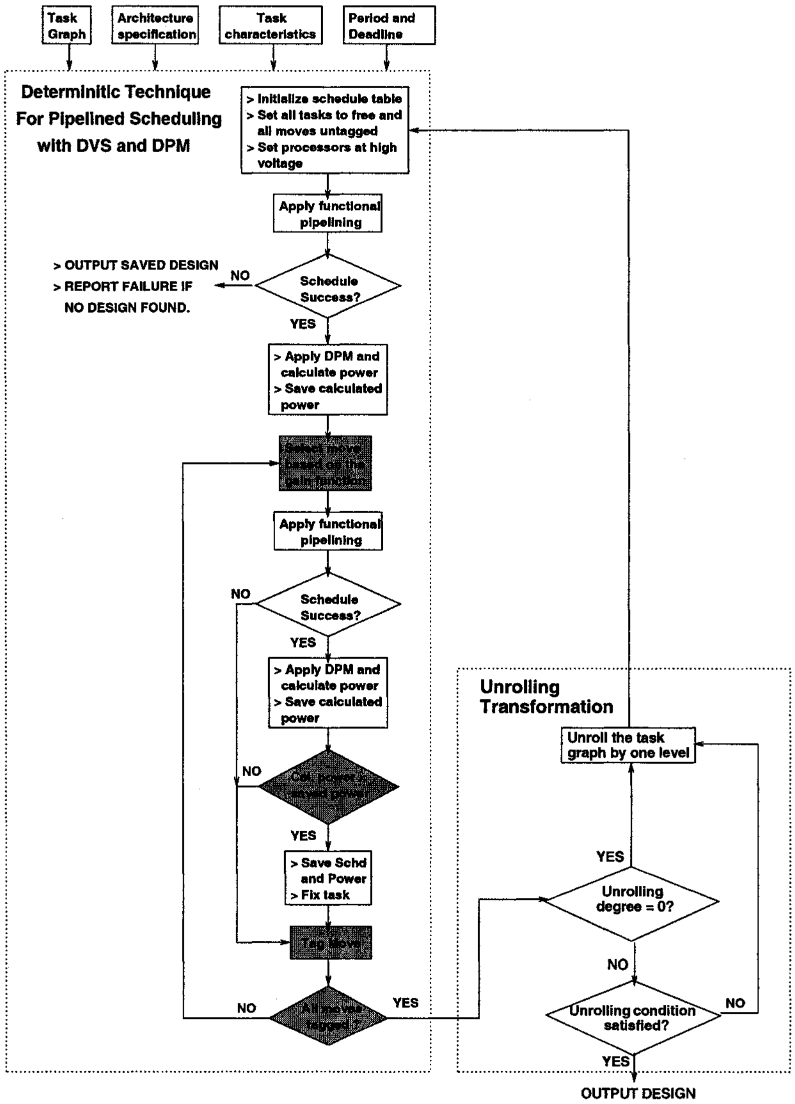

Task Graph
Architecture specification
Task characteristics
Period and Deadline
Deterministic Technique For Pipelined Scheduling with DVS and DPM
> Initialize schedule table
> Set all tasks to free and all moves untagged
> Set processors at high voltage
Apply functional pipelining
> OUTPUT SAVED DESIGN
> REPORT FAILURE IF NO DESIGN FOUND.
NO
Schedule Success?
YES
> Apply DPM and calculate power
> Save calculated power
Select move based on the gain function
Apply functional pipelining
NO
Schedule Success?
YES
> Apply DPM and calculate power
> Save calculated power
NO
Cal. power < saved power
YES
> Save Schd and Power
> Fix task
Tag Move
NO
All moves tagged ?
YES
Unrolling Transformation
Unroll the task graph by one level
YES
Unrolling degree = 0?
NO
Unrolling condition satisfied?
NO
YES
OUTPUT DESIGN

Fig. 8.5: Deterministic Optimization for System-level Low Power Design

0. Therefore, the gain will be given by

$$\begin{aligned} gain = \frac{1}{\mathcal{T}_{period}} \{ & \rho(v_i, pe_j, s_k^{norm}) \cdot \tau(v_i, pe_j, s_k^{norm}) \\ & - \rho(v_i, pe_m, s_n^{norm}) \cdot \tau(v_i, pe_m, s_n^{norm}) \\ & + \sum_{\forall (v_i, v_l) \in E} M_{i,l,cschd} \cdot \omega_{(v_i,v_l)} \cdot \rho_{comm} \cdot \tau_{comm} \\ & + \sum_{\forall (v_l, v_i) \in E} M_{l,i,cschd} \cdot \omega_{(v_l,v_i)} \cdot \rho_{comm} \cdot \tau_{comm} \\ & - \sum_{\forall (v_i, v_l) \in E} M_{i,l,aschd} \cdot \omega_{(v_i,v_l)} \cdot \rho_{comm} \cdot \tau_{comm} \\ & - \sum_{\forall (v_l, v_i) \in E} M_{l,i,aschd} \cdot \omega_{(v_l,v_i)} \cdot \rho_{comm} \cdot \tau_{comm} \} \end{aligned}$$

$$where\ \rho_{comm} = \rho_{bus} + \rho_{mem}\ and\ \tau_{comm} = \tau_{bus} + \tau_{mem}$$

The technique selects a move with the largest possible gain and verifies if the timing constraint is satisfied by functional pipelining. If the timing constraints are satisfied, the technique determines the power consumption of the schedule by application of DPM. The technique accepts a move if the average power consumed by the new schedule is less than the average power consumed by the previous schedule. The deterministic low power pipelined scheduling technique does not explore moves on tasks that are fixed, and moves that have been tagged. The move selection and scheduling procedure concludes when all tasks are fixed or all moves have been tagged. The complexity of functional pipelining is $O(n^2)$ where n is the number of tasks. Let $|PE|$ denote the number of processing elements, and $|S^{norm}|$ denote the number of operating states of each processing element. The maximum number of possible moves are $|PE| * |S^{norm}| * n$. Hence, the complexity of the low power functional pipelining technique is $O(|PE| * |S^{norm}| * n^3)$.

In the results section we denote the deterministic low power system-level design technique (with unrolling) as $LPPWU_{det}$, and low power pipelined scheduling technique (shown in the left hand side of Figure 8.5, without unrolling) as LPP_{det}.

8.4.4. Simulated annealing based optimization for Low Power Design

We utilize the simulated annealing technique [114] to develop an optimization strategy for low power design. Simulated annealing is a generic technique that has been widely applied for a large number of optimization problems. The simulated annealing technique as the name suggests models the annealing schedule that is utilized to improve the strength of metals. The basic technique includes a temperature variable that is decreased from a large to a very small value in finite number of small steps. At each temperature value the technique modifies the current solution to generate a prospective new solution. The new solution is accepted if the cost of the solution is better than the current solution or the following condition is satisfied $r < e^{D/T}$ where r is a random number in $[0, 1]$, T is the current temperature and D is the difference between the cost of the new and current solution, respectively. At higher values of the temperature variable a new solution that increases the cost is accepted with a higher probability. As the temperature reduces, the probability of accepting a solution that increases the cost also falls. We integrated the simulated annealing strategy with our heuristic.

The flowchart for the technique is shown in Figure 8.6. Again, the refinements in comparison to the overall heuristic discussed in Section 8.4.2 are shown as shaded boxes. The simulated annealing based optimization maintains a temperature variable that is initialized to a maximum value. A task is chosen at random and a move on the task is chosen, again at random. Functional pipelining is applied and the power consumption of the schedule is calculated. If the timing constraint is satisfied, the following acceptance test is made:

- If calculated power is less than the power consumed by the previous schedule, set acceptance condition to “satisfied”.
- If calculated power is greater than or equal to the power consumed by the previous schedule, generate a random number r such that $(0 < r < 1)$. Set acceptance condition to “satisfied”

