

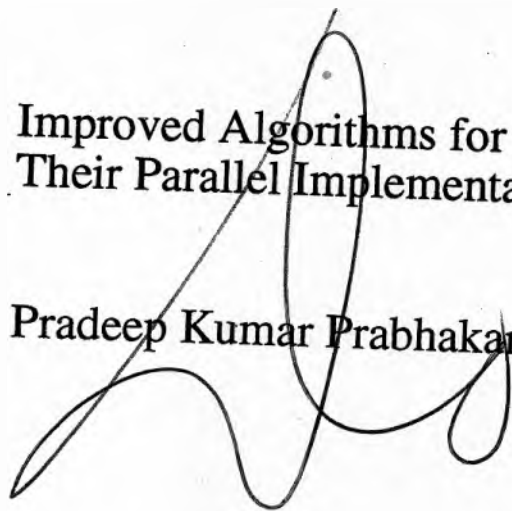
September 1998

UILU-ENG-98-2223
CRHC-98-12

University of Illinois at Urbana-Champaign

Improved Algorithms for High-Level Synthesis and
Their Parallel Implementations

Pradeep Kumar Prabhakaran



Coordinated Science Laboratory
1308 West Main Street, Urbana, IL 61801



REPORT DOCUMENTATION PAGE			Form Approved OMB NO. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comment regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 1998		3. REPORT TYPE AND DATES COVERED
4. TITLE AND SUBTITLE Improved Algorithms for High-Level Synthesis and Their Parallel Implementations			5. FUNDING NUMBERS	
6. AUTHOR(S) Pradeep Kumar Prabhakaran				
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES) Coordinated Science Laboratory University of Illinois 1308 W. Main Urbana, IL 61801			8. PERFORMING ORGANIZATION REPORT NUMBER UIIU-ENG-98-2223 (CRHC-98-12)	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12 b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) Scheduling, allocation and binding are the three major steps in high-level synthesis. In the first part of this thesis, a novel extension of the force directed scheduling algorithm which naturally handles loops and conditionals is presented. Three new parallel algorithms for the scheduling problem are presented. The parallel algorithms for force directed scheduling presented in this thesis produce exactly the same results as the sequential algorithms on which they are based. With small device features in sub-micron technologies, since interconnection delays play a dominant part in the cycle time and the interconnect power dissipation is a major component of the energy consumption, it is important to consider the impact of physical design during high-level synthesis. In the second part of this thesis, a simultaneous scheduling, binding and floorplanning algorithm is presented. In comparison to a traditional approach which separates high-level synthesis from physical design, our algorithm is able to make these stages interact very closely, resulting in solutions with better quality. A simultaneous scheduling, binding and floorplanning algorithm for minimizing the interconnect energy is also presented which is able to reduce the interconnect energy by up to around 67 percent in comparison to a latency-driven approach. In the third part of the thesis, two parallel algorithms for simultaneous scheduling, binding and floorplanning algorithm are presented. Experimental results are reported on an IBM SP-2 multicomputer, with close to linear speedups for a set of benchmark circuits, while giving solutions of almost the same or better quality.				
14. SUBJECT TERMS High-level synthesis, low-power design, parallel algorithms, timing-driven synthesis, floorplanning			15. NUMBER OF PAGES 101	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

IMPROVED ALGORITHMS FOR HIGH-LEVEL SYNTHESIS AND
THEIR PARALLEL IMPLEMENTATIONS

BY

PRADEEP KUMAR PRABHAKARAN

B.Tech., University of Kerala, 1990
M.E., Indian Institute of Science, Bangalore, 1992
M.S., The Ohio State University, Columbus, 1993

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1998

Urbana, Illinois

© Copyright by Pradeep Kumar Prabhakaran, 1998

IMPROVED ALGORITHMS FOR HIGH-LEVEL SYNTHESIS AND THEIR PARALLEL IMPLEMENTATIONS

Pradeep Kumar Prabhakaran, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1998
Prithviraj Banerjee, Advisor

Scheduling, allocation and binding are the three major steps in high-level synthesis. Physical design typically involves floorplanning and routing which have been traditionally performed after high-level synthesis. With small device features in sub-micron technologies, since interconnection delays play a dominant part in the cycle time and the interconnect power dissipation is a major component of the energy consumption, it is important to consider the impact of physical design during high level synthesis.

Force directed scheduling is an effective scheduling algorithm which has been shown to be capable of arriving at close to optimal schedules. In the first part of this thesis, a novel extension of the force directed scheduling problem which naturally handles loops and conditionals by coming up with a scheme of scheduling *hierarchical* signal flow graphs is presented. With complex chips that need to be designed in the future, it is expected that the run times of scheduling algorithms will be quite large. Parallel processing has been shown to be quite effective in decreasing the run times of many CAD algorithms. Three new parallel algorithms for the scheduling problem are presented. These parallel algorithms are portable across a wide range of parallel platforms. Results are reported on a set of high-level synthesis benchmarks on SGI Origin and IBM SP-2. While some parallel algorithms for VLSI CAD reported by earlier researchers have reported a loss of qualities of results, the parallel algorithms for force directed scheduling presented in this thesis produce exactly the same results as the sequential algorithms on which they are based.

In the second part of this thesis, a simultaneous scheduling, binding and floorplanning algorithm is presented. In comparison to a traditional approach which separates high-level synthesis from physical design, our algorithm is able to make these stages interact very closely, resulting in solutions with better quality. In addition, a detailed model is considered, taking into account multiplexor and register areas and delays. A simultaneous scheduling, binding and floorplan-

ning algorithm for minimizing the interconnect energy is also presented which is able to reduce the interconnect energy by up to around 67 percent in comparison to a latency-driven approach, which does not consider power during high-level synthesis.

Although the algorithm which is able to make these stages interact very closely results in solutions with better quality, that approach results in increased run times. This run time can be effectively reduced using parallel processing. In the third part of the thesis, two parallel algorithms for simultaneous scheduling, binding and floorplanning algorithm are presented. Experimental results are reported on an IBM SP-2 multicomputer, with close to linear speedups for a set of benchmark circuits, while giving solutions of almost the same or better quality.

To my parents

ACKNOWLEDGMENTS

I am grateful and indebted to my advisor, Professor Prithviraj Banerjee, for his continuous encouragement and guidance. His ready support and accessibility have made it easy to work with him. I would also like to thank the members of my committee, Professors I. Hajj, W. Kubit, C. L. Liu and S. Muroga, for their valuable insights and comments in guiding this thesis.

I would like to thank all of the members of the Professor Banerjee's research group for their friendship as well as their technical help.

Lastly and most importantly, I would like to thank my family who have always supported me in all my endeavors and continue to offer their constant encouragement and understanding.

TABLE OF CONTENTS

CHAPTER	PAGE
1 INTRODUCTION	1
1.1 High-level synthesis	1
1.2 Need for physical design driven high-level synthesis	3
1.3 Need for low-power driven high-level synthesis	3
1.4 Need for parallel processing in high-level synthesis	4
1.5 Related work	5
1.5.1 Related work in scheduling	5
1.5.2 Related work in incorporating physical design information in high-level synthesis	5
1.5.3 Related work in low-power design in high-level synthesis	6
1.5.4 Related work in parallel algorithms for VLSI CAD	7
1.6 Thesis contributions	8
1.7 Thesis outline	9
2 AN ALGORITHM FOR FORCE DIRECTED SCHEDULING OF HIERARCHICAL SIGNAL FLOW GRAPHS	11
2.1 Introduction	11
2.2 Basic force directed scheduling	11
2.2.1 Scheduling of conditionals and loops in the basic force directed scheduling scheme	14
2.3 Hierarchical force directed scheduling	14
2.3.1 Distribution graph of a hierarchical entity	16
2.3.2 The scheduling step	18
2.3.3 Speeding up the <i>DG</i> calculation by reuse of <i>sub-DGs</i>	20
2.4 Experimental results for hierarchical graphs	21
2.5 Summary	21
3 PARALLEL ALGORITHMS FOR FORCE DIRECTED SCHEDULING	22
3.1 Introduction	22
3.2 Parallel force directed scheduling algorithm for non-hierarchical graphs	23
3.2.1 Node based problem partitioning	23
3.2.2 Step based problem partitioning	25
3.3 Parallel hierarchical force directed scheduling	27

3.3.1	Node partitioning step	29
3.3.2	Pre-processing step	30
3.3.3	Force calculation and scheduling	30
3.3.4	An example	31
3.3.5	Load balancing	33
3.4	Analysis	33
3.4.1	Pre-processing step	33
3.4.2	The force-calculation and scheduling phase	34
3.5	Experimental results	34
3.5.1	Experimental results for non-hierarchical graphs	34
3.5.2	Experimental results for hierarchical graphs	36
3.6	Summary	37
4	SIMULTANEOUS SCHEDULING, BINDING AND FLOORPLANNING IN HIGH-LEVEL SYNTHESIS FOR LATENCY MINIMIZATION	38
4.1	Introduction	38
4.2	Simulated annealing	40
4.3	The floorplanning algorithm	41
4.3.1	Slack calculation	42
4.3.2	Clustering phase	45
4.3.3	Bounding curve calculation	46
4.3.4	Reorientation phase	48
4.3.5	Calculation of interconnect delays	49
4.4	The combined scheduling, binding and floorplanning algorithm	50
4.4.1	Cost function	52
4.4.2	Register and multiplexor allocation	52
4.5	The two-step algorithm	52
4.6	Experimental results	53
4.6.1	Comparison between the combined scheme and Fang and Wong's algorithm	56
4.6.2	Comparison between the combined scheme and a two-step scheme which uses a constructive floorplanning algorithm	57
4.6.3	Comparison between the combined scheme and a two-step scheme which uses a simulated annealing based floorplanning algorithm	57
4.7	Summary	60
5	SIMULTANEOUS SCHEDULING, BINDING AND FLOORPLANNING IN HIGH-LEVEL SYNTHESIS FOR INTERCONNECT ENERGY MINIMIZATION	62
5.1	Introduction	62
5.2	Power estimation using switching tables	63
5.3	The constructive power-driven floorplanning algorithm	64
5.3.1	Cost matrix	65
5.3.2	Transition matrix calculation	66

5.3.3	Estimation of interconnect energy	66
5.4	Overall algorithm	67
5.4.1	Cost function	67
5.5	Experimental results	68
5.6	Summary	69
6	PARALLEL ALGORITHMS FOR SIMULTANEOUS SCHEDULING, BINDING AND FLOORPLANNING	70
6.1	Introduction	70
6.2	Parallel algorithms for combined scheduling, binding and floorplanning	71
6.2.1	Synchronous multiple Markov chains	71
6.2.2	Asynchronous multiple Markov chains	72
6.3	Experimental results	75
6.3.1	Experimental results for the latency-driven scheme	75
6.3.2	Experimental results for the power-driven scheme	78
6.4	Summary	78
7	CONCLUSION AND FUTURE WORK	82
7.1	Thesis summary	82
7.2	Future work	83
	REFERENCES	84
	VITA	89

LIST OF TABLES

Table	Page
2.1 Comparison of hierarchical force directed scheduling with the optimal results . . .	21
3.1 Run times(seconds) and speedups for the <i>PARFDS</i> tool using node-based approach for non-hierarchical graphs on an SGI Origin multiprocessor	35
3.2 Run times (seconds) and speedups for the <i>PARFDS</i> tool using node-based approach for non-hierarchical graphs on IBM SP-2 multicomputer	35
3.3 Run times(seconds) and speedups for the <i>PARFDS</i> tool using step-based approach for non-hierarchical graphs on an SGI Origin multiprocessor	36
3.4 Run times for step-based approach for the <i>PARFDS</i> tool for non-hierarchical graphs on IBM SP-2 multicomputer	36
3.5 Run time(seconds) and speedup for the <i>PARHIERFDS</i> tool for hierarchical graphs on IBM SP-2 message passing multicomputer	37
3.6 Run time(seconds) and speedups for the <i>PARHIERFDS</i> tool for hierarchical graphs on SGI Origin multiprocessor	37
4.1 The library set	54
4.2 Comparison between Fang and Wong algorithm and the combined scheme for 2-step FIR filter	56
4.3 Comparison between Fang and Wong algorithm and the combined scheme for 3-step FIR filter	56
4.4 Comparison between two-step (using constructive floorplanner) and combined schemes for synthetic benchmark	57
4.5 Comparison between two-step (using constructive floorplanner) and combined schemes for 2-step FIR	58
4.6 Comparison between two-step (using constructive floorplanner) and combined schemes for 3-step FIR	58
4.7 Comparison between two-step (using constructive floorplanner) and combined schemes for 8-step Elliptic filter	59
4.8 Comparison between two-step (using annealing based floorplanner) and combined schemes for synthetic benchmark	60
4.9 Comparison between two-step (using annealing based floorplanner) and combined schemes for 2-step FIR	60
4.10 Comparison between two-step (using annealing based floorplanner) and combined schemes for 3-step FIR	61

4.11	Comparison between two-step (using annealing based floorplanner) and combined schemes for 8-step Elliptic filter	61
5.1	Comparison between latency-driven and power-driven schemes for 2-step FIR . .	68
5.2	Comparison between latency-driven and power-driven schemes for 3-step FIR . .	69
5.3	Comparison between latency-driven and power-driven schemes for 8-step Elliptic filter	69
6.1	Quality measures and run times for the sequential implementation for the latency driven algorithm	76
6.2	Quality comparisons for asynchronous Markov-chain approach for the latency driven algorithm	76
6.3	Quality comparisons for synchronous Markov-chain approach for the latency driven algorithm	76
6.4	Quality measures and run times for the sequential implementation of the power driven algorithm	79
6.5	Quality comparisons for asynchronous Markov-chain approach for the power driven algorithm	79
6.6	Quality comparisons for synchronous Markov-chain approach for the power driven algorithm	79

LIST OF FIGURES

Figure	Page
1.1 An example signal-flow graph	2
1.2 The example signal-flow graph scheduled to three steps	2
1.3 The example signal-flow graph after binding	2
1.4 Organization of the thesis	10
2.1 Figure showing distribution graphs for (a) no nodes scheduled and (b) node 3 scheduled to step 5	12
2.2 A hierarchical signal flow graph	15
3.1 Node based decomposition for a non-hierarchical graph for 4 processors	25
3.2 Step-based decomposition for a non-hierarchical graph for 4 processors	27
3.3 Syntax tree of example hierarchical graph with processor assignments	28
4.1 Effect of considering floorplanning during scheduling and binding	39
4.2 Slack distribution algorithm	44
4.3 Slacks for a sample SDFG and delay values after slack distribution	45
4.4 Example showing how combining clusters based on area affects the overall area of the floorplan	46
4.5 Computation of augmented bounding curve of a cluster	48
4.6 Reducing interconnect cost by reorientation	49
4.7 Representation of scheduling and binding problem as a placement problem in a two-dimensional table	50
4.8 Simultaneous scheduling, binding and floorplanning	51
4.9 Signal flow graph of FIR filter	53
4.10 Signal flow graph of synthetic benchmark	54
4.11 Signal flow graph of Elliptic filter benchmark	55
5.1 Example showing switching along edges between modules	65
5.2 Simultaneous scheduling, binding and floorplanning for low-power	67
6.1 Synchronous multiple Markov chains	72
6.2 Synchronous multiple Markov chains	73
6.3 Asynchronous multiple Markov chains	74
6.4 Asynchronous multiple Markov chains	74
6.5 Speed-up comparisons for FIR-2step filter for the latency driven algorithm	77

6.6	Speed-up comparisons for FIR-3step filter for the latency driven algorithm	77
6.7	Speed-up comparisons for Elliptic filter for the latency driven algorithm	78
6.8	Speed-up comparisons for FIR-2step filter for the power driven algorithm	80
6.9	Speed-up comparisons for FIR-3step filter for the power driven algorithm	80
6.10	Speed-up comparisons for Elliptic filter for the power driven algorithm	81

CHAPTER 1

INTRODUCTION

1.1 High-level synthesis

High-level synthesis translates a design description specified at a behavioral level into a register transfer level(RTL) design. The behavioral description is usually specified using hardware modeling languages like VHDL [54], Verilog [55] and HardwareC [56]. The signal and control flow can be conveniently represented using a signal-flow graph. Figure 1.1 shows an example signal-flow graph which represents a behavioral description shown.

High-level synthesis consists of two main stages. The first stage assigns each operation in the signal-flow graph to a specific time-step during which it will be executed, and a functional unit that will perform the operation. The result of this stage is a data-path description which consists of a set of functional modules, memory unit, registers and multiplexors, and their interconnection. The second stage is to generate a control unit which generates the control signals for each time step.

Traditionally, the first stage of high-level synthesis has been organized into three steps, namely scheduling, allocation and binding. Scheduling assigns each node of the signal flow graph into specific time steps. Allocation determines the number of units of each type of functional unit. Binding assigns a specific functional unit that will perform each operation. Figure 1.2 shows the example graph in Figure 1.1 scheduled in three steps, and Figure 1.3 shows the same graph after the binding stage, assuming two adders and one multiplier is allocated.

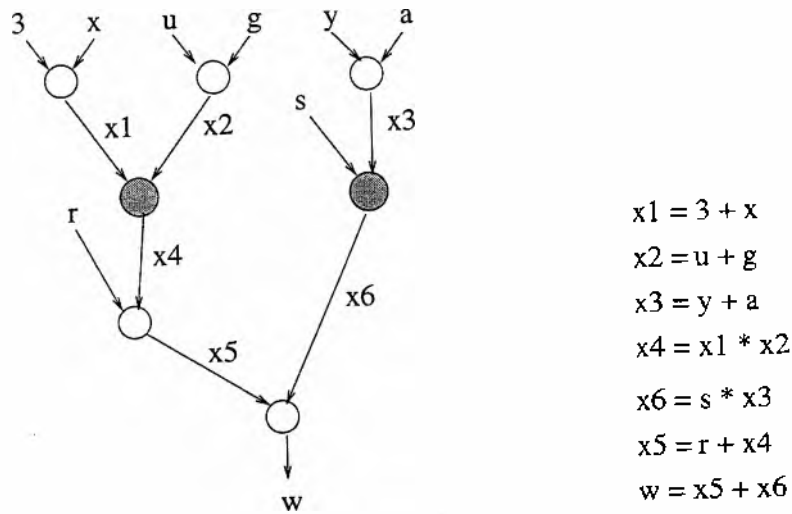


Figure 1.1 An example signal-flow graph

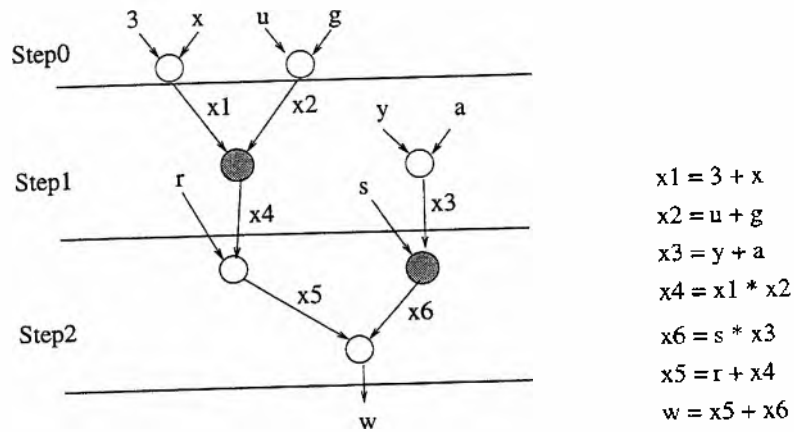


Figure 1.2 The example signal-flow graph scheduled to three steps

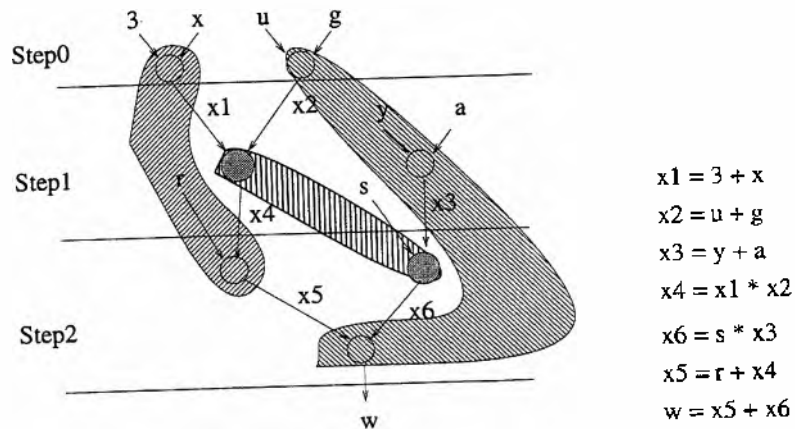


Figure 1.3 The example signal-flow graph after binding

A register transfer level design is characterized by three attributes, namely latency or the number of time steps, the area and the cycle time. In a typical design process, one of them is fixed by the user and the high-level synthesis system attempts to optimize the others. Area has to take into account the areas of functional units, registers, memory, control unit, steering logic and the interconnections. Cycle time is determined by the maximum propagation delay in a time step which includes the functional unit delays, register and multiplexor delays, interconnect delays and the control unit delay.

1.2 Need for physical design driven high-level synthesis

With small device features in sub-micron technologies, interconnection delays play a dominant part in cycle time. The minimum dimension of the transistor has decreased from $2\mu m$ in 1985 to $0.35\mu m$ in 1996. It will further decrease at the rate of 0.7x per generation to reach $0.07\mu m$ by 2010 [33], [34]. Such rapid scaling results in much higher degree of on-chip integration. In addition, the circuit performance will increasingly be determined by the interconnect performance. In addition to the interconnect delay being an important component of the cycle time, the interconnect power dissipation is a major component of the energy consumption. Therefore, it is important to consider the impact of physical design during high level synthesis.

1.3 Need for low-power driven high-level synthesis

Sub-micron features have lead to an increased fraction of the power to be dissipated by interconnects (up to 40 percent as reported in [11], [32]). Recently, there has been a proliferation of battery operated portable devices that demand intensive computation. Prolonging battery life requires design of power-efficient circuits. In order to increase the battery lifetime for portable applications, it is necessary to decrease the average power dissipation. In addition, decreasing the interconnect power is important to increase reliability in sub-micron technology by reduc-

ing electro-migration and hot-electron effects. Further, by decreasing the charge drawn by the circuit, ground bounce and excessive voltage drop on buses can be minimized.

1.4 Need for parallel processing in high-level synthesis

With the rapid improvement in VLSI technology, circuit design is becoming extremely complex and is placing increasing demands on CAD tools. Parallel processing is fast becoming an attractive solution to reduce the inordinate amount of time spent in VLSI circuit design [39]. Many workstation vendors have announced products based on multiprocessors. Furthermore, most CAD environments have large networks of workstations that can be used as parallel machines. It is clear that many CAD software vendors will soon be announcing products that will exploit the parallelism in the hardware.

Scheduling algorithms for data flow graphs corresponding to small designs containing about 50-100 nodes can take several minutes to execute on conventional workstations. With complex chips that need to be designed in the future containing thousands of nodes, it is expected that the run times of these algorithms will reach hours. Designers of high level synthesis tools wish to explore a very large design space of solutions with various architectural alternatives (e.g. number of adders, multipliers, registers, buses, memory ports, etc.) for various values of expected latencies. The run times for exploring all the different choices can easily take several hours. This task is very interactive in nature, and a designer typically would want to have the solutions in less than a minute, and not hours. In addition, researchers are proposing more complex cost functions to produce designs with higher speeds, lower power and increased testability during the scheduling process. Hence, it is important to investigate efficient parallel algorithms for the problem which can reduce the run times from hours to minutes.

1.5 Related work

The related research work in literature can be broadly classified into four areas as described in the following sections

1.5.1 Related work in scheduling

Numerous algorithms have been proposed for scheduling [40], [43], [44], [46], [47]. An excellent overview of existing scheduling algorithms appears in [45]. An integer linear programming approach to scheduling is described in [46] and [47]. Integer linear programming is suitable for small systems, but may result in very large run times for larger problems. List scheduling [43], [44] is a fast algorithm which tries to find a minimal schedule, given the resource constraints. Force directed scheduling [40], introduced by Paulin and Knight tries to minimize the number of resources for a given latency by trying to smooth out the resource requirement for different time steps within a given latency. Force directed scheduling have been found to be extremely effective and has gained interest since its introduction among other research groups [48], [49], [50], [51], [52]. Force directed list scheduling (FDLS) [40], which is a variation of force directed scheduling in which the user specifies the resources available and the signal flow graph is scheduled with minimum latency given the resource constraints.

1.5.2 Related work in incorporating physical design information in high-level synthesis

Previous researchers have addressed the problem of incorporating physical design information in high level synthesis. Fang and Wong [15] describes a integrated binding and floorplanning algorithm which performs a constructive binding for each move of a simulated annealing based floorplanning algorithm.

Weng and Parker [17] describe an algorithm for simultaneous scheduling, binding and floorplanning. They perform the floorplanning of modules in critical paths, followed by an iterative improvement phase which performs re-binding to reduce the latency.

Jang and Pangrle [31] performs connectivity binding based on geometric costs for bit-sliced designs. It is a two-pass approach. In the first pass, nodes are assigned to functional units based on connectivity using a bipartite weighted matching. In the second pass, long arcs are reduced by re-assignment of nodes and by introducing redundant nodes such as pass-through nodes or duplicate nodes.

Moshnyaga et.al [16] describes an algorithm which incorporates performance-driven placement in the module selection phase of the synthesis. It is an iterative approach in which a performance-driven placement and a mapping of faster modules to critical resources are performed in each iteration until a stop criterion is met.

Fasolt [18] performs floorplanning and analyzes the area impact during scheduling and binding. It automatically optimizes the bus topology of an RTL design using optimizing transformations of an existing design.

BUD [19], ELF [21] and Chippe [20] are other algorithms which attempt to incorporate physical design information in high-level synthesis.

1.5.3 Related work in low-power design in high-level synthesis

There has been considerable amount of research in considering power during high-level synthesis. A simultaneous scheduling and binding algorithm is described in [6] which tries to minimize the switching activity on buses in a bus-based architecture. It starts with a micro-architecture which satisfies the area and performance bounds, and refines switching activity on the buses using simulated annealing to arrive at an energy-efficient architecture. A high-level datapath power estimation algorithm which derives accurate power estimates for general *CDFGs* has been presented in [1].

An allocation method and controller optimization algorithm for low power is presented in [5]. It assigns weights to the edges of the *CDFG* based on capacitance and transition activity estimates. A high value of the weight between two nodes suggests that they should be assigned to the same resource to minimize switching activity and capacitance. It performs binding by iteratively assigning nodes connected by highest values of the weights to the same resource.

In addition, the don't care values of the controller are selected in order to minimize switching activity in idle components. An interactive improvement method for scheduling, clock selection, module selection and resource allocation with the aim of reducing the power consumption has been presented by the same authors in [4].

An algorithm for optimizing total power consumption and area during selection and placement of modules is presented in [7]. It describes a simulated annealing based floorplanning algorithm in which power-indexed shape curves are used to determine aspect ratios of modules in a slicing structure with the objective of minimizing the overall power.

Computational transformations such as operation reduction, operation substitution, word length reduction and transformation ordering have been used by [9] to reduce power. An in-depth survey of power minimization algorithms at architectural, logical and physical levels appears in [10].

1.5.4 Related work in parallel algorithms for VLSI CAD

There has been extensive research in the field of using parallel processing for VLSI CAD as is evident in the recent literature for cell placement [58], [61], [62], circuit extraction [60], test generation [64], [65] and logic synthesis [66], [72], [73].

The work described in this thesis is part of an ongoing project called ProperCAD [57] which is aimed at developing an integrated suite of parallel applications for VLSI CAD that run on a variety of parallel platforms. In the past, various tools have been developed to solve the problems at lower levels in the VLSI CAD hierarchy, namely, placement [58], routing [59], circuit extraction [60], design rule checking [68], logic synthesis [69], test generation [67], [71], and fault simulation [70].

Some researchers [53] have proposed a parallel system for distributed high level synthesis which uses coarse-grained parallelism to explore and evaluate many alternative VLSI designs efficiently. They propose a distributed version of force directed list scheduling in which each processor executes the FDLS algorithm on a separate module set. Force directed scheduling algorithm for minimizing resource requirement for a given latency assumes that the user speci-

fies the latency for each loop body. In comparison to that work, we propose parallel algorithms for force directed scheduling which exploits parallelism in the calculation of forces for various nodes in different steps. In addition, we also propose an extension of force directed scheduling algorithm which naturally handles loops and conditionals, and which automatically selects the latencies of the loop bodies. We assume that the overall latency of the signal flow graph is given, and attempt to minimize the resource requirement.

There has been published work related to the problem of performing simulated annealing in parallel. Aarts [37] et al introduced the concept of multiple Markov chains for parallel simulated annealing. Lee and Lee [36] refined it into an asynchronous approach and applied it to the problem of graph partitioning on shared memory multiprocessors. Chandy and Banerjee [38] applied asynchronous and synchronous versions of the multiple Markov chain approach to the problem of standard cell placement in an Actor [38] framework.

1.6 Thesis contributions

The thesis makes contribution in into two areas. In the first area, we describe novel sequential algorithms for high-level synthesis and in the second, we describe parallel algorithms for high-level synthesis. The specific contributions of the first part are

- An extension of force directed scheduling for handling hierarchical signal flow graphs (Chapter 2). It was able to generate close to optimal solutions for some benchmark circuits.
- A simultaneous scheduling, binding and floorplanning algorithm with the objective of minimizing the latency (Chapter 4). It was able to reduce the interconnect delays by up to around 46 percent compared to a two-step algorithm for benchmark circuits.
- A simultaneous scheduling, binding and floorplanning algorithm with the objective of optimizing interconnect energy dissipation (Chapter 5). It was able to reduce the interconnect energy by up to around 67 percent for practical benchmark circuits.

The contributions of the second part are

- Two parallel algorithms for force directed scheduling of non-hierarchical signal-flow graphs and one for hierarchical signal flow graphs (Chapter 3). They were able to generate solutions having the same quality as the sequential algorithm. In addition, speedups up to 7.68 and 7.31 on 8 processors were obtained on IBM SP-2 multicomputer, and SGI origin multiprocessor, respectively.
- Parallel algorithms for simultaneous scheduling, binding and floorplanning for optimizing the latency and the interconnect energy (Chapter 6). They were able to generate solutions of almost same quality as the sequential algorithm with close to linear speedups on the IBM SP-2 multicomputer.

1.7 Thesis outline

Chapter 2 describes an extension of the force directed scheduling algorithm to handle loops and conditional statements in a natural fashion. Chapter 3 describes three parallel algorithms for force directed scheduling. Chapter 4 describes a simultaneous scheduling, binding and floorplanning algorithm with the objective of minimizing the latency, based on simulated annealing. Chapter 5 describes a simultaneous scheduling, binding and floorplanning to minimize interconnect energy. Chapter 6 describes two parallel algorithms for simultaneous scheduling, binding and floorplanning. Chapter 7 concludes the thesis and describes areas of future work.