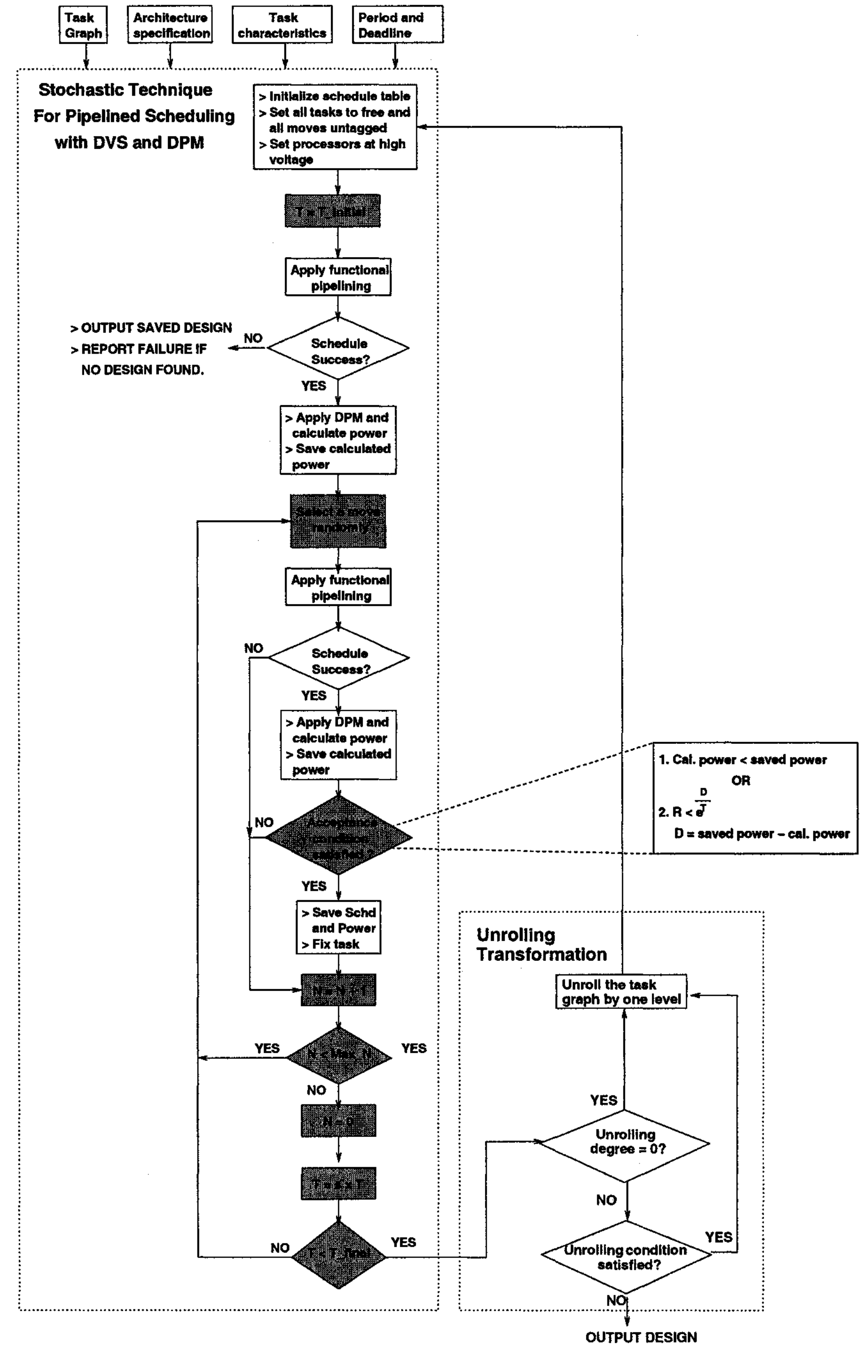

Task Graph
Architecture specification
Task characteristics
Period and Deadline
Stochastic Technique For Pipelined Scheduling with DVS and DPM
> Initialize schedule table
> Set all tasks to free and all moves untagged
> Set processors at high voltage
T = T_initial
Apply functional pipelining
Schedule Success?
NO
> OUTPUT SAVED DESIGN
> REPORT FAILURE IF NO DESIGN FOUND.
YES
> Apply DPM and calculate power
> Save calculated power
Select a move randomly
Apply functional pipelining
Schedule Success?
NO
YES
> Apply DPM and calculate power
> Save calculated power
Acceptance condition satisfied ?
NO
YES
1. Cal. power < saved power
OR
2. $R < e^{\frac{D}{T}}$
D = saved power – cal. power
> Save Schd and Power
> Fix task
N = N + 1
N < Max_N
YES
YES
NO
N = 0
T = a x T
T < T_final
NO
YES
Unrolling Transformation
Unroll the task graph by one level
YES
Unrolling degree = 0?
NO
Unrolling condition satisfied?
YES
NO
OUTPUT DESIGN

Fig. 8.6: Simulated annealing based optimization for low power design

if

$$r < e^{\frac{\rho(previous\ schedule)-\rho(new\ schedule)}{T_{current}}}$$

where $\rho(previous\ schedule)$ is the average power consumption of the previous schedule, $\rho(new\ schedule)$ is the average power consumption of the new schedule, and $T_{current}$ is the current temperature.

- If neither of the above two conditions are met, set acceptance condition to "not satisfied".

The new schedule is accepted if the acceptance condition is satisfied. At every temperature, a fixed number of moves z are explored, where z is a function of number of tasks in the graph. Unlike the deterministic heuristic, the simulated annealing based technique does not tag any unsuccessful move, and permits exploration on the same move in successive iterations. Further, the simulated annealing based technique also permits exploration of moves on fixed tasks. Hence, a particular task that has been assigned or fixed to a low power state on a processor can be re-assigned (or fixed) to a different state on an alternative processor. Once the z moves are explored, the temperature is scaled by s $(0 < s < 1)$. This outer loop is repeated as long as the current temperature is greater than minimum temperature. When current temperature becomes less than final temperature, the loop unrolling transformation is applied.

In the results section we denote the simulated annealing based low power system-level design technique (with unrolling) as $LPPWU_{sa}$, and low power pipelined scheduling technique (shown in the left hand side of Figure 8.6, without unrolling) as LPP_{sa}.

8.5. Results

This section presents and analyzes the results of experimentation that was performed to evaluate our techniques. In the following discussion we first discuss the experimental set-up that

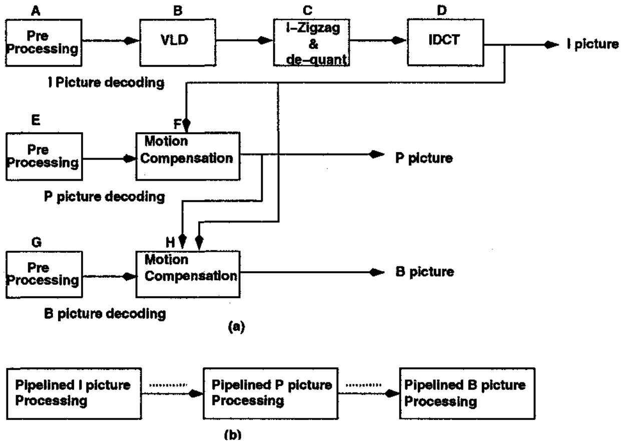

A
Pre Processing
B
VLD
C
I-Zigzag & de-quant
D
IDCT
I picture
I Picture decoding
E
Pre Processing
F
Motion Compensation
P picture
P picture decoding
G
Pre Processing
H
Motion Compensation
B picture
B picture decoding
(a)
Pipelined I picture Processing
Pipelined P picture Processing
Pipelined B picture Processing
(b)

Fig. 8.7: MPEG decoder task flow

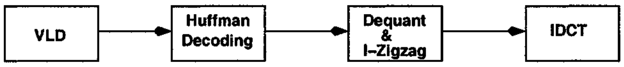

VLD
Huffman Decoding
Dequant & I-Zigzag
IDCT

Fig. 8.8: JPEG decoder task flow

includes applications, target architecture, and existing techniques from previous research that were utilized for conducting the experiments.

8.5.1. Experimental set-up

8.5.1.1. *Multimedia applications and synthetic task graphs.* We evaluated the performance of our techniques by experimenting with multimedia applications and synthetic task graphs, respectively.

Multimedia applications. We obtained system-level low power designs of JPEG decoder, MPEG-1 decoder and MP3 encoder on two-processor architectures. We collected results for different period constraints, with deadline set at 1.5 and 3 times the corresponding period constraint, respectively. The following paragraphs give a brief overview of the three applications.

JPEG Decoder : The JPEG decoding task graph is shown in Figure 8.8. The JPEG decoding

PCM
Filtering
Huffman Encoding

Fig. 8.9: MP3 encoder task flow

algorithm was modeled by four tasks consisting of: variable length decoding, huffman decoding, inverse-zigzag and quantization, and IDCT.

MPEG-1 Decoder : The MPEG-1 decoder task graph is shown in Figure 8.7. An MPEG-1 stream consists of I, P and B pictures. Decoding an I picture consists of the following tasks: preprocessing, variable length decoding, inverse zigzag and dequantization, and IDCT. P and B picture decoding consist of preprocessing and motion compensation. An I picture can be decoded independently. There exists a data dependence from an I picture to a P picture, and from I and P pictures to a B picture. A typical MPEG-1 encoded media-stream will have the I picture, followed by one or more dependent P pictures, and followed by one or more dependent B pictures. For example, {...I..PP..BBBB..P..BB... I...}. The MPEG-1 decoder can therefore be broken down into three separate task graphs (one each for I, P and B picture, respectively) that execute in mutual exclusion to satisfy the data dependence constraints. Every task graph operates on a stream of macroblocks to construct each I, P and B picture, respectively. Therefore, we can explore loop transformations for each task graph individually. A measure of power consumption can be obtained by adding the individual average power consumption of each task graph.

MP3 Encoder : The MP3 encoding task graph is shown in Figure 8.9. The MP3 encoding algorithm was modelled by three tasks consisting of: pulse code modulation, filtering, and huffman encoding.

Synthetic task graphs. We constructed synthetic task graphs that had the following structures: FFT, intree, outtree, fork-join, and meanvalue. The size of the task graphs was randomly varied from 10 to 40 nodes, the execution times from 200 to 500 ms, and the edge weights from 2 to 128 bytes. The run time and power of the nodes at multiple voltages was obtained by scaling of the randomly generated execution time value by a constant. Multiple task graph configurations were generated by combining any two of the above mentioned graph structures.

8.5.1.2. *Target architecture and characterization.* We consider an architecture consisting of multiple Intel StrongArm 1100 processors that inter-processor communication through distributed memory as described in Section 8.1.1 and Figure 8.1. We obtained execution times (CPU run time) and average power consumption estimates of the tasks multimedia applications by utilizing the JouleTrack simulator [115] for the StrongArm processor. The processor was run at the following specifications: 1.5 V - 206 MHz, 1.4 V - 192 MHz, 1.2 V - 162 MHz and 1.1 V - 133 MHz. The power consumption of the processor in the sleep state was assumed to be 64mW [116]. We assume that the time overhead to switch from any active state to the sleep state is 5 ms. The time overhead to switch between any two active states is assumed to be 50 μs. We estimated the power overhead for switching between voltage states $\mathcal{P}_{c_k}$ to be equal to the power consumption associated with the higher voltage state. We assumed a system bus with a 20pF per bit line capacitance, an operating voltage of 3.3 V and a wire switching frequency of 50MHz. The average power consumption of the bus was estimated to be 147mW by utilizing the model described by Wuytack et al. [117]. The size of each local memory in the architecture was assumed to be 256KB, with an average power consumption of 260mW [117].

8.5.1.3. *Existing techniques from previous research.* We compared our solutions against two existing techniques: i) a functional pipelining technique (FP) that applies DPM, but no DVS and ii) a technique based on the static scheduling scheme, described by Luo et al. [101], that applies DVS and DPM, but no loop transformations (LPS). We selected the techniques based on the current state of the art. Functional pipelining is a well known technique for obtaining pipelined design. Therefore, we generate the best design possible and apply DPM at the end. The LPS technique have been proposed for power minimization of periodic embedded system applications on

Techniques	% Power Reduction			
	Values	JPEG	MPEG-1	MP3
$MILP$ v/s FP	**Average**	**38.7**	**36.77**	**27.31**
	Std. dev.	7.09	17.66	17.76
$MILP$ v/s LPS	**Average**	**NA**	**NA**	**7.02**
	Std. dev.	NA	NA	4.06
$MILPU$ v/s FP	**Average**	**56.55**	**48.20**	**47.29**
	Std. dev.	7.06	4.17	1.01
$MILPU$ v/s LPS	**Average**	**NA**	**NA**	**29.69**
	Std. dev.	NA	NA	15.99
$MILPU$ v/s $MILP$	**Average**	**29.61**	**14.65**	**24.17**
	Std. dev.	6.55	13.30	18.24

Table 21: Summary of results for $MILP$

Techniques	% Power Reduction			
	Values	JPEG	MPEG-1	MP3
$MILP_{so}$ v/s FP	**Average**	**35.27**	**22.35**	**16.96**
	Std. dev.	7.39	19.66	30.42
$MILP_{so}$ v/s LPS	**Average**	**NA**	**NA**	**-4.46**
	Std. dev.	NA	NA	14.65
$MILPU_{so}$ v/s FP	**Average**	**55.31**	**37.33**	**47.09**
	Std. dev.	6.86	8.31	1.04
$MILPU_{so}$ v/s LPS	**Average**	**NA**	**NA**	**29.49**
	Std. dev.	NA	NA	15.69
$MILPU_{so}$ v/s $MILP_{so}$	**Average**	**30.48**	**15.77**	**30.75**
	Std. dev.	5.39	14.52	20.45