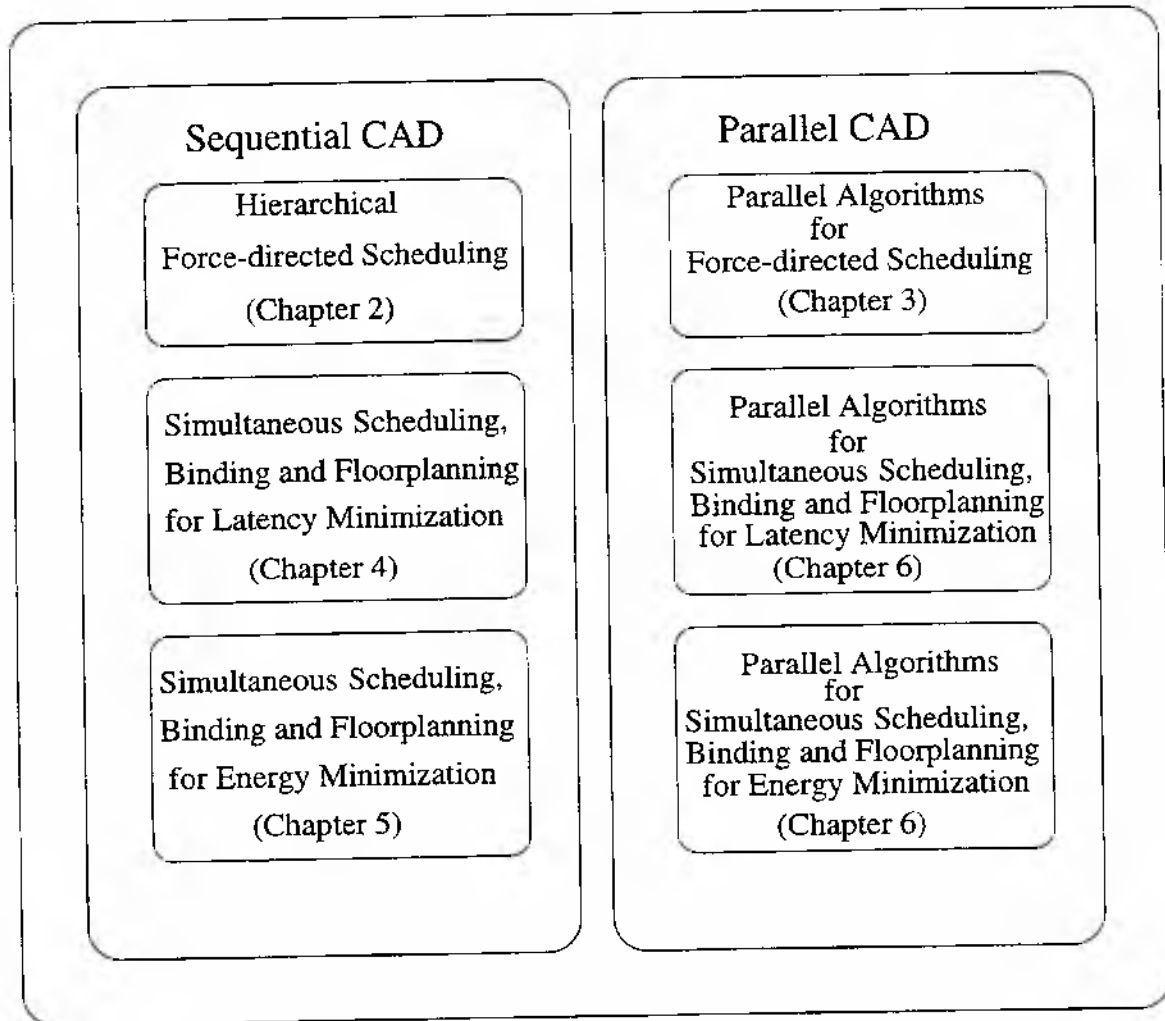


Figure 1.4 Organization of the thesis

CHAPTER 2

AN ALGORITHM FOR FORCE DIRECTED SCHEDULING OF HIERARCHICAL SIGNAL FLOW GRAPHS

2.1 Introduction

In this chapter, a novel extension of the force directed scheduling algorithm which naturally handles loops and conditionals by coming up with a scheme of scheduling *hierarchical* signal flow graphs, is presented.

In Section 2.2 the basic force directed scheduling algorithm is summarized. In Section 2.3 we propose a sequential force directed scheduling algorithm for hierarchical graphs. Section 2.4 we give the experimental results.

2.2 Basic force directed scheduling

The basic force directed scheduling algorithm [40] is an iterative algorithm which schedules one operation each iteration. The operation to be scheduled is selected based on a quantity termed *force* defined for each step. It is a measure of concurrency at that step. The force directed scheduling tends to balance the concurrency at each step without lengthening the execution time. First, it calculates the time frame of each operation, namely the time interval from the earliest start time to the latest start time for that operation. It is done by calculating the *as*

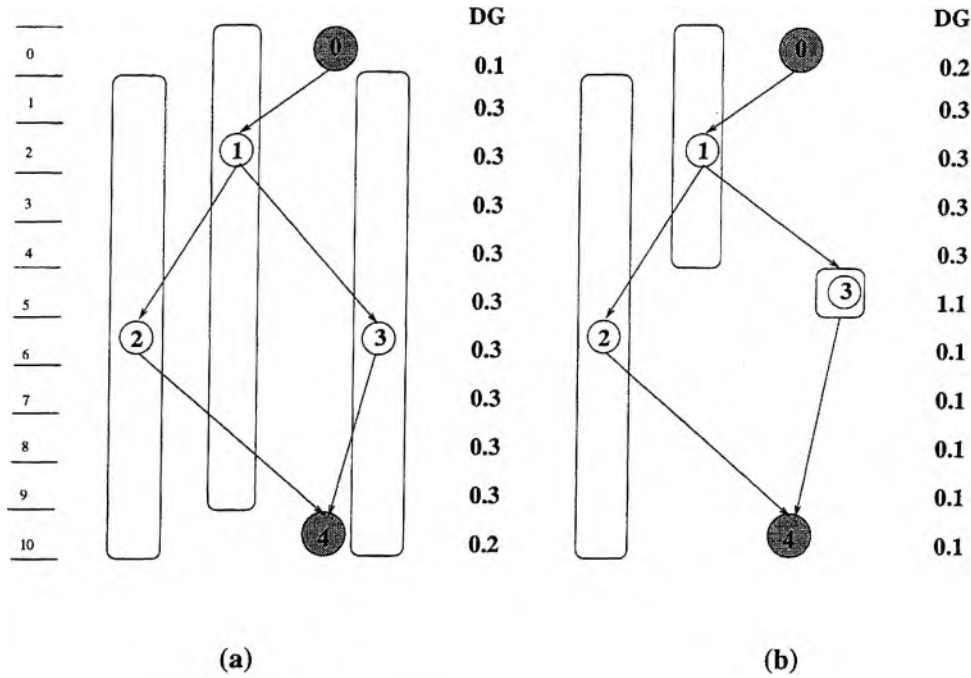


Figure 2.1 Figure showing distribution graphs for (a) no nodes scheduled and (b) node 3 scheduled to step 5

soon as possible (ASAP) and as late as possible (ALAP) schedule of the graph. At each step, the algorithm determines the force of each node to each step in its time frame. It then selects the node with the least force and schedules it to the corresponding step. The force consists of two components, *self* and *predecessor-successor (ps)* forces. Self force of a node n to a step s is a measure of the increase in concurrency due to scheduling that node to that step. Predecessor and successor forces are measures of increase in concurrency of the predecessors and successors, respectively as a result of scheduling n to s .

The concurrency of operations can be captured by a *distribution graph (DG)* for each operation type. For each operation type k , the *DG* in step i is given by

$$DG(i) = \sum_k Prob(op, i)$$

the sum being taken over all operations of type k . The mechanical analog for *DG* is the spring constant. Figure 2.1 shows an example of *DGs* for an example signal-flow graph.

The self force associated with the assignment of an operation with time frame from t to b to time step j ($t \leq j \leq b$) is given by

$$self_force(j) = \sum_{i=t}^b [DG(i) * x(i)]$$

where $x(i)$ is the change in operation probability (the mechanical analog being the displacement of spring), given by

$$x(j) = (h - 1)/h$$

and

$$x(i) = -1/h$$

where $h = (b - t + 1)$.

The predecessor force for the assignment of a node to a step j is the sum of the forces of the predecessors of the node arising out of change in time frames and the resulting change in concurrency. For a particular predecessor, it is quantified as:

$$Force(nt, nb) = \sum_{i=nt}^{nb} [DG(i)/(nb - nt + 1)] - \sum_{i=t}^b [DG(i)/(b - t + 1)] \quad (2.1)$$

where the interval from t to b is the old time frame and that from nt to nb is the new time frame.

The successor force is defined analogously.

The basic force directed scheduling algorithm is summarized below:

FORCE-DIRECTED()

- 1 **repeat**
- 2 *Evaluate time frames :*
- 3 2.1. *Find ASAP schedule*
- 4 2.2. *Find ALAP schedule*
- 5 *Update distribution graphs*
- 6 *calculate self, predecessor and successor forces,*
- 7 *Add them to get the total force for each feasible control step*

- 8 *Schedule operation with lowest force;*
- 9 *Set its time frame equal to the selected step.*
- 10 **until** *all operations scheduled*

2.2.1 Scheduling of conditionals and loops in the basic force directed scheduling scheme

The alternates of a conditional are mutually exclusive. Thus, for a step in which mutually exclusive operations intersect, the probability of only the operation with the highest probability is added to the corresponding *DG*. The force calculation and scheduling proceeds exactly the same way as described earlier for an acyclic graph without conditional structures.

When a loop is part of the behavioral description, the user have to specify a constraint on the loop iteration time or, alternately a constraint of the number of structural units available. For multiple embedded loops, the operations of the innermost loop are scheduled first, relative to the local timing constraint. After that, the entire loop is taken as a single operation with execution time equal to the loop's local time constraint.

Further details of the basic force directed scheduling can be found in [40].

2.3 Hierarchical force directed scheduling

As described in the previous section, existing approach for force directed scheduling of hierarchical signal flow graphs assumes that the user has to specify the latency of each loop. We extend the basic force directed scheduling scheme to naturally handle hierarchical signal flow graphs, so that the latencies of loops are selected automatically to minimize the total resource requirement.

Definition 1 A hierarchical graph is a signal flow graph in which each node can be

- a block, which is a set of nodes enclosed by a source and a sink

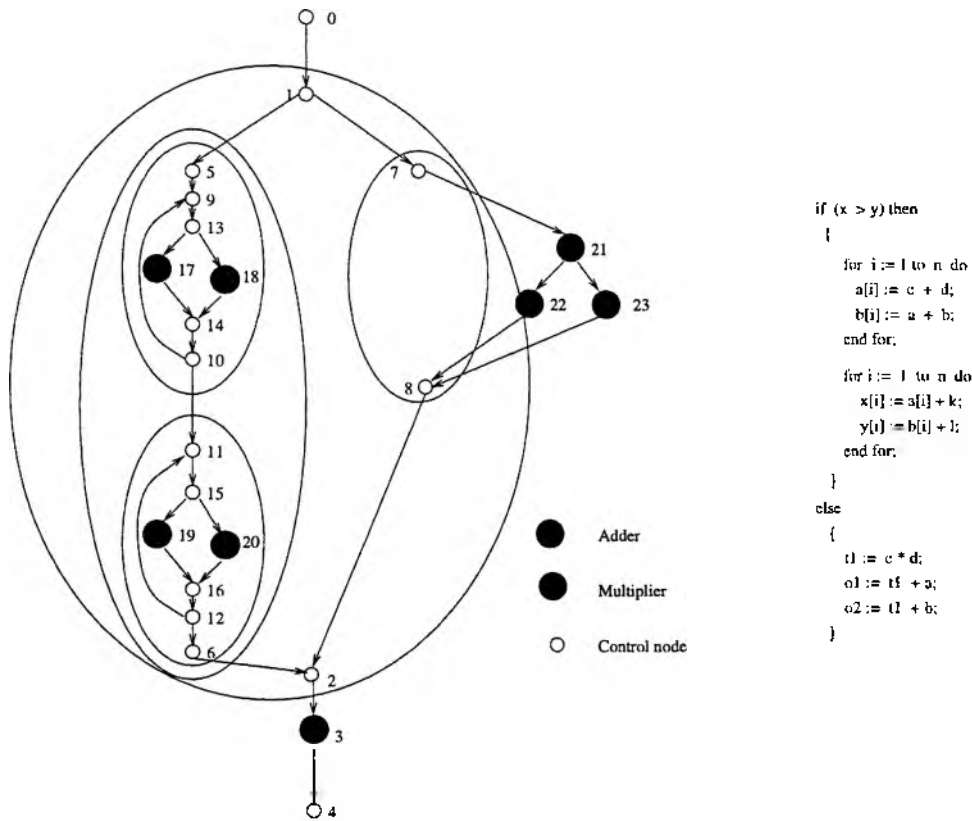


Figure 2.2 A hierarchical signal flow graph

- *atomic operation (eg an addition or multiplication) or*
- *a loop which encloses a block*
- *a conditional structure which encloses one or more alternates each of which are blocks.*

Figure 2.2 shows an example hierarchical signal flow graph. Node 0 is the source and node 4 is the sink. Node 1 is a conditional. Nodes 5 and 7 are the source nodes of the alternates for the conditional. Node 9 and 11 are source nodes for loops. Nodes 9 and 10 are the source and the sink, respectively of the loop body of the loop whose source is 5. The nodes of a hierarchical graph can be broadly classified as *control nodes* and *non-control nodes*. Control nodes include *source* and *sink* nodes which enclose hierarchical entities, *loop-source* and *loop-sink* which enclose a loop, and *if-source* and *if-sink* which enclose a conditional. Control nodes do not require resource allocation. The *loop-source* node contains information about the loop

such as the number of iterations. The *non-control* nodes are the regular nodes (eg adder) which require resource allocation. We assign a level number to each node. The level number of a node corresponds to the depth of nesting of that node in the signal flow graph. The nodes enclosed by the outermost *source-sink* pair are assigned a level number 0. The next inner level nodes (those enclosed by hierarchical nodes having level number 0) are assigned a level number 1 and so on.

2.3.1 Distribution graph of a hierarchical entity

For each hierarchical entity h , we can associate a corresponding *sub-DG*, given by the tuple $sub_DG_h^k = \langle P, lb, ub, k \rangle$ for each operation type k where lb and ub are the lower and upper limits for the earliest and latest start times respectively for all nodes enclosed by h , P is a vector such that $P(i)$, $lb \leq i \leq ub$ is the sum of the probabilities over all nodes $n \in h$ that n will be executed during time step i , and k is the type of the operation (eg adder, multiplier). The *sub-DG* of a hierarchical entity depends on the type of the entity. The calculation of *sub-DGs* for the various types of hierarchical *DGs* are explained below. In the following, $sub_DG_n^k(i)$ means the same as $P(i)$, the i^{th} element of the member P of the tuple $sub_DG_h^k$.

2.3.1.1 Calculation of *sub-DG* for an atomic node

The *sub-DG* for an atomic node a of type k is given by $sub_DG_a^k = \langle ADG_a, lb_a, ub_a, k \rangle$, where $lb_a = ASAP(a)$, $ub_a = ALAP(a) + l_k$ and

$$ADG_a(i) = \frac{(\sum_{j=ASAP(a)}^{ALAP(a)} \alpha(i, j))}{(ALAP(a) - ASAP(a) + 1)}$$

where l_k is the latency of operation of type k , and

$$\alpha(i, j) = \begin{cases} 1 & \text{if } j \leq i < j + l \\ 0 & \text{otherwise} \end{cases}$$

2.3.1.2 Calculation of sub_DG for a block

For a straight line block of code B , we calculate the sub_DG by recursively calculating the sub_DG s of each component node and adding them up for each step. Let $\langle P_j, lb_j, ub_j, k \rangle$ be the sub_DG for node j , where $j \in B$. The overall sub_DG of the block for each operation type k is given by $sub_DG_B^* = \langle BDG_B, lb_B, ub_B, k \rangle$ where

$$lb_B = \min_{j \in B} lb_j$$

$$ub_B = \max_{j \in B} ub_j$$

and

$$BDG_B(i) = \sum_{j \in B} P_j(i)$$

2.3.1.3 Calculation of sub_DG for a loop

For a loop structure, we determine the maximum and minimum latencies of the loop body, recursively calculate the sub_DG of the loop body for each start time of the loop source and for each corresponding latency and compute their weighted sum, the weights being the probability that the loop body is assigned that latency and the loop source has that particular start time. We assume that all the possible start times of the source and all latencies for that particular start time are equally probable.

Let L be a loop structure which executes for n iterations. Let e_{src} and l_{src} be the earliest and the latest start time of the source node of L and e_{snk} and l_{snk} be the earliest and latest start times of the sink of L respectively. For a particular start time s , $s \leq (l_{snk} - n * (e_{snk} - e_{src}))$ for the loop source, the maximum latency of the loop body is given by

$$l_{max} = (l_{snk} - s)/n$$

and the minimum latency by

$$l_{min} = (e_{snk} - e_{src})/(n - 1)$$

Let $\langle BDG_B^l, lb_B, ub_B, k \rangle$ be the *sub_DG* for the body of the loop for a given latency l and for operation type k . Then the *sub_DG* for L , denoted by $sub_DG_L^{s,l}$ for start time s for the loop source and latency l and is given by

$$sub_DG_L^{s,l} = \langle LDG_L^{s,l}, lb_L^{s,l}, ub_L^{s,l}, k \rangle$$

where $LDG_L^{s,l}(i) = BDG_B(i \bmod l)$, $lb_L^{s,l} = s$ and $ub_L^{s,l} = s + n \times l$. The overall *sub_DG* for L , is given by the tuple $sub_DG_L = \langle LDG_L, lb_L, ub_L, k \rangle$, where $lb_L = e_{src}$, $ub_L = l_{snk}$ and

$$LDG(i) = \sum_{s=e_{src}}^{l_{src}} \frac{1}{(l_{src} - e_{src} + 1)} \sum_{l=l_{min}}^{l_{max}} \frac{LDG_L^{s,l}(i)}{(l_{max} - l_{min} + 1)}$$

2.3.1.4 Calculation of *sub_DG* for conditional structure

For a conditional structure C , we recursively calculate the *sub_DGs* of each alternate for the conditional. The overall *sub_DG* is the *DG* formed by taking the stepwise maximum of the above *sub_DGs* at each step. Let the conditional structure have m alternates. Let $\langle P_j, lb_j, ub_j, k \rangle$ be the *sub_DG* for the j^{th} alternate. Then, the overall *DG* of the conditional structure is given by $sub_DG_C = \langle CDG_C, lb_C, ub_C, k \rangle$ where

$$lb_C = \min_{j=1}^m lb_j$$

$$ub_C = \max_{j=1}^m ub_j$$

and

$$CDG_C(i) = \max_{j=1}^m P_j(i)$$

for each operation type k .

2.3.2 The scheduling step

The hierarchical force directed algorithm proceeds level by level, scheduling the nodes at each higher level before proceeding to the next lower level. Consider a node n . Let $Succ(n, k)$

be the set of successors of n of type k and let $Pred(n, k)$ be the set of predecessors of n of type k which are in the same level as n in the signal flow graph. Define $Succ_DG(n, k, s)$ to be the aggregate DG of all nodes of type k in $Succ(n, k)$ when n is scheduled to step c . Similarly, $Pred_DG(n, k, s)$ is the aggregate DG of all nodes in $Pred(n, k)$ when n is scheduled to step c . Let $\langle sub_DG_j, lb_j, ub_j, k_j \rangle$ be the sub_DG of node j . Then, $Succ_DG(n, k, c)$ is given by

$$Succ_DG(n, k, c) = \langle SDG_n, lb_n^s, ub_n^s, k \rangle$$

where $lb_n^s = \min_{t \in Succ(n, k)} lb_t$, $ub_n^s = \max_{t \in Succ(n, k)} ub_t$, and $SDG_n(j) = \sum_{t \in Succ(n, k)} sub_DG_t(j)$. $sub_DG_t(j)$ is assumed to be zero if $j > ub_t$ or $j < lb_t$.

Similarly,

$$Pred_DG(n, k, c) = \langle PDG_n, lb_n^p, ub_n^p, k \rangle$$

where $lb_n^p = \min_{t \in Pred(n, k)} lb_t$, $ub_n^p = \max_{t \in Pred(n, k)} ub_t$, and $PDG_n(j) = \sum_{t \in Pred(n, k)} sub_DG_t(j)$.

The self force of a control node is defined to be zero. The self force for a non-control node is the same as that in the case of the basic force directed scheduling algorithm.

The predecessor force of n for time step c is given by

$$\sum_{k=1}^{numtypes} \sum_{i=lb}^{ub} (P[i] + DG[k][i]/3) \times DG[k][i] \quad (2.2)$$

and successor force by

$$\sum_{k=1}^{numtypes} \sum_{i=lb}^{ub} (P[i] + DG[k][i]/3) \times DG[k][i] \quad (2.3)$$

where the lb, ub and P in the first and second formula equals the corresponding lb, ub and P fields of $Pred_DG(n, k, c)$ and $Succ_DG(n, k, c)$ respectively. The term $DG[k][i]/3$ is the look-ahead factor. The total force is given by the sum of self, predecessor and successor forces. The total force of all nodes to all feasible time steps are determined and the node with the least force is scheduled.

2.3.3 Speeding up the *DG* calculation by reuse of *sub_DGs*

For hierarchical entities, it can be observed that the *sub_DG* depends only on the *relative* positions of the source and the sink, not the absolute values of their earliest or latest start times, as long as the scheduled status of the interior nodes does not change. Thus, the *sub_DGs* are invariant under translation. Since the calculation of *sub_DG* of a hierarchical entity usually involves calculation of the *sub_DGs* of its components repeatedly with different values for the earliest start time of the source or sink, but with the same latency, the *sub_DG* for those components can be calculated once, and then reused with suitable translation. This results in an order of magnitude improvement in run time. In practice, we maintain a table which stores the *sub_DGs* of hierarchical entities for each latency encountered so far. Each table entry has a *valid* bit which indicates whether the corresponding *sub_DG* is correct. Initially, all the entries in the table are invalid. When the algorithm needs the *sub_DG* for a node for a given latency, it first checks the table. If it encounters a valid entry, it can directly use the stored result. If the table entry is invalid, it calculates the *sub_DG*, stores it in the table and sets the valid bit. As long as none of the nodes within the hierarchical node is scheduled, the table entry remains valid, and can be reused. After a schedule step, all entries in the table corresponding to the hierarchical entities enclosing the nodes scheduled are invalidated.

The overall sequential algorithm is:

FORCE-DIRECTED-HIERARCHY()

- 1 **for** level $\leftarrow$ 0 **to** number of levels
- 2 **do repeat**
- 3 *Evaluate time frames :*
- 4 3.1. *Find ASAP schedule*
- 5 3.2. *Find ALAP schedule*
- 6 *Calculate self, predecessor and successor forces for the nodes*
- 7 *Add them to get the total force for each feasible control step*
- 8 *Determine the best node and step; set its time frame equal*
- 9 *to the selected step.*

10 *Invalidate the stored sub_DGs on the path from the scheduled*
11 *node to the root*
12 *Update the overall DG*
13 **until** *all operations scheduled*

2.4 Experimental results for hierarchical graphs

The hierarchical force directed scheduling scheme was implemented in a tool named *HIERFDS*. The following table gives the number of resources required by the schedule generated by *HIERFDS* for different benchmark circuits. The optimum number of each resource is also shown. It can be seen that the hierarchical force directed scheduling algorithm is able to arrive at almost the optimum number of resources in most cases.

Table 2.1 Comparison of hierarchical force directed scheduling with the optimal results

circuit	latency	Adders(optimal)	Mults(optimal)
kalman	3700	1(1)	2(2)
elliptic	17	2(2)	-
elliptic	27	2(1)	-
synth1	32	2(2)	4(4)
synth1	80	1(1)	2(1)
synth2	32	1(1)	2(2)
synth2	60	1(1)	1(1)

2.5 Summary

In this chapter, we presented a novel algorithm for scheduling hierarchical signal flow graphs which naturally handles loops and conditionals. It schedules the nodes in the hierarchical signal flow graph level by level, and automatically selects the latencies for loops and conditionals. The new algorithm was implemented in a tool *HIERFDS*. Experimental results indicate that the algorithm is quite effective in arriving at close to optimal results for a set of benchmark circuits.

CHAPTER 3

PARALLEL ALGORITHMS FOR FORCE DIRECTED SCHEDULING

3.1 Introduction

Force directed scheduling has been shown to be a very effective scheduling algorithm in that it is able to arrive at close to minimum number of resources for the problem of scheduling under a latency bound. With complex chips that need to be designed in the future, it is expected that the run times of scheduling algorithms will be quite large.

In this chapter, three novel parallel algorithms for the scheduling problem based on force directed scheduling are presented. The first two are based on force directed scheduling for non-hierarchical graphs, and the third one is based on the force directed scheduling for hierarchical graphs described in the previous chapter. The parallel algorithms are portable across a wide range of parallel platforms which includes shared memory multiprocessors distributed memory message-passing multicomputers and networks of workstations. Experimental results are reported on an 8 processor SGI Origin shared memory multiprocessor and an 80 processor IBM SP-2 distributed memory multicomputer. The parallel algorithms can attain speedups of up to 7.68 on 8 processors on IBM SP-2 and up to 7.31 on 8 processors on SGI Origin, compared to the sequential versions. In addition, while some parallel algorithms for VLSI CAD reported by earlier researchers have reported on loss of qualities of results, our parallel algorithms produce exactly the same results as the sequential algorithms on which they are based.

In Section 3.2 two approaches to implementing parallel force directed scheduling algorithm for non-hierarchical graphs are presented. In Section 3.3, a parallel implementation of the hierarchical force directed scheduling is described. In Section 3.4 an analysis of the algorithm is presented. In Section 3.5 the experimental results are presented, and Section 3.6 provides the summary.

3.2 Parallel force directed scheduling algorithm for non-hierarchical graphs

In this section, we will look at the force directed scheduling problem of simple acyclic graphs, corresponding to straight line code sequences in a behavioral description. We describe algorithms for graphs with conditionals and loops in the following section. For parallel implementation of the basic force directed scheduling, we have taken two problem partitioning approaches, node-based and step-based. In the node based method, we partition the nodes among the processors, and in step-based method, we partition the time steps among the processors.

3.2.1 Node based problem partitioning

The nodes are partitioned among the processors. The partitioning is done by first calculating the total available work, which is approximated by

$$W = \sum_{i=0}^{numnodes} (ALAP(i) - ASAP(i) + 1)$$

and work per processor, given by $W/numprocs$. Each processor selects a set of nodes S such that $\sum_{s \in S} (ALAP(s) - ASAP(s))$ is approximately equal to the work per processor. This partitioning is performed using a greedy algorithm in which processor 0 selects the first k_0 nodes such that $\sum_{i=0}^{k_0-1} (ALAP(i) - ASAP(i) + 1) \geq W/numprocs$, processor 1 selects the next k_1 nodes such that $\sum_{i=k_0}^{k_0+k_1-1} (ALAP(i) - ASAP(i) + 1) \geq W/numprocs$ and so on.

We can define a *local_DG* for each processor which is the *DG* calculated using only the nodes owned by that processor. The *global_DG* is obtained by combining the *local_DGs* of all the processors.

At each step, each processor calculates the *local_DG*. The processors then perform a global reduction operation by which the *local_DGs* are combined together to get the *global_DG*. Then, each processor calculates the forces of each owned node to each feasible time step, and sends the information to a *master* processor. The *master* determines the node with the least force and schedules it to the step with the least force. It then broadcasts that information to all the processors. Each processor, upon receiving this information, updates the time frames of other nodes. Since the time frames of the nodes could change as a result of the schedule operation, the load function is re-computed, and nodes re-assigned to processors, to minimize load imbalance. The processors then continue to the next iteration until all nodes are scheduled.

The node based parallel force directed scheduling algorithm is as follows:

PARALLEL-FORCE-DIRECTED-NODE()

- 1 *Compute the work per processor*
- 2 *Determine the list of nodes on current processor*
- 3 **repeat**
- 4 *Evaluate time frames :*
- 5 4.1. *Find ASAP schedule*
- 6 4.2. *Find ALAP schedule*
- 7 *Compute local distribution graph for the assigned nodes*
- 8 *Per form global reduction to get the overall distribution graph*
- 9 *Calculate self, predecessor and successor forces for the assigned nodes,*
- 10 *Add them to get the total force for each feasible control step*
- 11 *Send the operation with lowest force to the master*
- 12 *Receive the global best node and step;*
- 13 *Set its time frame equal to the selected step.*
- 14 **until** *all operations scheduled*

calculate the DG s of the steps in T . The processors then perform a global reduction operation by which the DG s computed by the processors are combined together to get the overall DG . Each processor then calculates the forces of each node in S to each time step in T it intersects. It then sends the information to a *master* processor which selects the least force node. The master then schedules that node to the step with the minimum force. It then broadcasts the information. All other processors, on receiving the information, updates the time frames of other nodes and continues to the next iteration. The steps are reassigned to processors after each iteration so that they get approximately equal amount of work.

The step based parallel force directed scheduling algorithm is as follows:

PARALLEL-FORCE-DIRECTED-STEP()

- 1 **repeat**
- 2 *Compute work per processor*
- 3 *Determine the list of steps for the current processor*
- 4 *Evaluate time frames :*
- 5 4.1.*Find ASAP schedule*
- 6 4.2.*Find ALAP schedule*
- 7 *Compute distribution graph for the assigned steps*
- 8 *Per form global reduction to get the overall distribution graph*
- 9 *Calculate self, predecessor and successor forces for the*
- 10 *nodes which intersect the assigned steps,*
- 11 *Add them to get the total force for each feasible control step*
- 12 *Send the operation with lowest force to the master*
- 13 *Receive the global best node and step; set its time frame*
- 14 *equal to the selected step.*
- 15 **until** *all operations scheduled*

Figure 3.2 shows an example showing an non-hierarchical graph and a sample step-based decomposition among four processors. We show experimental results of this parallel algorithm in Section 3.5.