Table 22: Summary of results for $MILP_{so}$

Techniques	% Power Reduction			
	Values	JPEG	MPEG-1	MP3
$MILP_{lat}$ v/s FP	**Average**	**34.99**	**22.83**	**20.78**
	Std. dev.	6.29	21.23	18.00
$MILP_{lat}$ v/s LPS	**Average**	**NA**	**NA**	**-1.73**
	Std. dev.	NA	NA	6.05
$MILPU_{lat}$ v/s FP	**Average**	**53.68**	**38.14**	**45.58**
	Std. dev.	11.88	13.72	1.08
$MILPU_{lat}$ v/s LPS	**Average**	**NA**	**NA**	**27.41**
	Std. dev.	NA	NA	16.51
$MILPU_{lat}$ v/s $MILP_{lat}$	**Average**	**28.49**	**16.67**	**28.67**
	Std. dev.	14.08	16.25	15.60

Table 23: Summary of results for $MILP_{lat}$

Techniques	% Power Reduction			
	Values	JPEG	MPEG-1	MP3
$MILP_{ls}$ v/s FP	**Average**	**31.47**	**15.92**	**0.41**
	Std. dev.	6.87	13.18	30.42
$MILP_{ls}$ v/s LPS	**Average**	**NA**	**NA**	**-26.54**
	Std. dev.	NA	NA	12.03
$MILPU_{ls}$ v/s FP	**Average**	**52.59**	**33.46**	**45.38**
	Std. dev.	12.19	5.88	1.08
$MILPU_{ls}$ v/s LPS	**Average**	**NA**	**NA**	**27.20**
	Std. dev.	NA	NA	16.21
$MILPU_{ls}$ v/s $MILP_{ls}$	**Average**	**30.30**	**18.97**	**41.84**
	Std. dev.	13.67	14.98	14.62

Table 24: Summary of results for $MILP_{ls}$

multiprocessor architectures. We compared our formulations against the technique to demonstrate that under tight period constraints, it may not be possible to obtain valid schedules for periodic applications without pipelining and unrolling. LPS and FP were implemented in C and executed on our benchmarks to generate the comparative data. In addition to comparing against existing research, we also evaluated the quality of our two heuristic techniques by comparison with the optimal $MILP$ formulation.

8.5.2. Evaluation of MILP formulations

In this section, we present the results for experiments that were performed to evaluate our MILP based techniques. We evaluated our MILP based techniques by mapping multimedia applications on dual processor board level architectures as described above. The complexity of

MILP based techniques grows exponentially with the size of the input. Therefore, the larger sized synthetic task graphs were not utilized for experimentation. We obtained solutions to the MILP formulations by submitting them to the Xpress-MP simplex solvers installed in the NEOS servers [118] [119]. Hereafter, it will be assumed that the $MILP$, $MILP_{so}$, $MILP_{lat}$ and $MILP_{ls}$ denote formulations that are applied on task graphs that are not unrolled. When these formulations are applied on unrolled task graphs, they will be denoted as $MILPU$, $MILPU_{so}$, $MILPU_{lat}$ and $MILPU_{ls}$, respectively.

In the Appendix, Figures A.1-A.4, A.5-A.8, and A.9-A.12 plot the results of the MILP based formulations ($MILP$, $MILP_{lat}$, $MILP_{so}$, $MILP_{ls}$) for JPEG, MPEG-1 and MP3 applications, respectively. The x-axis plots the period constraints that were utilized for the experiments, and the left hand side y-axis plots the normalized power consumption. The power consumption for the different period constraints is normalized with the power consumed by FP based solution for the lowest period constraint. The unrolling degree of the final solution is also plotted by a line graph on the right hand side y-axis. In each of these figures, plot (a) refers to deadline equal to 1.5 times period and plot (b) presents the results for deadline set to 3 times the period. The average percentage power reduction and standard deviation for $MILP$, $MILP_{so}$, $MILP_{lat}$ and $MILP_{ls}$ are summarized in Tables 21, 22, 23 and 24, respectively for the three applications.

8.5.2.1. *Comparison of MILP with existing techniques.* As can be observed from the first two rows of Table 21 the designs generated by our MILP formulation give large reductions in power consumption in comparison to the existing techniques. In particular the LPS technique is unable to generate designs for JPEG and MPEG-1 with tight performance constraints that require pipelined scheduling. Hence, the non-applicable (NA) entries in first two columns. In comparison to the FP technique our formulation gives an average power reduction of 34.26 % for the three applications.

8.5.2.2. *Comparison of pipelining with integrated pipelining and unrolling.* The last row of Table 21 demonstrates that integration of unrolling transformation with pipelined scheduling during system-level low power design leads to large reductions in power consumption. The average power reduction for the three applications is over 22.81 %. Further as can be observed from the plots in Figures A.1, A.5 and A.9 the unrolling degree required to generate the large power savings is at most one. Large reductions in power consumption coupled with the low unrolling degree establish the viability of integrated pipelining and unrolling transformation during system-level low power design.

8.5.2.3. *Comparison of MILP with $MILP_{so}$, $MILP_{lat}$ and $MILP_{ls}$.* The three integer programming based approximations ($MILP_{so}$, $MILP_{lat}$ and $MILP_{ls}$) give power consumption reductions in comparison to FP as summarized in Tables 22, 23 and 24, respectively. The LPS technique is unable to satisfy the performance constraints for JPEG and MPEG-1. LPS does marginally better than our integer programming based approximations for the MP3 application. The benefits due to unrolling transformation as observed for the optimal formulation also hold for the approximations.

The comparison between the run times of the techniques and their result quality are plotted in Figures 8.10 and 8.11, respectively. In Figure 8.10, the run times are normalized to the run time of the optimal MILP formulation. In Figure 8.11, the individual power consumption of the solution produced by each formulation is normalized with that of $MILP_{ls}$. As expected, the optimal $MILP$ formulation produced best results (minimum power consumption), but with a large run time. The average run time of the formulation was $24.66 sec$. The $MILP_{so}$ and $MILP_{lat}$ formulations produced comparable results with not much difference in their average run times (average run time of $6.3 sec$ and $5.8 sec$, respectively). The $MILP_{ls}$ formulation gave the worst result, but with the smallest run time (average run time of $1.33 sec$). The gradient in power consumption from the optimal to the modified formulations was a moderately increasing

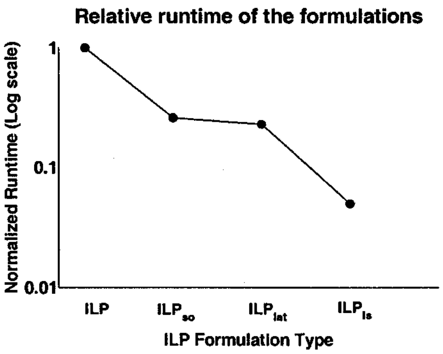


Fig. 8.10: Runtimes of the MILP formulations

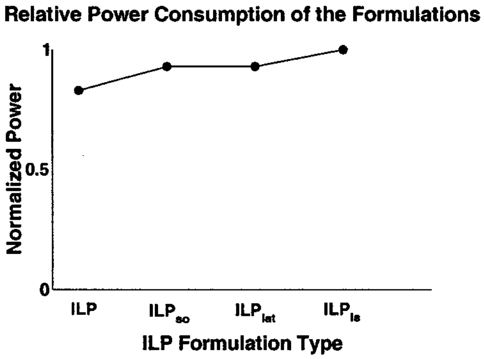


Fig. 8.11: Relative power consumption of the MILP formulations

curve whereas, the corresponding gradient in run times was a steeply decreasing curve. Although $MILP_{so}$ and $MILP_{lat}$ formulations produced similar results with similar run times, the reason for speedup and the quality of result for the two is entirely different. The exponentially lower run times of the various linear programming based approximations provide the designer with desirable techniques for trading design quality for faster turn around times.

8.5.3. Evaluation of heuristic techniques

We evaluated the performance of our deterministic and simulated annealing based heuristic techniques by experimenting with multimedia applications and synthetic task graphs. Similar to the previous experiment we mapped the multimedia applications on dual processor architectures. The synthetic task graphs were designed on both dual and quad processor architectures. In the Appendix Figures A.13, A.14, A.15, A.16 and A.17 plot the results of the experiments performed with our heuristic techniques. We experimented with both single and multiple synthetic task graphs. The multiple task graphs were scheduled over the hyper-period (τ_{hp}) that is given by the lowest common multiple of the periods of the various graphs. Every synthetic task graph G_i was unrolled K times before our techniques are applied, where K is given by $K = \frac{\tau_{hp}}{(\tau_{period}(G_i))}$. Figures

Techniques	% Power Reduction				
	Values	JPEG	MPEG-1	MP3	Synthetic
LPP_{det} v/s FP	**Average**	**28.6**	**33.04**	**27.31**	**46.96**
	Std. dev.	16.93	17.50	17.76	3.19
LPP_{det} v/s LPS	**Average**	**NA**	**NA**	**7.02**	**26.20**
	Std. dev.	NA	NA	4.06	11.97
$LPPWU_{det}$ v/s FP	**Average**	**54.92**	**45.05**	**45.75**	**51.15**
	Std. dev.	6.91	4.55	27.23	2.55
$LPPWU_{det}$ v/s LPS	**Average**	**NA**	**NA**	**19.76**	**31.84**
	Std. dev.	NA	NA	15.55	12.22
$LPPWU_{det}$ v/s LPP_{det}	**Average**	**35.68**	**14.84**	**13.41**	**7.56**
	Std. dev.	5.93	13.82	16.52	7.04

Table 25: Comparison of deterministic technique with other heuristic techniques

Techniques	% Power Reduction				
	Values	JPEG	MPEG-1	MP3	Synthetic
LPP_{sa} v/s FP	**Average**	**36.22**	**35.37**	**27.31**	**47.54**
	Std. dev.	10.33	18.4	17.76	3.63
LPP_{sa} v/s LPS	**Average**	**NA**	**NA**	**7.02**	**26.69**
	Std. dev.	NA	NA	4.06	12.58
$LPPWU_{sa}$ v/s FP	**Average**	**55.45**	**46.50**	**45.75**	**51.98**
	Std. dev.	6.95	5.34	27.23	2.63
$LPPWU_{sa}$ v/s LPS	**Average**	**NA**	**NA**	**19.76**	**32.85**
	Std. dev.	NA	NA	15.55	11.71
$LPPWU_{sa}$ v/s LPP_{sa}	**Average**	**29.78**	**13.91**	**13.41**	**8.17**
	Std. dev.	6.46	13.83	16.52	6.30

Table 26: Comparison of simulated annealing based technique with other heuristic techniques

Techniques	% Deviation from optimum power consumption			
	Values	JPEG	MPEG-1	MP3
LPP_{det} v/s ILP	**Average**	**14.60**	**9.54**	**0.0**
	Std.Dev	14.75	3.16	0.0
LPP_{sa} v/s ILP	**Average**	**2.94**	**5.31**	**0.0**
	Std.Dev	2.88	3.15	0.0

Table 27: Comparison between heuristic and optimum MILP based techniques

	ILP	ILP_{so}	ILP_{lat}	ILP_{ls}	LPP_{det}	LPP_{sa}
Power	1	1.38	1.27	2.18	1.15	1.03
Time	1	0.25	0.23	0.05	0.002	0.09

Table 28: Comparison of all techniques for multimedia applications

A.16 and A.17 plot the normalized reductions in power consumption in comparison to FP.