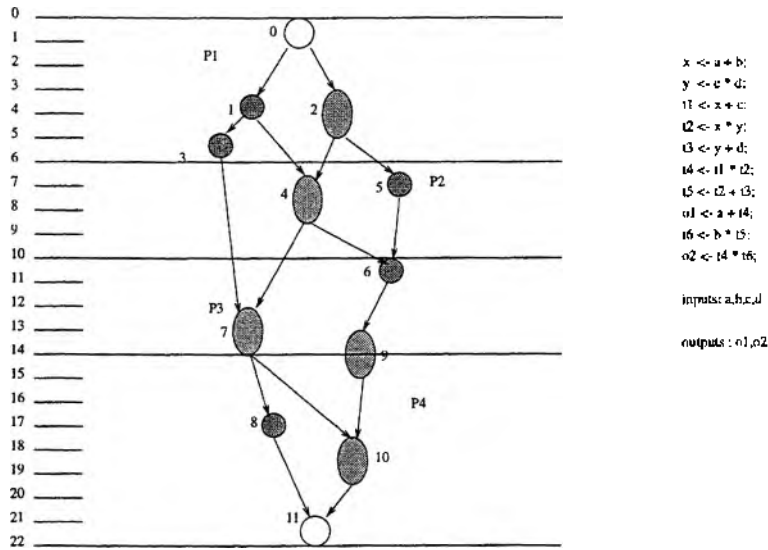


Figure 3.2 Step-based decomposition for a non-hierarchical graph for 4 processors

3.3 Parallel hierarchical force directed scheduling

In this section, we describe a parallel algorithm for the hierarchical scheduling approach explained in chapter 2. As in the sequential case, the scheduling is done one level at a time starting from level 0, scheduling all nodes at a particular level before proceeding to the next. The processors are assigned to the hierarchical nodes based on a cost function (specified in the next section) based on the number of nodes enclosed by the node. Call the set of processors assigned to a node n the *process group* of n . For nodes enclosed by a hierarchical node n , processors in the process group of n are assigned based on the cost function. For nodes at the lowest level (the atomic nodes), the processors in the process group of the immediately enclosing hierarchical node are assigned cyclically.

Figure 3.3 shows the syntax tree of example hierarchical signal flow graph of Figure 2.2 in the previous chapter, the levels of various nodes and processor assignments.

For the graph in Figure 3.3, the process group of node 9 is $\{0,1\}$. As in the case of the sequential algorithm, we store *sub_DGs* in a table and reuse them as long as they are valid. However, since the *sub_DGs* calculated by a particular processor are stored in its local memory, it is not immediately accessible to other processors.

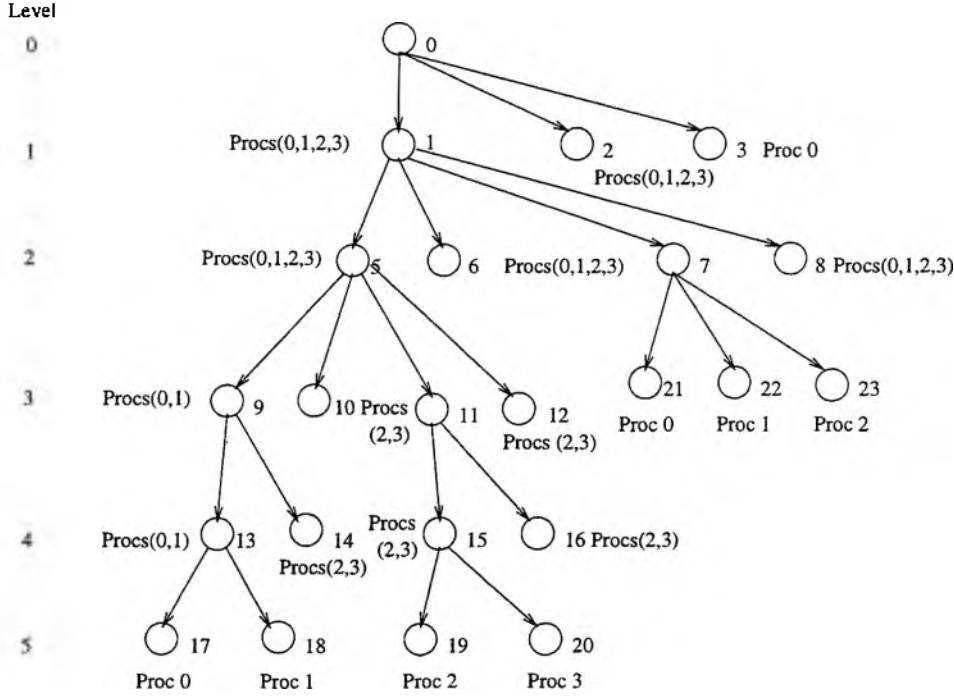


Figure 3.3 Syntax tree of example hierarchical graph with processor assignments

The processors in the process group of each hierarchical node collectively calculates the *sub_DGs* of that node. It is done in a bottom up manner, since calculation of *sub_DG* for a hierarchical entity requires the *sub_DGs* of the nodes it enclose. The processors calculating the *sub_DGs* of nodes at a particular level communicates them with the processors in the process group of the hierarchical node enclosing those nodes by a group-level *allgather* operation. Once this process completes with the calculation of *sub_DGs* of the root node, all processors have, in their local memories all the *sub_DGs* needed to calculate forces for the nodes assigned to them. This is due to the fact that scheduling is done in a top down manner. At a particular level l , each processor needs only the *sub_DGs* of the nodes enclosed by the hierarchical node immediately enclosing it (which is at level $l - 1$). Since all nodes at level $l - 1$ has been already scheduled, changes in time frames of the children of a particular node h in level $l - 1$ does not affect the time-frames of the children of another node h' at level $l - 1$, and hence does not have to be included in the list of predecessors or successors used in calculating the *ps* force (Equations 2.2, 2.3) for a child of h . Thus, a processor calculating the force for a child of h needs only the *sub_DGs*

of other children of h , which have been calculated and distributed among the processors in the process group of h .

Consider for example node 15 in Figure 2.2. When its force is calculated, node 11 has already been scheduled, since it is at a lower level. Thus, scheduling node 15 does not affect the time-frames of nodes 13 or 14, and hence those nodes does not contribute to the predecessor force of 15. Only the *sub_DG* of node 16 is required which was calculated earlier by processors 2 and 3. Only those processors assigned to a particular hierarchical node performs the calculation of forces and *sub_DGs* for the nodes enclosed by the hierarchical node. One advantage of this static division of work is that we can exploit locality in reusing the *sub_DGs* stored in each processor's local memory.

For each hierarchical entity, there is possibly more than one assigned processor. One of them is assigned ownership for that node. The role of the owner of a node is to update the *sub_DG* of the node during scheduling when one of the nodes enclosed by the node is scheduled. Since the *sub_DGs* of all nodes at a level are known, the calculation of forces at any given level closely resembles that in the case of the basic force directed scheduling algorithm. The parallel algorithm has three major steps:

- Node partitioning step
- The pre-processing step, and
- The force calculation and scheduling step

3.3.1 Node partitioning step

In the node partitioning step, the nodes are distributed among the processors. Associated with each hierarchical entity is a *height*, which denotes the depth of the syntax tree corresponding to the nodes in that entity. Consider for example, the nodes in level 0. It contains hierarchical entities having possibly different heights. Consider the set of nodes with a given height (say h). The processors are partitioned across all the nodes in that set so that the number of processors assigned to node i is proportional to a cost function which is an approximation of the work in-

volved for that node, given by :

$$C(i) = ALAP(i) - ASAP(i) + 1 + \sum_{j \in N(i)} C(j)$$

if i is a hierarchical node and

$$C(i) = ALAP(i) - ASAP(i) + 1$$

otherwise. where $N(i)$ is the set of nodes enclosed by i . A particular processor among the processors assigned to each hierarchical node is assigned ownership for that node. In level 1, the processors assigned to the parent hierarchical entity are distributed among the nodes contained in that entity. In the next level, the processors assigned to the nodes in that level are distributed among the nodes contained in them and so on until we reach the leaves.

3.3.2 Pre-processing step

In the pre-processing step, the *sub_DGs* for each hierarchical entity for all feasible latencies for that entity that arise during the force calculation and scheduling step are calculated and stored. This is achieved by setting the latency of the immediately enclosing hierarchical entity to its maximum value within the limits of its *ASAP* and *ALAP* schedule, (since that results in the maximum possible flexibility for the nodes within). The *sub_DGs* are calculated level by level, starting at the deepest. As the computation proceeds from one level to the next, the *sub_DGs* of each node in the current level for the feasible latencies calculated in the current step are distributed to all processors in the process group of the node immediately enclosing the node in the hierarchy, by an allgather operation. Once the pre-processing step is complete, all the processors have the *sub_DGs* required for calculating the forces.

3.3.3 Force calculation and scheduling

After the pre-processing step, the processors calculate the forces of the nodes to each feasible step and sends the best node to a master processor. The master selects the node with the

least force and schedules it to the corresponding step. It then broadcasts the information to all processors. Due to this scheduling, the pre-calculated *sub_DG* of the hierarchical nodes enclosing the scheduled node (and all hierarchical nodes in the path from the node to the root node in the hierarchy) are no longer valid. Thus, they are recalculated. The *sub_DG* of a node is recalculated by the processor owning the node and sent to the processor owning the parent of the node, starting at the level in which the nodes are being scheduled currently. Since the scheduling proceeds from the lowest to the highest level, all the control nodes enclosing the scheduled node have been scheduled, and the new *sub_DG* doesn't have to be distributed to all processors in the group, since the *sub_DGs* are needed only for force calculation. However, the *sub_DG* of the root node (which is the overall *DG* of the graph) is required for calculation of forces and hence is broadcast to all processors by the owner processor of the root node. Consider for example the situation when node 15 is scheduled. The *sub_DGs* of nodes 11,5,1 and 0 become invalid. Processor 2, the owner of node 11 recalculates the *sub_DG* of 11 and sends it to processor 0. Processor 0 receives that and recalculates the *sub_DGs* of nodes 5,1 and 0 since it owns all those nodes. Once processor 0 calculates the *sub_DG* of node 0 (which is the overall *DG* of the graph), it broadcasts it to other processors.

3.3.4 An example

We consider the hierarchical graph in Figure 2.2 in the previous chapter, whose syntax tree, with processor assignments is shown in Figure 3.3. At level 0, it has a conditional node (node 1) and an add operation (node 3). The conditional operation has two alternatives whose sources are nodes 5 and 7 respectively. Nodes 5 and 7 are at level 1. The second alternative has a multiply and two add operations, which are at level 2. Assuming that we have 4 processors, they are assigned to the various nodes as shown in Figure 3.3. Assume that we have to schedule the graph for latency 10. Assume, for simplicity that each loop executes once and assume that add operation has delay 1 and multiply operation has delay 2. Since node 3 has a delay of 1, the conditional can have latency up to 9 for the overall schedule to have latency 10. The minimum latency of the conditional is given by the maximum of the minimum latencies of its alternates.

In the example, the second conditional has a minimum latency 3 (which is the sum of the delays of one multiply operation and one add operation). The first alternate has minimum latency 2, since both the loops have minimum latency 1 (based on the simplifying assumption that each loop executes once). Hence, the minimum latency of the conditional is 3. Thus, the conditional statement can have latencies ranging from 3 to 9. The first alternative can have latencies ranging from 2 to 9, and the second, from 3 to 9.

Consider the node 9. In the preprocessing step, processors 0 and 1 recursively calculate the *sub_DGs* for node 9 for latencies 1 to 4, and processors 2 and 3 calculate the *sub_DGs* for node 11 for latencies 1 to 4. After that, processors 0,1,2, and 3 perform an allgather operation, after which processors 0,1,2 and 3 have the *sub_DGs* for nodes 9 and 11 for feasible latencies. However, processors 0 and 1 does not have the *sub_DGs* of nodes 15 or 16. But, they are not required for the *DG* calculation for node 0 or for the calculation of forces for nodes 9 and 10. In the next step, processors 0,1,2 and 3 calculate and store the *sub_DGs* for nodes 5,6,7 and 8 after which they perform an allgather operation which results in processors 0,1,2 and 3 having the *sub_DGs* for feasible latencies for nodes 5,6,7 and 8.

Consider now the force calculation phase. Processors 0,1,2 and 3 calculate the forces for nodes 1,2 and 3 and schedules them. Assume that node 2 is scheduled first. The *sub_DG* of node 0 is invalidated and should be recalculated. However, the *sub_DG* of node 1 can still be reused, since the *sub_DG* of a hierarchical node depends only on the latency of the node, provided that none of the interior nodes are scheduled. After 1,2 and 3 are scheduled, processors 0,1,2 and 3 calculate the forces for nodes 5,6,7 and 8 and schedules them. After they are scheduled, processors 0 and 1 calculate the forces for nodes 9 and 10, processors 2 and 3 calculate the forces of nodes 11 and 12, and processors 0,1 and 2 calculates the forces for nodes 21, 22 and 23 respectively. Assume that node 9 is scheduled at this point. Now the *sub_DGs* of the nodes in the path from node 9 to the root (node 0), i.e., nodes 9,5,1 and 0 are invalidated, and they are recomputed by the processors owning those nodes, as explained before. Once nodes 9,10,11,12,21,22 and 23 in level 3 have been scheduled, the nodes in level 4 are considered and so on till all nodes are scheduled.

3.3.5 Load balancing

As explained above, each hierarchical entity has a list of processors assigned to it in the first step. Due to the way the *sub_DGs* were calculated in the pre-processing step, each of the processor associated with a hierarchical node has the *sub_DGs* for all feasible latencies of the component nodes of the hierarchical entity in their local memories, and hence can calculate the forces of any of those nodes. The force calculation of a particular node to a particular step represents a unit of work. The total work involved in calculating the forces of the component nodes of a hierarchical entity is distributed among the processors in the process group of that entity. This distribution is done cyclically among the processors with the grain size ranging from 1 to $total_work/(nprocs)$. In the figure, during the force calculation for nodes 5,6,7 and 8, the $\langle node, step \rangle$ pairs are cyclically distributed among processors 0,1,2 and 3.

3.4 Analysis

3.4.1 Pre-processing step

Assume that the given hierarchical graph has N nodes and has depth d . Let L be the overall latency. During pre-processing, the *sub_DGs* of all nodes are calculated for all feasible latencies. The computation required for the pre-processing step for a particular level t for each hierarchical node at that level is at most $c_1 n_{t+1} L^2$ where n_{t+1} is the number of nodes contained within the hierarchical node at level $t+1$ and c_1 is some constant. Thus, the overall computation required for the pre-processing stage is $O(NL^2)$. Assuming good load balancing, the computation time for the pre-processing for the parallel algorithm is $O(\frac{NL^2}{p})$.

The time for the allgather operation, performed for each level of the hierarchical graph during the pre-processing, depends on the algorithm used for implementing that operation, the topology of the machine, etc. For example, it can be implemented in $2t_s(\sqrt{p}-1) + t_w m(p-1)$ time for a 2-D mesh with store and forward routing [41], where t_s is the start-up time, t_w is the per-word transfer time, and m is the size of the message. An upper bound on the size of the message during pre-processing is $O(NL^2)$.

3.4.2 The force-calculation and scheduling phase

In this phase, after each scheduling step, each processor calculates the forces, sends its best node along with the force to a master processor, and the master processor selects the overall best node and broadcasts it to all processors.

There are at most N schedule steps, since at a given step, more than one node could be scheduled. Each force calculation step takes $O(LN^2/p)$ time assuming good load balance. Thus, the force calculation for all steps takes $O(LN^3/p)$ since there are at most N steps.

The communication time again depends on the algorithm used for the reduction and broadcast and also on the topology of the machine. For example, for a 2-D mesh with store and forward routing, each of the above can be performed in time $2(t_s + t_w m_1) \lceil \frac{\sqrt{p}}{2} \rceil$, where m_1 is the length of the messages which is a constant (since the message contains only the node with the least force, the corresponding time step, and the force) [41]. The total communication time, in that case, over all steps is $4N(t_s + t_w m_1) \lceil \frac{\sqrt{p}}{2} \rceil$.

3.5 Experimental results

The parallel force directed scheduling algorithm for non-hierarchical graphs was implemented in a tool named *PARFDS* and the parallel force directed scheduling for hierarchical graphs was implemented in a tool named *PARHIERFDS* using the portable Message Passing Interface (*MPI*) [42], which has been ported on a variety of parallel machines. We have tested the algorithms on a set of high level synthesis benchmarks. The resulting schedules were optimal for most cases and very close to optimal for others. The run times for some benchmarks for different latencies are as shown below. The results are shown for an IBM SP-2 distributed memory multicomputer, and an *SGI* Origin shared memory multiprocessor.

3.5.1 Experimental results for non-hierarchical graphs

Tables 3.1 and 3.2 shows the run times and speedups for node-based approach for non-hierarchical graphs and Tables 3.3 and 3.4 show the run times and speedups for step-based ap-

Table 3.1 Run times(seconds) and speedups for the *PARFDS* tool using node-based approach for non-hierarchical graphs on an SGI Origin multiprocessor

		time(speedup)			
circuit	latency	1 proc	2 proc	4 proc	8 proc
Random(60)	60	22.03(1.0)	11.20(1.97)	5.69(3.87)	3.09(7.13)
Random(60)	80	30.81(1.0)	15.62(1.97)	7.92(3.89)	4.21(7.31)
elliptic	17	0.35(1.0)	0.19(1.84)	0.125(2.80)	0.085(4.12)
elliptic	27	0.658(1.0)	0.603(1.09)	0.315(2.08)	0.197(3.34)

Table 3.2 Run times (seconds) and speedups for the *PARFDS* tool using node-based approach for non-hierarchical graphs on IBM SP-2 multicomputer

		time(speedup)			
circuit	latency	1 proc	2 proc	4 proc	8 proc
Random(60)	60	56.80(1.0)	28.91(1.97)	14.65(3.88)	7.58(7.49)
Random(60)	80	79.58(1.0)	40.18(1.98)	20.31(3.92)	10.36(7.68)
elliptic	17	0.938(1.0)	0.525(1.79)	0.339(2.77)	0.203(4.62)
elliptic	27	1.78(1.0)	1.64(1.08)	0.859(2.07)	0.468(3.79)

proach for non-hierarchical graphs. The qualities of the results (in terms of the hardware resources used) for each latency case are identical in the sequential and parallel executions, since the parallel algorithms computes the same results as the sequential algorithm. From the results we can make the following observations. If the latency is close to the minimum, then the time frames of the nodes are small, in which case the total work involved in calculating the forces is relatively small. This results in low overall running time and poor speedup. For the node based partitioning scheme, we are getting speedups of 3.79 to 7.68 on 8 processors on an IBM SP-2, and 3.34 to 7.31 on 8 processors on SGI Origin. Clearly, the problem has been parallelized such that the grain size of computation between communications is quite large. Hence, we have effectively parallelized the problem.

Both partitioning schemes are comparable in their run times and speedups for a given parallel environment. Clearly we have been successful in balancing the load equally in both cases.

Table 3.3 Run times(seconds) and speedups for the *PARFDS* tool using step-based approach for non-hierarchical graphs on an SGI Origin multiprocessor

circuit	latency	time(speedup)			
		1 proc	2 proc	4 proc	8 proc
Random(60)	60	22.02(1.0)	12.07(1.82)	6.30(3.50)	3.62(6.08)
Random(60)	80	30.81(1.0)	16.2(1.90)	8.41(3.66)	4.71(6.55)
elliptic	17	0.35(1.0)	0.228(1.53)	0.147(2.38)	0.123(2.84)
elliptic	27	0.658(1.0)	0.671(0.98)	0.369(1.78)	0.254(2.59)

Table 3.4 Run times for step-based approach for the *PARFDS* tool for non-hierarchical graphs on IBM SP-2 multicomputer

circuit	latency	time(speedup)			
		1 proc	2 proc	4 proc	8 proc
Random(60)	60	56.80(1.0)	31.65(1.79)	16.55(3.43)	9.20(6.18)
Random(60)	80	79.58(1.0)	42.27(1.88)	22.07(3.61)	11.83(6.73)
elliptic	17	0.938(1.0)	0.731(1.28)	0.435(2.16)	0.311(3.01)
elliptic	27	1.78(1.0)	1.85(0.96)	1.03(1.73)	0.637(2.79)

3.5.2 Experimental results for hierarchical graphs

Tables 3.5 and 3.6 show the run times and speedups for hierarchical graphs. For the parallel algorithms for hierarchical graphs, the speedup results are similar to that of non-hierarchical graphs. Again the qualities of the results (in terms of the hardware resources used) for each latency case are identical in the sequential and parallel executions. As in the case of non-hierarchical graphs, if the latency is close to the minimum, then the time frames of the nodes are small, resulting in low overall running time and poor speedup. For Kalman, there are only few nodes in each level which results in low parallelism. The higher latencies in the above table correspond to the latencies which results in close to the minimum resource requirement for the corresponding circuit.

Table 3.5 Run time(seconds) and speedup for the *PARHIERFDS* tool for hierarchical graphs on IBM SP-2 message passing multicomputer

Run time(speedup)					
circuit	latency	1 proc	2 proc	4 proc	8 proc
kalman	3700	17.56(1.0)	13.89(1.26)	7.01(2.5)	4.21(4.17)
elliptic	17	1.19(1.0)	0.514(2.31)	0.349(3.4)	0.222(5.36)
elliptic	27	3.25(1.0)	1.58(2.05)	0.826(3.93)	0.475(6.84)
synth	100	15.05(1.0)	10.34(1.46)	6.41(2.35)	4.53(3.32)
synth	200	64.33(1.0)	36.50(1.76)	27.62(2.33)	16.10(4.00)

Table 3.6 Run time(seconds) and speedups for the *PARHIERFDS* tool for hierarchical graphs on SGI Origin multiprocessor

time(speedup)					
circuit	latency	1 proc	2 proc	4 proc	8 proc
kalman	3700	8.45(1.0)	6.25(1.35)	3.00(2.81)	2.57(3.29)
elliptic	17	0.365(1.0)	0.197(1.85)	0.130(2.80)	0.205(1.78)
elliptic	27	1.211(1.0)	0.619(1.96)	0.326(3.72)	0.281(4.30)
synth	100	5.676(1.0)	3.468(1.64)	2.141(2.65)	1.641(3.46)
synth	200	27.164(1.0)	14.802(1.84)	10.86(2.50)	6.274(4.33)

3.6 Summary

In this chapter, we presented some novel parallel algorithms for scheduling signal flow graphs. The parallel algorithms are portable across a wide range of parallel platforms. We reported results on an 8-processor SGI Origin and an 80 processor IBM SP-2. While some parallel algorithms for VLSI CAD reported by earlier researchers have reported on loss of qualities of results, the parallel algorithms presented in this chapter produced exactly the same results as the sequential algorithms on which they are based.

CHAPTER 4

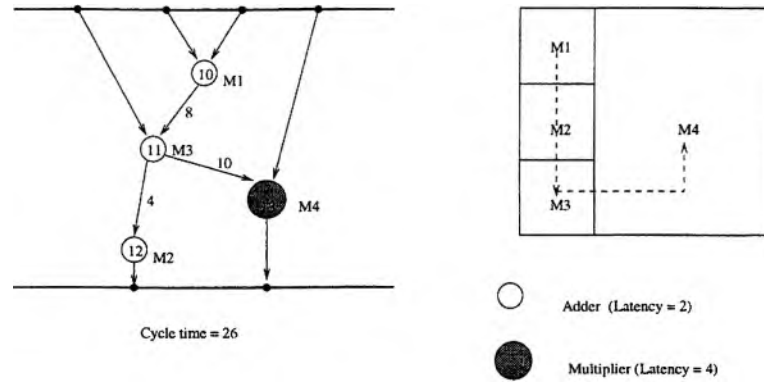
SIMULTANEOUS SCHEDULING, BINDING AND FLOORPLANNING IN HIGH-LEVEL SYNTHESIS FOR LATENCY MINIMIZATION

4.1 Introduction

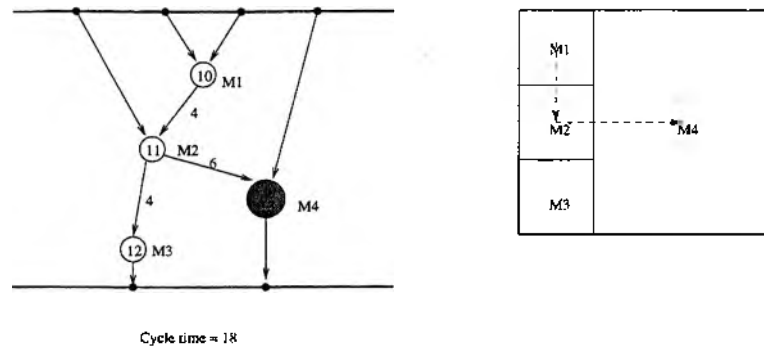
In this chapter, a simultaneous scheduling, binding and floorplanning algorithm is presented. In comparison to a traditional approach which separates high-level synthesis from physical design, our algorithm is able to make these stages interact very closely, resulting in solutions with lower latency and area. In addition, a detailed model is considered, taking into account multiplexor and register areas and delays.

The latency of a schedule depends on the cycle time and the number of cycles in the schedule. The cycle time is determined by the longest path in a schedule step, which includes the functional unit delays, multiplexor delays, register delays and interconnect delays. The number of cycles is determined by the schedule. A smaller number of cycles might require some operations to be chained together in a cycle, for example, but it could increase the cycle time. The area of the floorplan includes the areas of functional units, registers, multiplexors, control and wiring.

With sub-micron technologies, interconnect delay between modules is increasingly becoming a major part of the cycle time. Interconnect delays strongly depend on the number and sizes



(a) Binding operation 10 to M1, operation 11 to M3 and operation 13 to M4



(b) Binding operation 10 to M1, operation 11 to M2 and operation 13 to M4

Figure 4.1 Effect of considering floorplanning during scheduling and binding

of modules used and their relative positions. The area of the generated floorplan is determined by the number of modules used, and their positioning, which is partly decided by how small the overall latency should be. Thus, to minimize the execution time, it is important to consider the scheduling and binding information during floorplanning. Similarly, during scheduling and binding, the effect on floorplanning should be considered.

Figure 4.1 shows an example of how scheduling and binding without considering floorplanning can increase the latency of a schedule. It can be seen in Figure 4.1(a) that binding operation 10 to M_1 and 11 to M_3 results in a cycle delay of 26, since modules (M_1, M_3) and (M_3, M_4) are farther away in the floorplan, whereas binding 10 to M_1 and 11 to M_2 as in Figure 4.1(b) results in a cycle time of 18 since modules (M_1, M_2) and (M_2, M_4) are closer in the floorplan.

We first present an efficient floorplanning algorithm which attempts to minimize the interconnect delays and also the overall area of the floorplan. Following that, we present a simulated annealing algorithm which simultaneously optimizes the latency of a schedule and the overall area of the floorplan. Routing area is not considered in this work. We compare the results with Fang and Wong [15] algorithm and also with that of a two step algorithm which performs a simulated annealing based combined scheduling and allocation in the first step followed by a floorplanning algorithm in the second step.

In contrast to [17], we take into account the latencies of all paths simultaneously during floorplanning, also attempting to minimize the area of the layout at the same time. In contrast to [15], we perform a fast constructive timing-driven floorplanning after each step of a simulated annealing algorithm which performs combined scheduling and allocation. This approach permits us to incorporate multiplexors in the floorplan which is not easy in the case of [15]. In addition, our algorithm can alter the schedule, which adds another degree of freedom to optimize interconnect delays, latency and area.

In Section 4.2, the simulated annealing method is summarized. In Section 4.3 a constructive timing-driven floorplanning algorithm is presented. In Section 4.4, the algorithm for combined scheduling, binding and floorplanning is described. Section 4.5 gives an overview of the two-step algorithm used for comparison. In Section 4.6, the results are presented for a set of high-level synthesis benchmarks.

4.2 Simulated annealing

Simulated annealing is a hill-climbing optimization algorithm which is theoretically capable of arriving at a globally optimum solution. It is analogous to the physical process of crystal formation. The basic methodology is to perform a sequence of moves from a starting solution, which are essentially perturbations from the solution. It starts at a very high 'temperature', and reduces the temperature gradually according to a cooling schedule. At each temperature, a number of moves are performed. The number of moves have to be selected such that the process reaches an equilibrium at each temperature. However, since that will require an unreasonably

large number of moves, in practice, the number of moves per temperature is set to a fixed constant. The probability that a move is accepted is a function of the temperature given by $e^{-\frac{\Delta C}{T}}$ where ΔC is the change in cost as a result of the move, and T is the temperature. Thus, at high temperature, a move could be accepted with non-zero probability even if it will result in an increased cost. These 'hill-climbing' moves enable the annealing process to escape from being trapped in local minima. The initial temperature is selected so that most of the moves are accepted. Towards the end of the annealing process, when the temperature is low, only moves that will strictly reduce the cost are accepted.

4.3 The floorplanning algorithm

The constructive timing driven floorplanning algorithm considers criticality of interconnections between modules in addition to the overall area of the floorplan. Consider nodes a and b in Figure 4.3(a) for example. Since it is the critical path, the delay of interconnections between the module pairs (r_1, m_1) , (m_1, m_4) and (m_4, r_3) directly affects the overall cycle time of the schedule. Thus, it is desirable to have them as close as possible to each other in the floorplan. However, since the interconnection between modules m_3 and r_2 is not critical, they can be farther away in the floorplan without possibly affecting the cycle time. Our algorithm tries to use this key intuition formally.

The algorithm has two main phases. In the first phase, the slacks of connections between modules are calculated from the schedule. In the second phase, modules are clustered using a greedy clustering algorithm in a bottom-up manner, based on the slacks calculated in the previous phase. Along with the clustering, the bounding curve of each cluster is determined in a bottom up manner using a new curve-addition technique, and the minimum cost point is determined from that. This point determines the slicing structure of the floorplan generated by the algorithm.

In the final phase, the slicing structure is constructed in a top down manner based on the bounding curves computed in the previous phase. This results in a set of modules, at each level of the slicing structure, each of which is either atomic (a leaf node of the slicing tree) or com-

pound (a non-leaf node in the slicing tree). These modules are then reoriented in a bottom-up manner in order to further reduce the interconnect delay, employing a technique similar to one proposed in [23], [14]. The various phases are explained in detail in the following sections.

4.3.1 Slack calculation

The cycle time consists of the worst case module delays, the multiplexor and register delays, the interconnect delays between the modules, and the delay in the control unit for any step. In this paper, we do not consider the control unit delay.

An extended signal flow graph [15] is constructed for each step of the schedule. For a given step, at most one operation of the signal flow graph can be assigned to a functional unit. Each node in the extended *SDFG* is a functional unit, a register or a multiplexor. There is an edge between two nodes in the extended *SDFG* if there is a data flow between the operations assigned to the modules corresponding to those nodes. For example, consider the *SDFG* in Figure 4.3. The operation b of the signal flow graph is assigned to module m_4 . One of the operands for b is computed in the previous cycle, and stored in register r_3 . The other operand is computed in the same cycle by m_1 . Assume that m_4 requires a mux (mux_1) at the second input. Then, there are edges in the *SDFG* from r_3 to mux_1 , mux_1 and m_1 to m_4 , and m_4 to r_3 .