8.5.3.1. *Comparison with existing techniques.* The first four rows of Tables 25 and 26 summarize the performance of our deterministic and simulated annealing based techniques, respectively in comparison to FP and LPS. As before the "NA" in JPEG and MPEG-1 columns of rows 2 and 4 denote that LPS was unable to generate results for tight performance constraints. Both our heuristic techniques give large reduction in power consumption for multimedia applications and synthetic task graphs in comparison to existing techniques.

The results in the last row of Tables 25 and 26 support the earlier observation that integration of unrolling transformation in system level low power design leads to further reductions in power consumption.

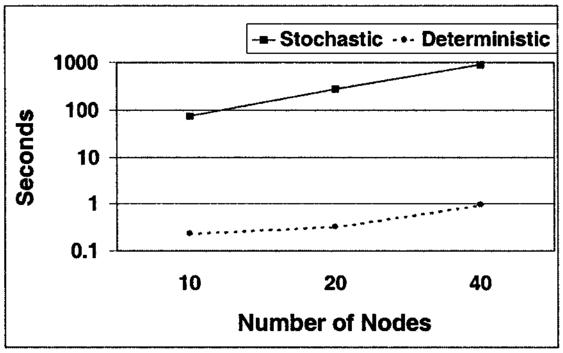


Fig. 8.12: Runtimes of the heuristic techniques

8.5.3.2. *Comparison between heuristic and optimum MILP based techniques.* Table 27 compares our deterministic and simulated annealing based optimization techniques against optimum MILP technique for the multimedia applications. We did not apply our MILP based technique to the synthetic task graphs due to large design turn around times. The simulated annealing technique produces better results than the deterministic strategy. The worst case power consumption of simulated annealing is within 6 % of the optimal.

Table 28 presents the overall summary of all techniques in terms of power and solution time for the multimedia applications. All values in the table are normalized with respect to the corresponding value generated by the optimal MILP technique. Both our heuristic based techniques give better results than the linear programming based approximate solutions. The simulated annealing based technique gives the best trade-off between design quality and solution time.

The average run times of the deterministic and simulated annealing based techniques for large synthetic task graphs are plotted in Figure 8.12. The experiments were conducted on a Sun SPARC 950 MHz dual processor workstation. The run time of the simulated annealing based technique was several times greater than the deterministic technique. Best results with simulated annealing were obtained with an initial temperature $T_{initial} = 1000$, final temperature

$T_{final} = 0.0005$, scaling factor $s = 0.9$, and number of moves per temperature $z = 3N$ where N is the number of tasks in the task graph.

8.6. Conclusion

In this chapter, we addressed the problem of system-level low power design of an embedded multiprocessor architecture executing periodic applications such as multimedia and network traffic processing. We presented an MILP formulation that integrated loop transformation techniques namely, pipelining and unrolling with system-level low power techniques, namely, DVS and DPM to minimize the power consumption of the application, subject to performance constraint. We presented several linearization schemes that can be employed to linearize seemingly non-linear equations in the MILP formulation. Although the MILP is of tremendous value due to its ability to generate optimal solutions, its solution time grows exponentially with the number of inputs. Therefore, we also proposed several techniques to counter this problem. We proposed three MILP based techniques that relax one or more constraints to arrive at the solution at a faster rate. We also presented a heuristic technique that can easily be adapted to perform deterministic or simulated annealing based optimization to solve the same problem in polynomial time.

We performed extensive experimentation with several multimedia applications, as well as large synthetic task graphs. We compared our formulations with existing techniques such as FP and LPS [101]. The integration of pipelining and loop unrolling gave large reductions in power consumption in comparison to existing techniques. All our techniques gave better results than the existing strategies for system level low power design. In particular our simulated annealing based optimization technique gave the best trade-off between result quality and solution generation time.

CHAPTER 9

CONCLUSION

In this thesis, we addressed two important system-level problems of low power high performance GALS based SoC architectures. As the primary contribution, we defined the application specific NoC design problem and proposed several techniques for their design and optimization. As a secondary contribution, we proposed linear programming based, and heuristic techniques for low power mapping and scheduling of tasks on a multiprocessor architecture. We summarize this thesis by discussing our contributions toward the two problems, comparing the relative merits of our proposed techniques, and presenting an insight into future work in this field of research.

9.1. Application specific NoC design

In this section, we present a summary of our NoC design techniques, and discuss open issues and future work. In Section 9.1.1, we compare the results of the different techniques, and in Section 9.1.2, we present some of the assumptions, and future work.

9.1.1. Comparison of the different NoC design techniques

We proposed four techniques for design of custom NoC architectures for application specific SoC. We compared the four techniques with and without port constraints on the target router architecture. For the port constrained case, we assumed a router architecture with a maximum of 5 ports. In each case, we assumed maximum allowable link length of $6mm$. Table 30 summarizes the solution quality and runtimes of the techniques for port constrained as well as the port un-constrained case.

9.1.1.1. *Router architecture without port constraints.* For the port un-constrained case, the low complexity heuristic (ANOC) generated solutions in fraction of a second for all benchmarks. However, the approximation algorithm and GA generated results that were very close to optimal.

	Power			Routers		
	$\frac{ANOC}{ILP}$	$\frac{LP}{ILP}$	$\frac{GA}{ILP}$	$\frac{ANOC}{ILP}$	$\frac{LP}{ILP}$	$\frac{GA}{ILP}$
Port un-constrained	1.18	1.04	1.01	0.94	1.18	1.1
Port constrained	NA	1.22	1.03	NA	1.12	1.12

Table 29: Average power and router consumption

Runtime (secs)				
Graph size	ILP	ANOC	LP	GA
5	~ 5	< 1	< 1	~ 5
15	several hours	< 1	~ 1	~ 100
25	several hours	< 1	~ 5	~ 1000

Table 30: Runtimes

The low router consumption of ANOC compared to other techniques is due to the merging step in the algorithm that performs local optimization, and merges routers that are close to each other. On the other hand, the approximation algorithm and the GA optimize for power, and do not merge routers if it results in a higher power consumption.

9.1.1.2. *Port constrained router architecture.* ANOC does not address port constraints. As a result, we do not compare ANOC with other proposed techniques. On the other hand, the other three techniques address port constraints. On average, the results generated by the GA came closest to the optimal, followed by the approximation algorithm. Under port constraints, the approximation algorithm employs heuristics to arrive at valid solutions. This explains its slightly poorer performance compared to the case when no port constraints exist.

9.1.2. Discussion and future directions

Our techniques synthesize an application specific NoC with an objective of minimizing the communication power subject to the performance constraints. The techniques account for both the physical link and router power consumption. The techniques incorporate a system-level floorplanning stage to calculate the link power accurately. We demonstrated that the custom topologies generated by our techniques perform better than both traditional mesh based and QNOC based NoC architectures in terms of power and resource consumption.

The discussion on router characterization assumed a particular router architecture with data width of 32, 4 virtual channels at every port (2 for input and 2 for output), depth of virtual channels at 16, a packet size of 256 bits, and that performed wormhole switching. However, our techniques are not limited to the particular router architecture. They are applicable for any router architecture in which the input and output port power varies linearly with bandwidth. We expect that these characteristics are applicable to all other router architectures, and therefore so are our techniques.

The technique requires that the router architecture be characterized for its power consumption. During characterization the router architecture has a certain number of virtual channels (two in our case). The final number of virtual channels might vary due to the possibility of deadlocks. We found that for our router architectures with 4 virtual channels, deadlock did not occur in any design. Further, even for generic NoC architectures increasing the number of virtual channels results in diminishing performance improvements. Therefore, the router architecture can be characterized with a fixed number of virtual channels, and the deviation from the architecture is expected to be minimal.

Although our objective functions are aimed at dynamic power minimization, we do address router utilization. We reduce the number of available routers at floorplanning stage by eliminating redundant routers. At the interconnection architecture design stage we reduce the number of hops for each traffic trace and thereby utilizing minimum number of routers.

The NoC is designed after the computation architecture has been finalized. Consequently, even before the NoC is generated the bandwidth production and consumption requirements of the various computation architecture cores are known. Therefore, we assume that cores can produce and consume the required bandwidth. The only requirement that we place on the NoC is that the unit router input and output port should be able to consume and produce the required core bandwidth, respectively. In the case that the computation architecture has a wide variation in the bandwidth requirements, the designer can choose to develop a NoC architecture with either average or worst case communication traffic by specifying respective communication trace graphs.

The formulation for floorplanning presented in Section 4.2.1 of Chapter 4 does not address aspect ratios and orientations of individual cores. At the floorplanning stage the contribution of the thesis is in terms of a unique cost function that addresses NoC specific constraints, and extensions for addressing mesh based layouts. Please refer to [69] for more complete floorplanning formulations. All of these formulations can be combined with the cost function proposed in the thesis to generate NoC specific layouts.

Technology scaling will result in an increase in the static power consumption due to the routers and dynamic power consumption due to the physical links. Our techniques account for the contribution of physical links toward the total power consumption. Static power consumption can be minimized by two techniques. First, router ports that do not support any traffic traces can be eliminated from the design. Thus, the final topology would consist of heterogeneous router architectures. Virtual channels are chief contributors of leakage power in a router architecture [40]. Their contribution can be reduced by over 80 % by deploying run time power management schemes such as those proposed by Chen et al. [40].

In nanoscale technologies, architectures are expected to be inherently faulty. The traditional techniques of introducing redundancy to improve fault tolerance will prove expensive in terms of leakage power as well as area. Novel techniques for design of fault tolerant NoC is an open problem. Our techniques do not currently address fault tolerant topologies. Automated design techniques for fault tolerant topologies that support static routing would be focus of future work. As we generate custom topologies, adaptive routing techniques that can be readily applied to regular topologies cannot be easily integrated into our architectures.

Our thesis addresses the NoC design in isolation. Integration of computation architecture design with NoC synthesis has the potential for larger power savings.

9.2. System-level power minimization

In this section, we summarize our contributions in the field of system-level power minimization, and present an overview of future work.

The increased power consumption is leading to higher die temperatures which contribute to an increase in leakage current and power. This cascading effect can result in a condition known as thermal run-off that eventually causes chip burn-out. Therefore, novel design techniques are required for thermal aware design. We aim to address this problem both at the MPSoC architecture design stage and at application development. At the design time, novel thermal aware floorplanning based techniques can be utilized to influence the MPSoC core selection. Similarly, at the application design time thermal aware application to core mapping can mitigate the problem.

In this thesis, we propose optimal MILP based formulations and heuristics for multiprocessor scheduling and mapping. There has been very little work done on provably good algorithms for low power multiprocessor scheduling and mapping, under performance constraints. We plan to develop good algorithms for the scheduling and mapping problem.

المنارة للاستشارات

Appendix A

...

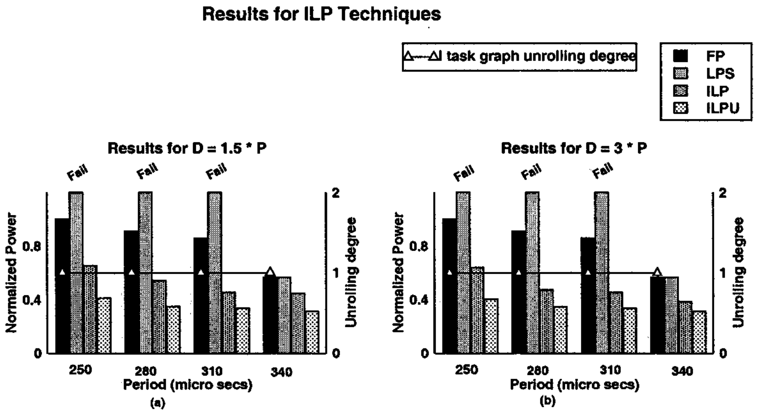


Fig. A.1: JPEG: Results for $MILP$

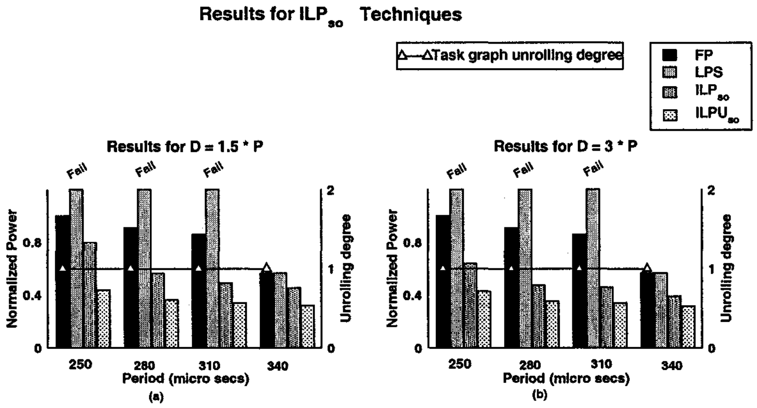


Fig. A.2: JPEG: Results for $MILP_{so}$

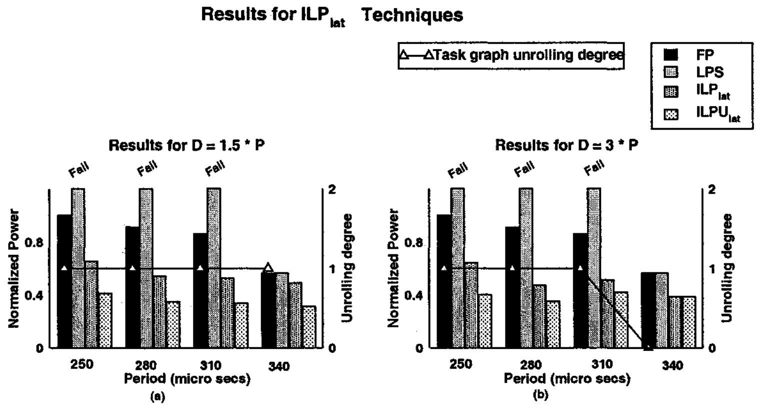


Fig. A.3: JPEG: Results for the $MILP_{lat}$

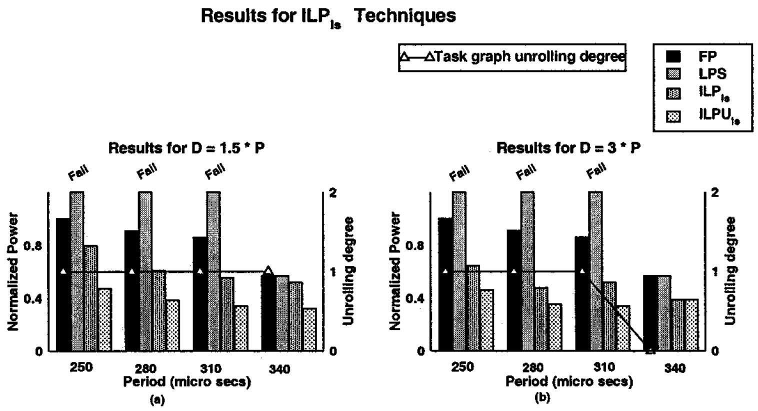


Fig. A.4: JPEG: Results for $MILP_{ls}$

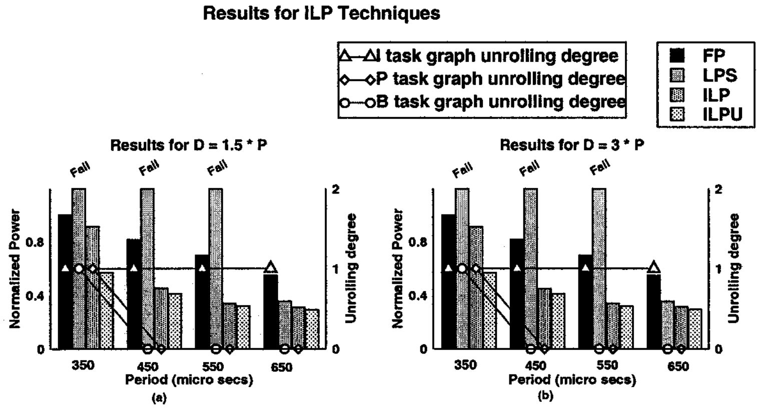


Fig. A.5: MPEG: Results for $MILP$

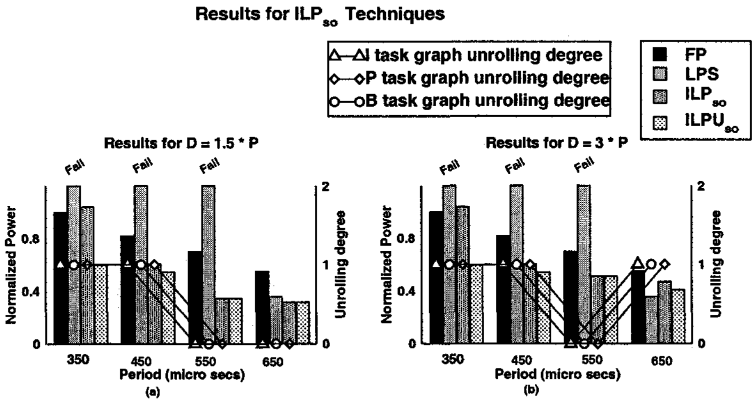


Fig. A.6: MPEG: Results for the $MILP_{so}$

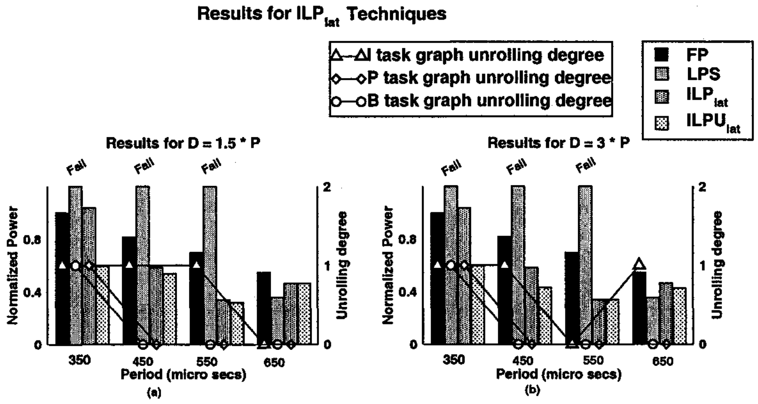


Fig. A.7: MPEG: Results for $MILP_{lat}$

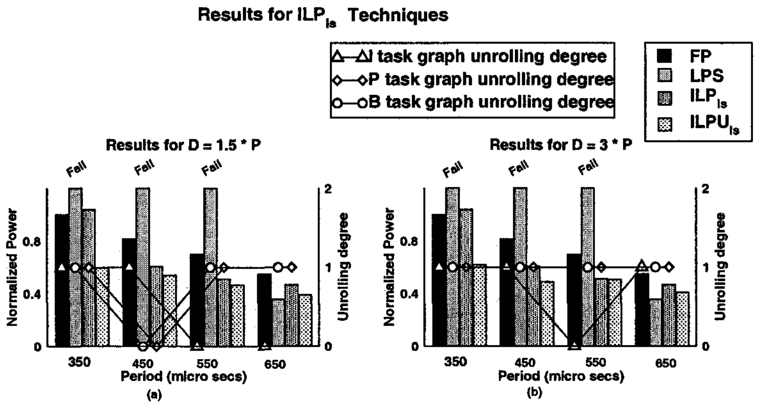