The nodes of the extended *SDFG* which do not have a predecessor node in that step are defined to be the *lower boundary* nodes of the step, and those which do not have any successors are defined to be the *upper boundary* nodes. For example, registers which store the results from the operation in a previous cycle which are input to operations in the current cycle are *lower boundary* nodes, where as those which store the results computed by the current cycle and used in subsequent cycles are *upper boundary* nodes. In Figure 4.3 of the example signal flow graph, r_1 and r_2 are lower boundary nodes, and r_3 is an upper boundary node in step 2.

The longest delay T for any step is calculated, assuming zero interconnect latency. Based on this, the slacks of each arc in the extended signal flow graph are calculated. It is the longest delay that can be assigned to that arc without affecting the cycle time. Thus, the arcs of the critical

paths will be assigned a slack value of zero. The calculation of edge slacks is performed in a manner similar to that proposed in [15] as follows.

Let T be the longest delay for a time step assuming zero interconnect delays. Consider the nodes in a given time step S . Let $ES(i)$ and $LS(i)$ denote the earliest start time and latest start time for node i in S of the extended $SDFG$. $ES(i)$ is determined by performing an *ASAP* (as soon as possible) schedule of the nodes in S , assuming that the *lower boundary* nodes start at time 0. By *ASAP*, the starting time of a node is set to the earliest time during which its source operands are available from the predecessors.

$$\begin{aligned} ASAP(n) &= 0 \text{ if } n \text{ is a lower boundary node,} \\ &= \max_{p \in \text{preds}(n)} (ASAP(p) + \text{latency}(p)) \text{ otherwise} \end{aligned}$$

$LS(n)$ is determined by performing an *ALAP* (as late as possible) schedule of the nodes in S , assuming that the *upper boundary* nodes finish at time T .

$$\begin{aligned} ALAP(n) &= T - \text{latency}(n) \text{ if } n \text{ is an upper boundary node,} \\ &= \min_{s \in \text{succs}(n)} (ALAP(s) - \text{latency}(n)) \text{ otherwise} \end{aligned}$$

The slack of edge e_{ij} from module i to module j is calculated as:

$$\text{Slack}(i, j) = LS(j) - ES(i) - \text{latency}(i)$$

Once the edge slacks are calculated, we assign a delay value to each edge so that the slack is reduced to zero. The edges in the critical path are assigned a delay of zero, for example. These delays can be regarded as a measure of urgency which indicates how close the pair of modules should be in the floorplan. The lower the delay, the closer the modules should be, to minimize the effect of interconnect delays on the overall latency of the schedule. It can be seen that assigning a delay value to an arc will cause the slack values of other arcs to change. In order to treat each arc independent of others, we distribute the slacks among the edges in a path (similar to *zero slack algorithm* [29] for standard cell placement). The distribution of slacks to calculate edge delays is done as follows: Figure 4.3 shows the slack values for edges for the example graph, and the delay values assigned by distributing the slacks. Initially, we start

DISTRIBUTESLACKS()

- 1 Start with all edges having delay value zero
- 2 **repeat**
- 3 Select the edge e with the least non-zero slack s
- 4 Find the longest path P which includes e such that all the edges in P have slack $\geq s$.
- 5 Let k be the length of the path
- 6 Add delay s/k to the delay value of each edge in P
- 7 Recompute the slacks of all edges, using the new delay values
- 8 **until** until all edges have slack zero

Figure 4.2 Slack distribution algorithm

with the assumption that all edges have zero delays. Hence, step 1 has latency 9, and step 2 has latency 12 and the cycle time of the schedule is $\max(12, 9) = 12$. Consider the edge between m_3 and r_2 . The earliest start time of c , $ES(c)$, the operation assigned to m_3 in step 1, is 1, which is the earliest time during which the contents of register r_0 is available at the beginning of the cycle. The latest start time of register write to r_2 is 11, without violating the cycle delay of 12. Thus, the slack of the edge from m_3 to r_2 is $(11 - 1 - 5) = 5$. Similarly, the slack of edge between r_0 and m_3 is 5. While distributing the slacks, the edge with the lowest non-zero slack in step 1 is (r_0, m_3) with slack value 5. The longest path in step 1, with slack ≥ 5 , and including edge (r_0, m_3) is (r_0, m_3, r_2) . This has two edges. Thus, we assign a delay of $5/2 = 2.5$ each to (r_0, m_3) and (m_3, r_2) . This makes the slacks of both the above edges zero. The delay values of other edges are derived in a similar way, and are as shown in Figure 4.3(b).

These delay values can be considered as weights assigned to the edges such that the less the weight, the more critical the edge is. After the edge weights in the extended *SDFG* are calculated for each step, the overall edge weight for the interconnection between each pair of modules is calculated by finding the minimum of the weights for that edge across all steps. The edge weights between every pair of modules is stored in a matrix S , which is used to compute the cost matrix C_l as follows.

$$C_l(i, j) = \frac{M - S(i, j)}{M}$$

where $M = \max_{i,j} S(i, j)$. Each entry in the matrix C_l represents the urgency of the interconnect between modules i and j .

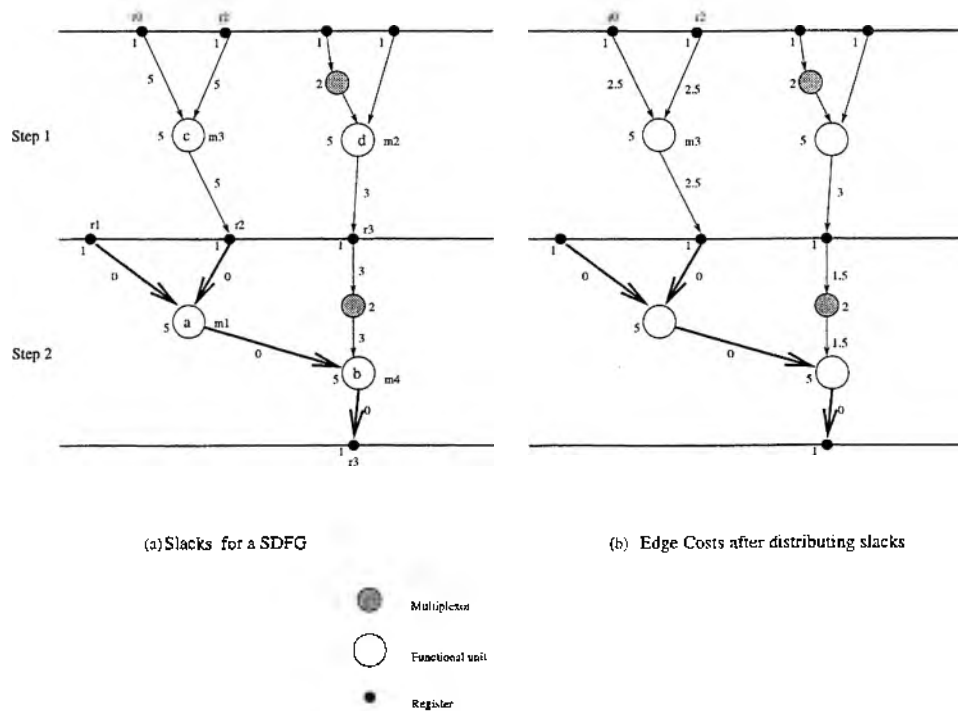


Figure 4.3 Slacks for a sample SDFG and delay values after slack distribution

4.3.2 Clustering phase

We use a greedy algorithm to cluster modules. Initially, we start with a set of degenerate clusters which is essentially the set of modules itself, each module being a cluster. The clustering is done based on the cost matrix C_i and the areas of the clusters. Figure 4.4 shows an example of how incorporating wasted area in the choice of the two clusters can affect the overall area. In (a), modules A and B are clustered together, and the result clustered with C . This results in a lot of wasted area. In (b), modules B and C are clustered together, and the result clustered with A resulting in no wasted area, and a floorplan with lower area.

The pair of clusters which are connected by the edge which has the highest cost is selected and combined into one. If two pairs of clusters have the same cost, that pair which have similar areas are selected for clustering. When two clusters are coalesced to form a larger cluster, the set of arcs from that cluster is the union of all arcs from the modules within the cluster to the modules in other clusters. The cost of the interconnection between the new cluster and the ex-

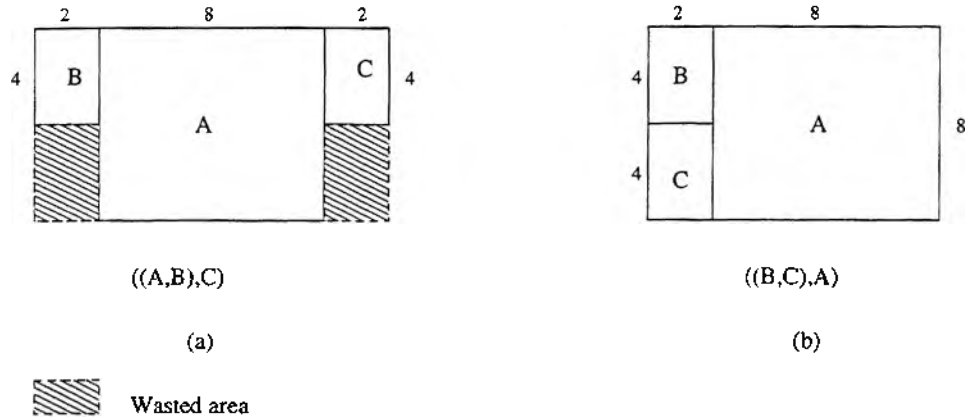


Figure 4.4 Example showing how combining clusters based on area affects the overall area of the floorplan

isting clusters is calculated as the maximum of the costs of all arcs from or to the new cluster. This combining process is repeated until we have one cluster.

4.3.3 Bounding curve calculation

The bounding curve [13], [24], [26] of a module is the set of all minimal (w_i, h_i) pairs for the module, where w_i is the minimum width of the module when its height is h_i (and vice versa). Given a slicing tree, we can associate a bounding curve with each node. Consider the leaf nodes. The points in the bounding curve are determined as explained above. The bounding curve of a non-leaf node is determined by combining the bounding curves of the child nodes. The combining process is determined by the type of dissection specified by the parent node, viz vertical or horizontal (denoted by + and * respectively). The combined curve gives the set of all feasible dimensions of the module formed by putting the child modules adjacent to each other horizontally (for a vertical slice), or vertically (for a horizontal slice). Given two bounding curves Γ and Λ , $\Gamma + \Lambda$ is given by the curve $\{(u, v + w) | (u, v) \in \Gamma \text{ and } (u, w) \in \Lambda\}$. $\Gamma * \Lambda$ is similarly defined by the curve $\{(u + v, w) | (u, w) \in \Gamma \text{ and } (v, w) \in \Lambda\}$. Only the minimal points need to be stored, i.e., there does not exist two points i and j in the bounding curve such that $w_i \leq w_j$ and $h_i \leq h_j$.

In addition to the width and length measures for each module, we add a third dimension, which is the cumulative interconnect delay measure. For a leaf node, it is zero. For a non-leaf node, it is determined by adding the cost of the child nodes and the Manhattan distance between the centers of the modules for the slicing structure being considered, weighted by the cost of the edge. Thus, the points in the augmented bounding curve consists of tuples of the form $\langle w_i, h_i, d_i \rangle$. Consider combining two points $\langle w_1, h_1, d_1 \rangle$ and $\langle w_2, h_2, d_2 \rangle$ assuming a vertical slice. The resulting point is given by $\langle w_1 + w_2, \max(h_1, h_2), (d_1 + d_2 + (w_1 + w_2)/2) \rangle$.

In contrast to the conventional schemes for curve addition [24], [26], [13], [28], [27] to compute the bounding curves, we assume that the slicing structure of the layout is not known during the augmented bounding curve calculation. Consider a step during clustering in which a set of child clusters is combined to form a parent cluster. Since the clustering and augmented bounding curve computation is done in a bottom up manner, the augmented bounding curves of all the child clusters are known at this point. The set of all slicing structures of the parent cluster is now enumerated [22], [14]. Corresponding to each slicing structure, the augmented bounding curve of the parent module is computed from the augmented bounding curves of the child clusters as explained above. These augmented bounding curves, corresponding to all the slicing structures of the parent, are now combined using a minimum function. As a result, we get a combined augmented bounding curve B whose individual points are taken from the augmented bounding curves for the different slicing structures and such that the points in the resulting curve are minimal. In other words, if $B_1, B_2 \dots B_k$ are the bounding curves corresponding to the different slicing structures, the combined curve B consists of the set of all points in $B_1 \cup B_2 \cup \dots B_k$ and there does not exist a pair of points i and j in B such that $w_i \leq w_j$, $h_i \leq h_j$ and $d_i \leq d_j$. The identity of the slicing structure associated with each point is recorded, which aids in generating the floorplan from top down.

Once the augmented bounding curve for the highest level cluster is determined, the cost of each point in that curve is computed based on a function which is a combination of area and interconnect costs determined by the bottom-up clustering and augmented bounding curve computation. The minimum cost point is selected. This point, along with the topology information stored by the previous phase uniquely determines the slicing structure of the floorplan.

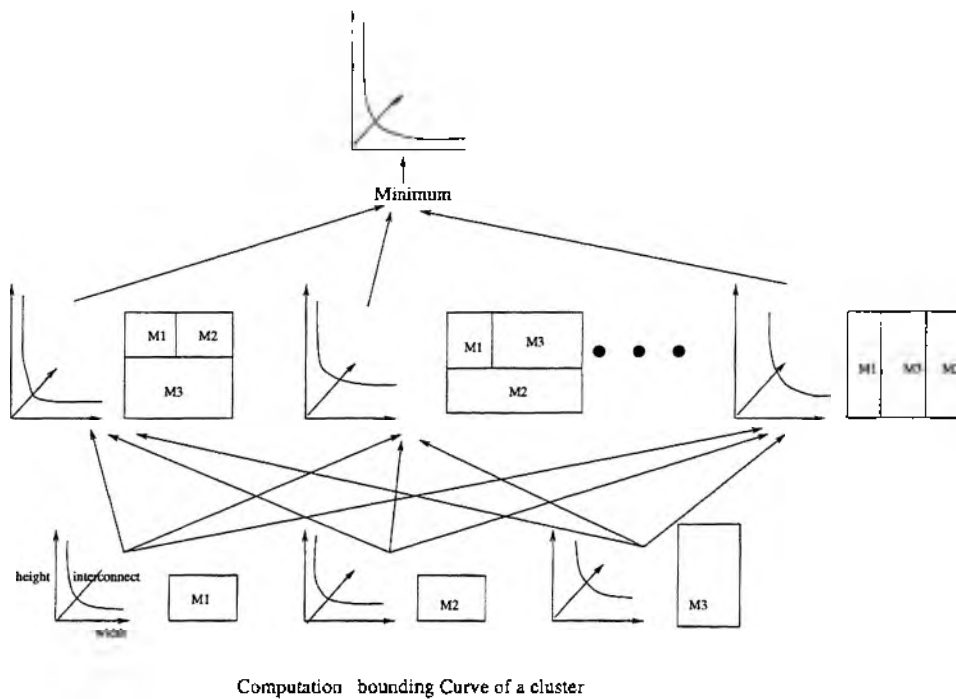
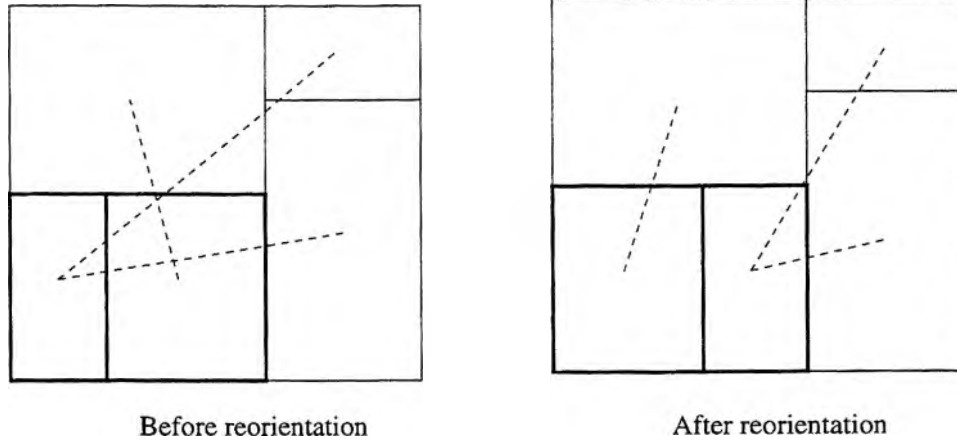


Figure 4.5 Computation of augmented bounding curve of a cluster

Figure 4.5 shows an example in which the augmented bounding curve of a cluster consisting of three child clusters are combined to form the augmented bounding curve for the parent cluster.