Fig. A.8: MPEG: Results for $MILP_{ls}$

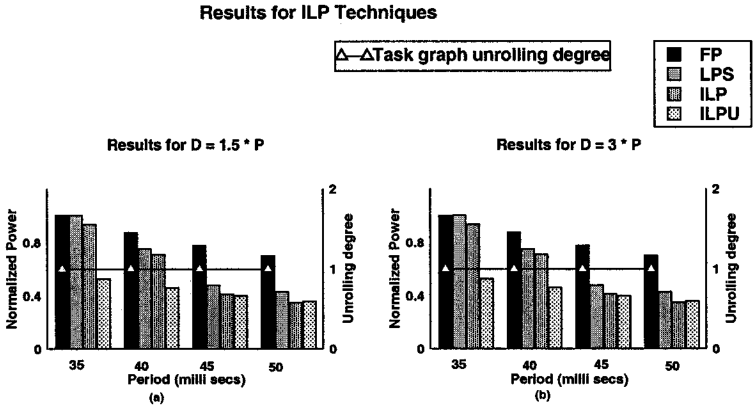


Fig. A.9: MP3: Results for $MILP$

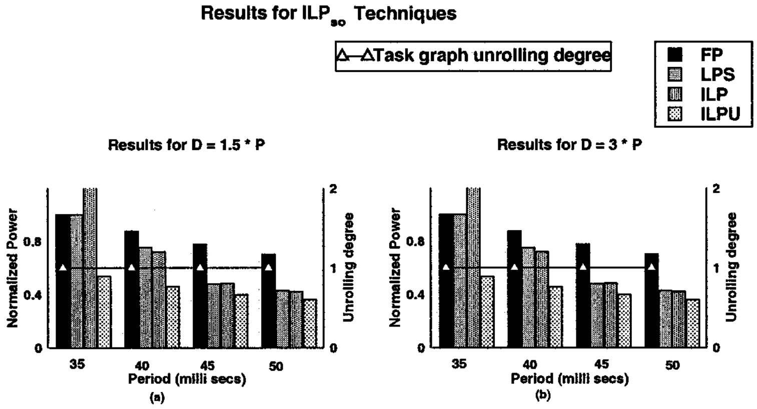


Fig. A.10: MP3: Results for the $MILP_{so}$

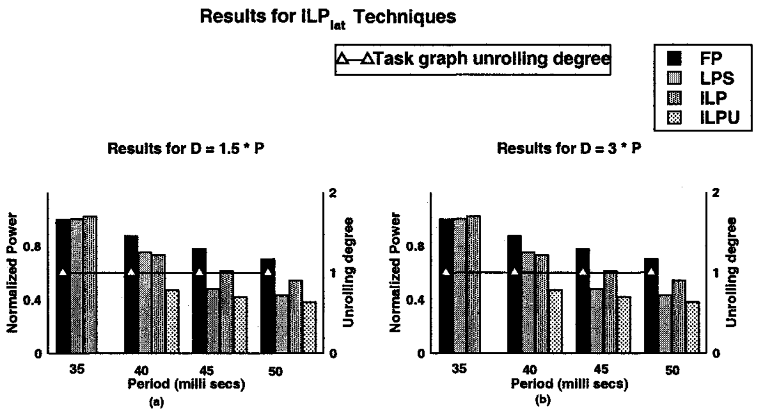


Fig. A.11: MP3: Results for $MILP_{lat}$

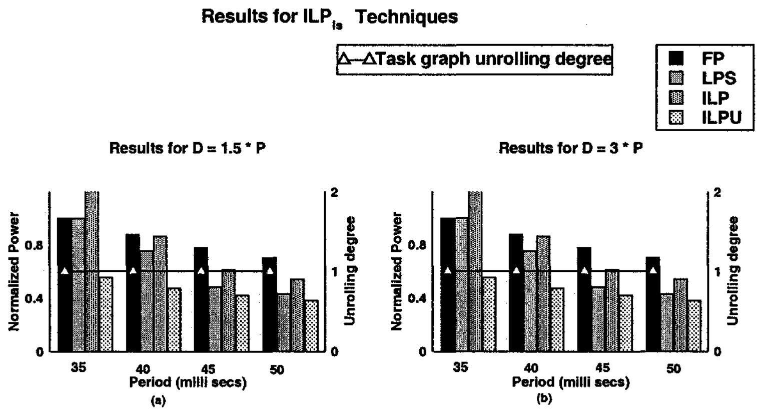
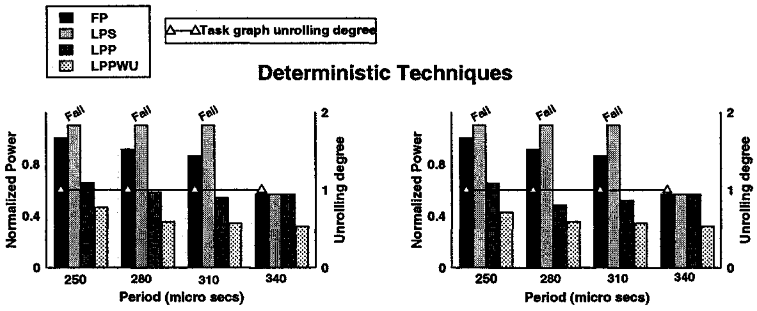


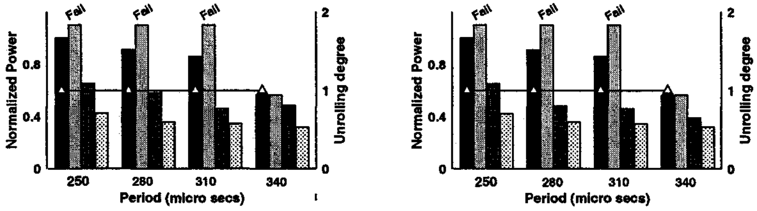
Fig. A.12: MP3: Results for $MILP_{ls}$



Deterministic Techniques

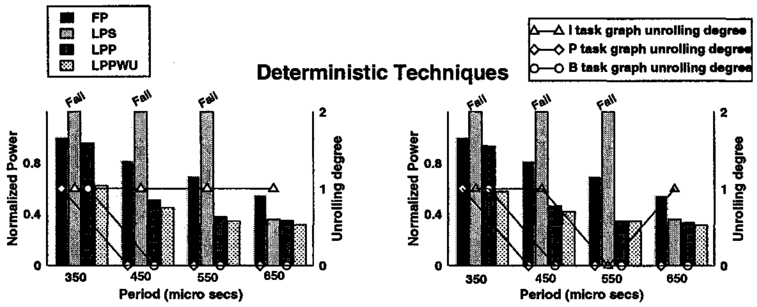
(a) Results for D = 1.5 P (b) Results for D = 3 P

Stochastic Techniques



(c) Results for D = 1.5 P (d) Results for D = 3 P

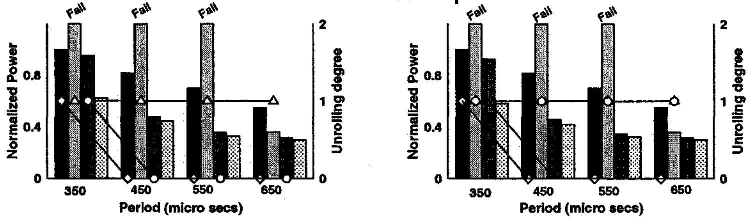
Fig. A.13: Heuristics: Results for JPEG decoding



Deterministic Techniques

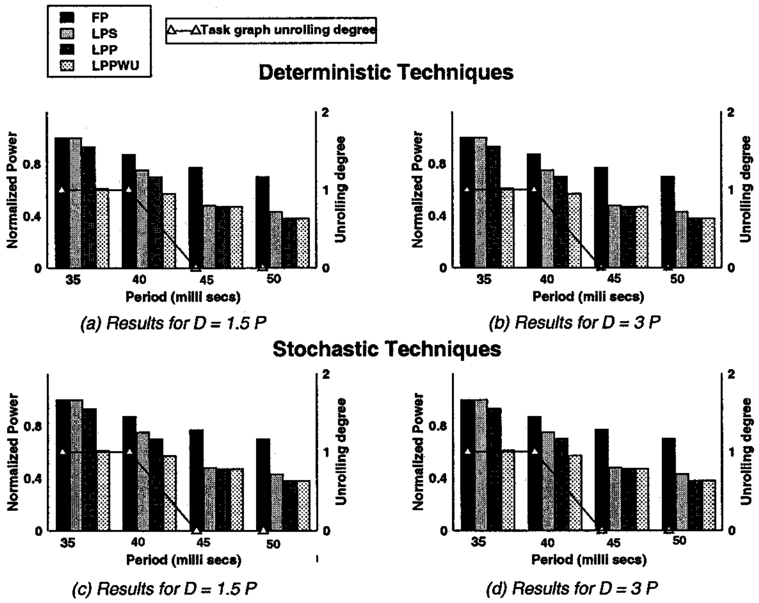
(a) Results for D = 1.5 P (b) Results for D = 3 P

Stochastic Techniques



(c) Results for D = 1.5 P (d) Results for D = 3 P

Fig. A.14: Heuristics: Results for MPEG-1 decoding



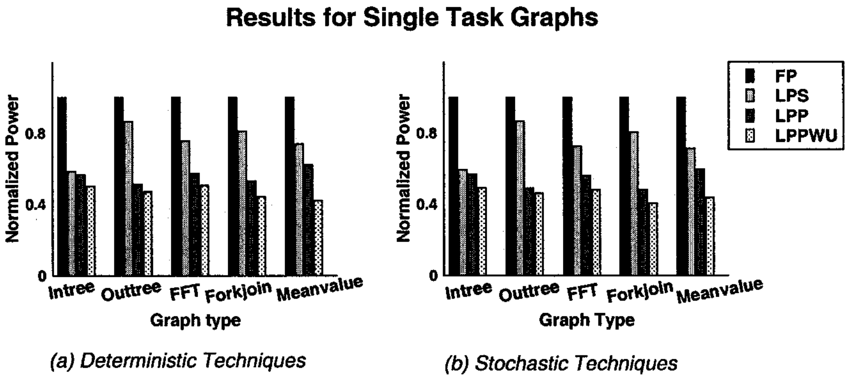
(a) Results for D = 1.5 P

(b) Results for D = 3 P

(c) Results for D = 1.5 P

(d) Results for D = 3 P

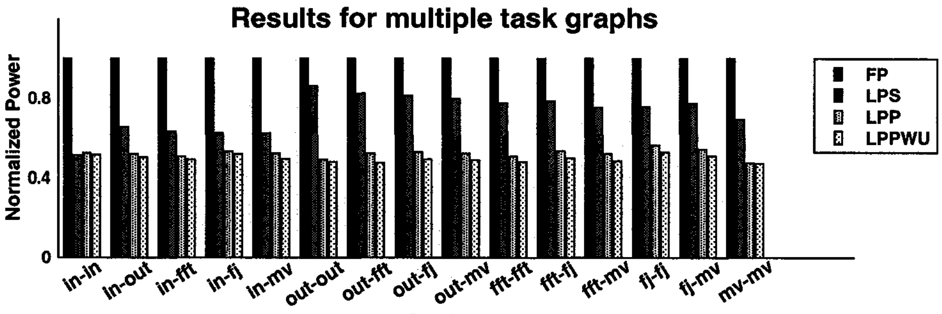
Fig. A.15: Heuristics: Results for MP3 encoding



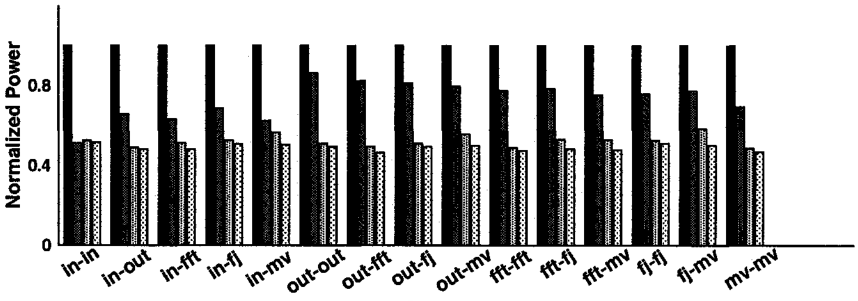
(a) Deterministic Techniques

(b) Stochastic Techniques

Fig. A.16: Single synthetic task graphs



(a)Deterministic Techniques



(b)Stochastic Techniques

Fig. A.17: Multiple synthetic task graphs

REFERENCES

[1] K. Srinivasan, and K. S. Chatha, "An ILP Formulation for System-level Power and Performance Optimization in Multiprocessor SoC Architectures," in *Proceedings of VLSI Design*, (Mumbai, India), January 2004.

[2] K. Srinivasan, N. Telkar, V. Ramamurthi, and K. S. Chatha, "System-level Design Techniques for Power and Performance Optimization of Multiprocessor SoC Architectures," in *Proceedings of ISVLSI*, (Lafayette, Lousiana, USA), January 2004.

[3] D. M. Chapiro, *Globally Asynchronous Locally Synchronous Systems*. PhD thesis, Stanford University, 1984.

[4] D. Sylvester and K. Keutzer, "A Global Wiring Paradign for Deep Submicron Design," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, pp. 242–252, February 2000.

[5] R.Ho, K. Mai and M. Horowitz, "The Future of Wires," *Proceedings of IEEE*, pp. 490–504, April 2001.

[6] W. J. Dally and B. Towles, "Route packet, not wires: On-chip interconnection networks," in *Proceedings of DAC*, June 2002.

[7] L. Benini and G. D. Micheli, "Networks on chips: A new soc paradigm," *IEEE Computer*, pp. 70–78, January 2002.

[8] R. Dick and N. Jha, "MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Cosynthesis of Distributed Embedded Systems," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 17, October 1998.

[9] B.P. Dave, G. Lakshimnarayana and N.K. Jha, "COSYN: Hardware-Software Cosynthesis for Heterogeneous Distributed Embedded Systems," *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, vol. 7, March 1999.

[10] S. Prakash and A. Parker, "SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems," *Journal of Parallel and Distributed Computing*, vol. 16, pp. 338–351, 1992.

[11] M. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. S. M. Frank, S. Amarasinghe, and A. Agrawal, "The RAW Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs," *IEEE Micro*, pp. 25–35, March-April 2002.

[12] K. Srinivasan, K. Chatha, and G. Konjevod, "Linear Programming based Techniques for Synthesis of Network-on-Chip Architectures," *IEEE Transactions on VLSI*, vol. 14, no. 4, pp. 407–420, 2006.

[13] K. Srinivasan, K. S. Chatha, and G. Konjevod, "Linear programming based techniques for synthesis of network-on-chip architectures," in *Proceedings of ICCD*, (San Jose, USA), October 2004.

[14] K. Srinivasan, and K. S. Chatha, "A Methodology for Design and Optimization of Network-on-Chip Architectures," in *Proceedings of ISQED*, (San Jose), March 2006.

[15] K. Srinivasan, and K. S. Chatha, "A Low Complexity Heuristic for Design of Custom Network-on-Chip Architectures," in *Proceedings of DATE*, (Munich, Germany), March 2006.

[16] K. Srinivasan, and K. S. Chatha, "A Technique for Low Energy Mapping and Routing in Network-on-Chip Architectures," in *Proceedings of ISLPED*, (San Diego, CA, USA), August 2005.

[17] K. Srinivasan, and K. S. Chatha, "Layout Aware Design of Mesh based Network-on-Chip Architectures," in *Accepted at Proceedings of ISSS-CODES*, 2006.

[18] K. Srinivasan, K. S. Chatha, and G. Konjevod , "An Automated Technique for Topology and Route Generation for Application Specific On-Chip Interconnection Networks," in *Proceedings of ICCAD*, (San Jose, USA), October 2005.

[19] K. Srinivasan, K. S. Chatha, and Goran Konjevod, "Application Specific Network-on-Chip Design with Guaranteed Quality Approximation Algorithm," in *Accepted at Proceedings of ASPDAC*, (Japan), January 2007.

[20] K. Srinivasan, and K. S. Chatha, "ISIS: A Genetic Algorithm based Technique for Synthesis of On-Chip Interconnection Networks," in *Proceedings of VLSI Design*, (Calcutta, India), January 2005.

[21] K. Srinivasan, and K. S. Chatha, "SAGA: Synthesis Technique for Guaranteed Throughput NoC Architectures," in *Proceedings of ASPDAC*, (Shanghai, China), January 2005.

[22] N. Banerjee, P. Vellanki, and K. S. Chatha , "A Power and Performance Model for Network-on-Chip Architectures," in *Proceedings of DATE*, (Paris, France), February 2004.

[23] P.Guerrier and A.Greiner, "A Generic Architecture for On-Chip Packet-Switched Interconnections," in *Proceedings of DATE*, (Paris, France), March 2000.

المنارة للاستشارات

[24] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, and M. Millberg, "Network on Chip: An architecture for billion transistor era," in *Proceedings of IEEE NorChip Conference*, November 2000.

[25] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabeay and A. Sangiovanni-Vincentelli, "Addressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design," in *Proceedings of Design Automation Conference*, pp. 667–672, June 2001.

[26] S. Kumar, A. Jantsch, M. Millberg, J. Oberg, J. Soininen, M. Forsell, K. Tiensyrj A. Hemani, "A Network on Chip Architecture and Design Methodology," in *IEEE Computer Society Annual Symposium on VLSI*, (Pittsburg, Pennsylvania), April 2002.

[27] A. Andriahantenaina and A. Greiner, "Micro-network for SoC: Implementation of a 32-port SPIN network," in *Proceedings of DATE*, (Munich, Germany), March 2003.

[28] A. Andriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino, "SPIN: a Scalable, Packet Switched, On-Chip Micro-network," in *Proceedings of DATE*, (Munich, Germany), March 2003.

[29] D. Siguenza-Tortosa and J. Nurmi, "Proteo: A New Approach to Network-on-Chip," in *Proceedings of IASTED International Conference on Communication Systems and Network*, (Malaga, Spain), 2002.

[30] D. Siguenza-Tortosa and J. Nurmi, "VHDL-based simulation environment for Proteo NoC," in *High-Level Design Validation and Test Workshop*, (Paris, France), October 2002.

[31] M. Dall'Osso, G. Biccari, L. Giovanninni, D. Bertozzi, and L. Benini, "xpipes: a Latency Insensitive Parameterized Network-on-Chip Architecture for Multi-Processor SoCs," in *Proceedings of ICCD*, (San Jose, CA), October 2003.

[32] A. Jalabert, S. Murali, L. Benini, and G. De Micheli, "xpipesCompiler: A tool for instantiating application specific Networks on Chip," in *Proceedings of DATE*, 2004.

[33] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The Nostrum backbone - a communication protocol stack for networks on chip," in *VLSI Design*, (Mumbai, India), January 2004.

[34] M. Millberg, E. Nilsson, Rikard Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip," in *Proceedings of DATE*, pp. 890–895, February 2004.

[35] J. Dielissen, A. Rădulescu, K. Goossens, and E. Rijpkema, "Concepts and implementation of the Philips network-on-chip," in *Proceedings of IP-Based SOC Design*, Nov. 2003.

[36] E. Rijpkema, K. G. W. Goossens, and A. Radulescu, "Trade Offs in the Design of a Router with Both Guaranteed Best-Effort Services for Networks on chip," in *Proceedings of DATE*, 2004.

[37] D. Bertozzi, L. Benini, and G. De Micheli, "Low power error resilient encoding for on-chip data buses," in *Proceedings of DATE*, 2003.

[38] H. Zimmer and A. Jantsch, "A Fault Model Notation and Error-Control Scheme for switch-to-Switch Buses in a Network-on-Chip," in *Proceedings of ISSS/CODES*, 2003.

[39] F. Worm, P. Ienne, P. Thiran, G. De Micheli, "An Adaptive Low-Power Transmission Scheme for On-Chip Networks," in *Proceedings of ISSS*, (Kyoto, Japan), 2002.

[40] X. Chen and L-S Peh, "Leakage Power Modeling and Optimization in Interconnection Networks," in *Proceedings of ISLPED*, (Seoul, Korea), 2003.

[41] T. Simunic and S. Boyd, "Managing Power Consumption in Networks on Chips," in *Proceedings of DATE*, (Paris, France), 2002.

[42] E. Nilsson and J. Oberg, "Reducing Power and Latency in 2-D Mesh NoC using Globally Pseudochronous and Locally Synchronous Clocking," in *Proceedings of ISSS-CODES*, 2004.

[43] J. Duato, S. Yalamanchili, L. Ni , *Interconnection Networks, an Engineering Approach.* IEEE Computer Society, 1997.

[44] H. Seigel, "A model of SIMD machines and a comparison of various interconnection networks," *IEEE Transactions on Computers*, vol. 28, pp. 907–917, December 1979.

[45] W. Dally, "Performance analysis of k-ary n-cube interconnection network," *IEEE Transactions on Computers*, vol. 39, pp. 775–785, June 1990.

[46] J. Draper and J. Ghosh, "A Comprehensive Analytical Model for Wormhole Routing in Multicomputer Systems," *Journal of Parallel and Distributed Computing*, vol. 23, pp. 202–214, 1994.

[47] A.G. Wassal and M.A. Hasan, "Low-power system-level design of VLSI packet switching fabrics," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 20, pp. 723–738, June 2001.

[48] T. T. Ye, L. Benini and G. De Micheli, "Analysis of Power Consumption on Switch Fabrics in Network Routers," in *Proceedings of DAC*, 2002.

المنارة للاستشارات

[49] D. Pamunuwa, J. Oberg, L. R. Zheng, M. Millberg, A. Jantsch, and H. Tenhunen, "Layout, performance and power trade-offs in mesh-based network-on-chip architectures," in *Proceedings of IFIP International Conference on Very Large Scale Integration(VLSI-SOC), Darmstadt, Germany*, pp. 362–367, December 2003.

[50] H-S Wang, L-S Peh and S. Malik, "Orion: A Power-Performance Simulator for Interconnection Network," in *Proceedings of International Symposium on Microarchitecture*, (Istanbul, Turkey), November 2002.

[51] E. Bolotin, I. Cidon, R. Ginosar and A. Kolodny, "Cost considerations in Network on Chip," in *Integration - the VLSI journal*, November 2003.

[52] J. Hu and R. Marculescu, "Energy-Aware Mapping for Tile-based NoC Architectures Under Performance Constraints," in *Proceedings of ASP-DAC*, 2003.

[53] S. Murali, and G. De Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," in *Proceedings of DATE*, 2004.

[54] G. Ascia, V. Catania, and M. Palesi, "Multi-objective Mapping for Mesh-based NoC Architectures," in *Proceedings of ISSS-CODES*, 2004.

[55] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, A. Kolodny, "Efficient link capacity and qos design for network-on-chip," in *Proceedings of DATE*, March 2006.

[56] S. Murali, M. Coenen, A. Radulescu, K. Goossens, G. DeMicheli, "A methodology for mapping multiple use cases on networks on chips," in *Proceedings of DATE*, March 2006.

[57] F. Angiolini, P. Meloni, S. Carta, L. Benini, L. Raffo, "Contrasting noc and a traditional interconnect fabric with layout awareness," in *Proceedings of DATE*, March 2006.

[58] Steenhof, "Network-on-chip for high-end consumer electronics tv system architectures," in *Proceedings of DATE*, March 2006.

[59] L. Benini, "Application specific network-on-chip," in *Proceedings of DATE*, March 2006.

[60] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli, "Efficient synthesis of networks on chip," in *ICCD*, 2003.

[61] U. Ogras and R. Marculescu, "Energy and Performance Driven NoC Communication Architecture Synthesis Using A Decomposition Approach," in *Proceedings of DATE*, 2005.

[62] U. Ogras and R. Marculescu, "Application Specific Network-on-Chip Architecture Customization Via Long Range Link Insertion," in *Proceedings of ICCAD*, 2005.

المنارة للاستشارات

[63] D'andrade R, "U-Statistic Hierarchical Clustering," *Psychometrica*, vol. 4, no. 58-67, 1978.

[64] "Hierarchical Clustering, http://www.analytictech.com/networks/hiclus.htm,"

[65] W.J. Dally and C.L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547–553, 1987.

[66] "Dash Optimization, http://www.analytictech.com/networks/hiclus.htm,"

[67] S. Warnakulasuriya and T. M. Pinkston, "Characterization of deadlocks in irregular networks," *Journal of Parallel and Distributed Systems*, vol. 62, no. 1, pp. 61–84, 2002.

[68] N. Thepayasuwan, and A. Doboli, "Layout Conscious Approach and Bus Architecture Synthesis for Hardware-Software Co-design of Systems on Chip Optimized for Speed," *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, vol. 13, pp. 525–538, 2005.

[69] S. Sait and H. Youssef, *VLSI Physical Design Automation: Theory and Practice*. McGraw-Hill Inc, 1994.

[70] S.N. Adya, and I.L. Markov, "Fixed Outline Floorplanning: Enabling Hierarchical Design," *IEEE Transactions on VLSI Systems*, vol. 11, pp. 1120–1135, December 2003.

[71] T.E. Cormen, C.E. Leiserson and R.L. Rivest, *Introduction To Algorithms*. McGraw-Hill, 1989.

[72] K. Jain , "A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem ," *Combinatorica*, vol. 1, pp. 39–60, 2001.

[73] Pulleyblank , "Polyhedral Combinatorics ," *Handbooks on OR and MS*, vol. 1, pp. 371–446.

[74] K. Banerjee and A. Mehrotra, "Inductance Aware Interconnect Scaling," in *ISQED*, 2002.

[75] "JPEG Hardware Compressor, http://www.analytictech.com/networks/hiclus.htm,"

[76] V. Ramamurthi, Jason McCollum, Christopher Ostler, and K. S. Chatha, "System-level Methodology for Programming CMP based Multi-threaded Network Processor Architectures," in *Proceedings of ISVLSI*, (Tampa, Florida, USA), May 2005.

[77] S. Pasricha, N. Dutt, E. Bozorgzadeh, and M. Ben-Romdhane, "FABSYN: Floorplan-Aware Bus Architecture Synthesis," *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, vol. 14, pp. 241–253, 2006.

المنارة للاستشارات

[78] D. Kim, K. Chung, C. Yu, C. Kim, I. Lee, J. Bae, J. Park, S. Kim, Y. Park, N. Seong, J. Lee, J. Park, S. Oh, S. Jeong, L. Kim, "An SoC With 1.3 Gtexels/s 3-D Graphics Full Pipeline for Consumer Applications," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 71–84, 2006.

[79] M. Alho. and J. Nurmi, "Implementation of Interface Router IP for Proteo Network-on-Chip," in *Proceedings of International Workshop on Design and Diagnostics of Electronic Circuits and Systems*, (Poznan, Poland), April 2003.

[80] Intel Corporation, "Ixp1200 network processor datasheet," 2001.

[81] Texas Instruments, "Tms320c8x system-level synopsis," 1995.

[82] Motorola, "C-5 network processor data sheet," 2002.

[83] Sun Microsystems, "Sx2500 board." http://www.sun.com/products-n-solutions/boards/sx2500/.

[84] Alacron Inc, "Alacron fastimage 1500." http://www.alacron.com/.

[85] Synergy Inc, "Synergy mantaqx." http://www.synergymicro.com/.

[86] L. Benini and G. De Micheli, "A survey of design techniques for sytem-level dynamic power management," 2000.

[87] N. K. Jha, "Low Power System Scheduling and Synthesis," in *Proceedings of ICCAD*, (San Jose, CA), November 2001.

[88] "Intel Corporation, Intel StrongARM SA-1110 Microprocessor Developer's Manual," 2001.

[89] J. M. Rabaey, *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, 1996.

[90] J. Hennessy and D. Patterson, *Computer Architecture, A Quantitative Approach*. Morgan Kaufmann Publishers, Inc.

[91] W. Kwon, and T. Kim, "Optimal Voltage Allocation Techniques for Dynamically Variable Voltage Processors," in *Proceedings of DAC*, (Anaheim, CA), June 2003.

[92] G. Quan and S. Hu, "Energy Efficient Fixed Priority Scheduling for Real-Time Systems on Variable Voltage Processors," in *Proceedings of DAC*, (Las Vegas, Nevada), June 2001.

[93] I. Hong, M. Potkonjak and M. B. Srivastava, "Energy Efficient Fixed Priority Scheduling for Real-Time Systems on Variable Voltage Processors," in *Proceedings of ICCAD*, (San Jose, CA), November 1998.

[94] Y. Shin, and K. Choi, "Energy Efficient Fixed Priority Scheduling for Real-Time Systems on Variable Voltage Processors," in *Proceedings of DAC*, (New Orleans, Louisiana), June 1999.

[95] K. Flautner, S. Reinhardt, and T. Mudge, "Automatic Performance Setting for Dynamic Voltage," in *Proceedings of MOBICOM*, (Rome, Italy), July 2001.

[96] R. Rao and S. Vrudhula, "Energy Optimal Speed Control of a Generic Device," *Accepted for IEEE Transactions on CAD.*

[97] R. Rao and S. Vrudhula, "Energy Optimal Speed Control of Devices with Discrete Speed Sets," in *Proceedings of DAC*, (Anaheim, CA), June 2005.

[98] R. Rao and S. Vrudhula, "Energy Optimization for a two-device Data Flow Chain," in *Proceedings of ICCAD*, (San Jose, CA), November 2004.

[99] J. Zhuo and C. Chakrabarti, "System-level Energy Efficient Dynamic Task Scheduling," in *Proceedings of DAC*, 2005.

[100] Y. Zhang, X. Hu, and D. Chen, "Task Scheduling and Voltage Selection for Energy Minimization," in *Proceedings of DAC*, 2002.

[101] J. Luo and N.K. Jha, "Power Conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems," in *Proceedings of ICCAD*, 2000.

[102] F. Gruian and K.Kuchcinski, "Energy Efficient Fixed Priority Scheduling for Real-Time Systems on Variable Voltage Processors," in *Proceedings of ASP-DAC*, (Yokohama, Japan), January 2001.

[103] J. Liu, P. Chou, N. Bagherzadeh, F. Kurdahi, "Power Aware Scheduling Under Timing Constraints for Mission Critical Embedded Systems," in *Proceedings of DAC*, 2001.

[104] N.K. Bhambha, S. Bhattacharya, J. Teich, E. Zitzler, "Hybrid Global/Local Search Strategies for Dynamic Voltage Scaling in Embedded Microprocessors," in *Proceedings of International Workshop on Hw/SW Codesign*, 2001.

[105] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, R. Lauwereins , "Energy Aware Runtime Scheduling for Embedded Multiprocessor SoCs ," *IEEE Design and Test of Computers*, vol. 18, no. 5, pp. 46–58, 2001.

[106] A.Manzak and C. Chakrabarti , "A Low Power Scheduling Scheme With Resources Operating at Multiple Voltages ," *IEEE transactions on VLSI systems*, vol. 10, no. 1, February 2002.

[107] Y.-H. Lin, C.-T. Hwang, and A. C.-H. Wu, "Scheduling Techniques for Variable Voltage Low Power Designs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 2, no. 2, pp. 81–97, 1997.

[108] M. C. Johnson and K. Roy, "Datapath Scheduling with Multiple Supply Voltages and Level Converters," *ACM Transactions on Design Automation of Electronic Systems*, vol. 2, no. 3, pp. 227–248, 1997.

[109] S. P. Mohanty, N. Ranganathan, and S. K. Chappidi, "ILP Models for Energy and Transient Power Minimization during Behavioral Synthesis," in *Proceedings of 17th International Conference on VLSI Design*, pp. 745–748, 2004.

[110] S. P. Mothanty and N. Ranganathan, "A Framework for Energy and Transient Power Reduction during Behavioral Synthesis," *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, vol. 12, pp. 562–572, June 2004.

[111] W. Shiue and C. Chakrabarti , "Memory design and exploration for low power, embedded systems," in *Proceedings of SIPS*, pp. 281–290, 1999.

[112] G. Esakkimuthu, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, "Memory design and exploration for low power, embedded systems," in *Proceedings of ISLPED*, pp. 244–246, 2000.

[113] G. DeMicheli , *Synthesis and Optimization of Digital Circuits.* Mgraw-Hill International Editions, 1994.

[114] N. Metropolis et al., "Equation of State Calculation by Fast Computing Machines," *Journal of Chemistry and Physics*, pp. 1087–1092, 1953.

[115] A. Sinha, and A.P Chandrakasan, "Memory design and exploration for low power, embedded systems," in *Proceedings of DAC*, June 2001.

[116] V.Raghunathan, C. Schurgers, S. Park, M. Srivastava, "Energy Aware Wireless Microsensor Networks," *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 40–50, March, 2002.

[117] S.Wuytack, J.L. da Silva, F. Catthoor, G. Jong, C. Couvreur, "Memory Management for Embedded Network Applications," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems.*

المنارة للاستشارات

[118] J. Czyzyk, M. Mesnier and J. More , "The NEOS Server ," *IEEE Journal on Computational Science and Engineering*, pp. 68–75, 1999.

[119] M. Ferris, M. Mesnier and J. More , "NEOS and Condor:Solving Optimization Problems over the Internet ," tech. rep.