4.3.4 Reorientation phase

Once the topology is known, a reorientation of the clusters is performed. It is based on a technique similar to that proposed in [23], [14]. Consider a cluster size of 2 for example. For the top level cluster, there are two child nodes. For the given topology, there are two orientations for each child module which conserve the area of the parent (four, if the child module is square). Each of these requires the same area, but they have different interconnect costs. The orientations which gives the best interconnect cost is determined. This process is recursively performed on each of the child node. When deciding the orientation of a module, the interconnects to all other modules in the floorplan is taken into account (in addition to the sibling). Thus, this approach tries to minimize the interconnection cost in a global fashion. Figure 4.6 shows a reorientation of a cluster.



An example of how reorientation reduces interconnect cost
(dashed lines indicate interconnections)

Figure 4.6 Reducing interconnect cost by reorientation

4.3.5 Calculation of interconnect delays

The length of the interconnection between two modules in a floorplan is taken as the Manhattan distance between the centers of the modules. Given the interconnect length l_{int} , the following equations are used to calculate the interconnect delay [15], [32].

$$R_{int} = \rho \frac{l_{int}}{W_{int} H_{int}}$$

R_{int} = resistance of wire, W_{int} = width of wire, H_{int} = height of wire, l_{int} = length of wire.

$$C_{int} = \epsilon_{ox} \left\{ 1.15 \left(\frac{W_{int}}{t_{ox}} \right) + 2.8 \left(\frac{H_{int}}{t_{ox}} \right) \right\}$$

C_{int} = capacitance per unit length

$$T = R_{int} C_{int} + 2.3 (R_{tr} C_{int} + R_{tr} C_L + R_{int} C_L)$$

where R_{tr} = equivalent resistance of driver transistor, C_L = load capacitance (assume to be equal to gate capacitance of one transistor).

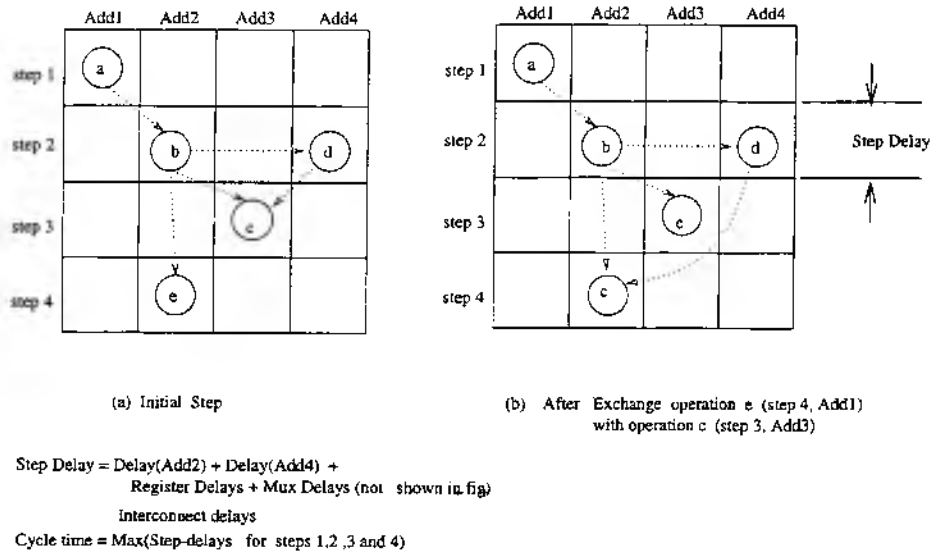


Figure 4.7 Representation of scheduling and binding problem as a placement problem in a two-dimensional table

We make the following assumptions (as in [15]) about wire height H_{int} , wire width W_{int} and thickness of oxide t_{ox} . $H_{int} = 1/3W_{int}$ and $t_{ox} = 1/2W_{int}$ and interconnect material is Aluminum. As in the case of [15], we select the nominal values of R_{tr} and C_L given in [35].

4.4 The combined scheduling, binding and floorplanning algorithm

The combined scheduling, binding and floorplanning is a simulated annealing based algorithm similar to the combined scheduling and binding algorithm proposed by Srinivas and Newton [12]. It transforms the combined scheduling and binding into a placement problem in a two-dimensional table. The rows represent the steps and columns represent functional units as shown in Figure 4.7. We consider two types of moves:

- Move an operation from one position in the table to another
- Interchange two operations

```

COMBINED-SCHEME()
1  COMPUTE-START-TEMPERATURE()
2  while  $T \geq 0.1$  and not converged
3      do for  $k \leftarrow 1$  to moves-per-temp
4          do GENERATE-MOVE ()
5              ALLOCATE-REGISTERS ()
6              ALLOCATE-MUXES()
7              CONSTRUCT-FLOORPLAN()
8              CALCULATE-LATENCY()
9               $C \leftarrow \alpha L + \beta A$ 
10             if  $\Delta C < 0$ 
11                 then ACCEPT-MOVE()
12             else Accept with prob  $e^{-\frac{\Delta C}{T}}$ 
13              $T \leftarrow T \times 0.9$ 

```

Figure 4.8 Simultaneous scheduling, binding and floorplanning

Each move is performed only if the precedence constraints are satisfied, and at most one operation is assigned to a functional unit. Two operations are chained together in a step if there is a precedence constraint from one of them to the other and the latter operation is an immediate successor of the former. Each move is evaluated by performing the fast, constructive timing driven floorplanning algorithm, and calculating the area of the floorplan and latency of the schedule based on that. The latency of the schedule is calculated as the product of the number of cycles and the cycle delay, where the cycle delay is the delay of the longest step, which includes the functional unit delays, register and mux delays and the interconnect delays between modules. It is assumed that the external inputs to the signal flow graph are available at the beginning of the first cycle. The interconnect delays between two modules is calculated based on their positions in the floorplan constructed above. This is in contrast to [12] where interconnect delays are not considered in the cycle time, and area is estimated as a cost function dependent on the areas of the modules. The overall algorithm is as shown in Figure 4.8.

The initial temperature is determined by finding the average change in cost for a set of random moves from the starting configuration and selecting the temperature which leads to an accept probability of 0.95. The number of moves per temperature is set as a constant (typically 20 to 30) times the number of operations in the signal flow graph.

4.4.1 Cost function

The cost function is a linear function of area and the latency of the schedule.

$$C = \alpha L + \beta A$$

where area A is given by the layout generated by the constructive floorplanning algorithm, and the latency L is the product of the number of cycles and the delay of the longest cycle, where the delay is the longest path in a step including the latencies of functional units, mux delays, register delays and the interconnect delays.

4.4.2 Register and multiplexor allocation

For simplicity, we assume that there is a set of registers associated with each module. This set can have size zero if the module does not need to store any values, in which case, it will be collapsed before area calculation. The number of registers for any given module is determined by calculating the maximum clique size of an interval graph representing the lifetimes of values computed by the module. This is done efficiently using the left edge algorithm for non-hierarchical graphs with no loops.

We assume that there is a k to 1 multiplexor at each input of the modules, where k is the number of different modules feeding into that input. If k has value 1, that multiplexor need not exist and will be collapsed before area calculation.

4.5 The two-step algorithm

The two-step approach used for comparison performs a combined scheduling and binding using simulated annealing in the first step, followed by a floorplanning algorithm. Two floorplanning algorithms have been used for comparison, the first one being the constructive floorplanning algorithm described in section 4.3, and the second one being a simulated annealing based floorplanning algorithm described by Wong and Liu [13].

The combined scheduling and binding algorithm is briefly summarized below.

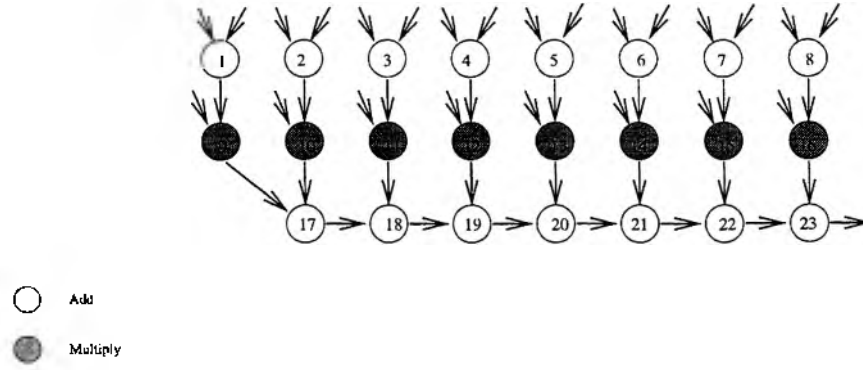


Figure 4.9 Signal flow graph of FIR filter

A simulated annealing algorithm similar to that proposed by Srinivas and Newton [12] is used. The algorithm performs the following moves:

- Move an operation from one position in the table to another
- Interchange two operations

The operations and locations are randomly generated. Each move makes sure that the precedence constraints among operations are not violated. Two operations a and b are chained together in a step if both of them are scheduled to the same step, and one of them consumes the result generated by the other. The area A used for evaluating the cost function is taken as the sum of the areas of modules, and the latency L of the schedule is the product of the number of cycles and the delay of the longest step (which contains the critical path). The delay considers only the functional unit, mux and register latencies, since the interconnect delays are not known at this stage. Registers and muxes are allocated as in the case of the combined approach.

4.6 Experimental results

We conducted experiments on a set of high-level synthesis benchmarks used by other researchers, which include a 2-step FIR filter, 3-step FIR filter, a 3-step schedule for a synthetic benchmark and an 8-step elliptic filter, shown in Figure 4.9, 4.10 and 4.11. The cost function was selected such that latency has more importance in comparison to the area.

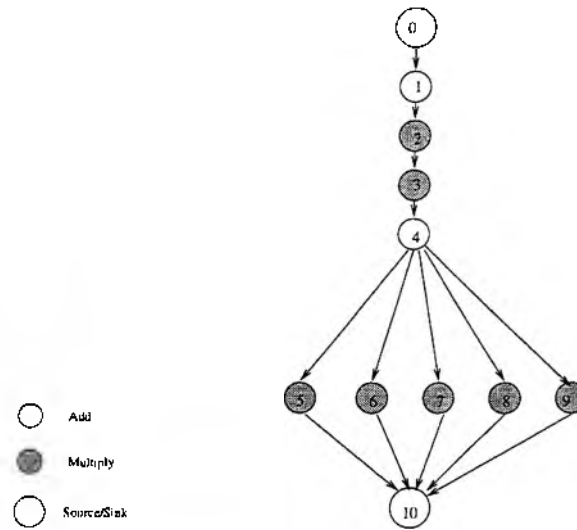


Figure 4.10 Signal flow graph of synthetic benchmark

In the following, 'combined scheme' refers to our simultaneous scheduling, binding and floorplanning algorithm. We use two library sets for functional modules as in [15]. The libraries are as shown in Table 4.1. The areas are in square microns and delays in nano seconds.

Table 4.1 The library set

Library 1						
Fabrication	16 bit adder		16 bit multiplier		16 bit register	
Technology	Delay	Area	Delay	Area	Delay	Area
1.6 μm	18	746875	200	8711250	3	597500
1.2 μm	13	420000	150	4900000	3	336000
Library 2						
Fabrication	8 bit adder		8 bit multiplier		8 bit register	
Technology	Delay	Area	Delay	Area	Delay	Area
1.2 μm	13	46875	32	564375	2	37500
1.0 μm	12	21250	30	261875	1.5	17000

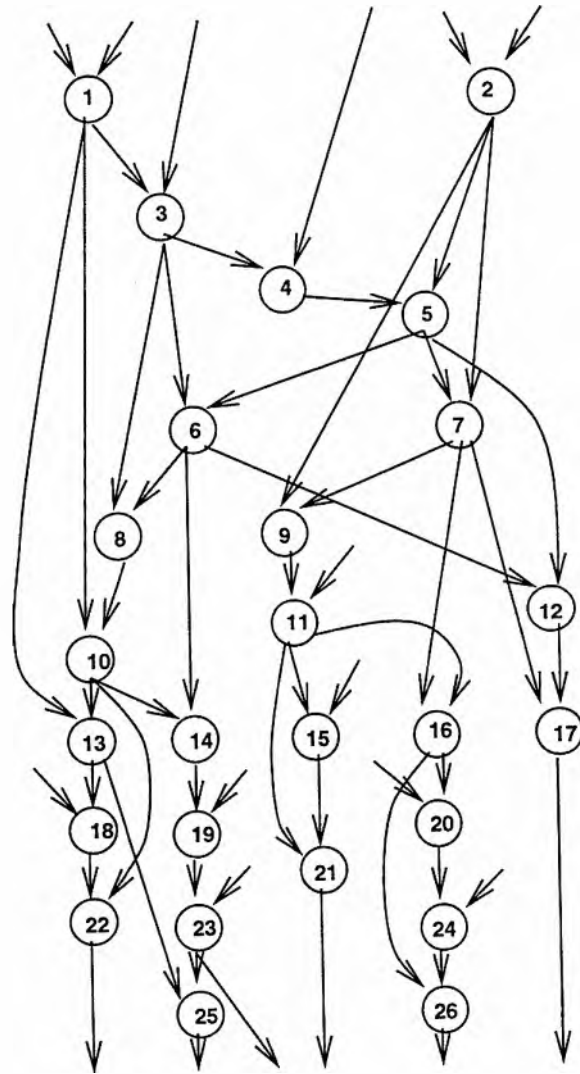


Figure 4.11 Signal flow graph of Elliptic filter benchmark

4.6.1 Comparison between the combined scheme and Fang and Wong's algorithm

Fang and Wong's algorithm [15] starts with a scheduled data flow graph. It performs the simulated annealing based floorplanning algorithm proposed by Wong and Liu [13]. After each move, a performance driven binding algorithm is performed, which attempts to optimize the cycle time for the floorplan considered by the annealing algorithm. The cost function includes the interconnection delay along the critical path. Further details of the Fang and Wong's algorithm appears in [15]. Tables 4.2 and 4.3 shows the comparison between Fang and Wong's algorithm and the combined scheme.

Table 4.2 Comparison between Fang and Wong algorithm and the combined scheme for 2-step FIR filter

Fabrication Technology μm	Fang and Wong		combined scheme	
	Area mm^2	Wiring delay(ns)	Area mm^2	Wiring delay(ns)
1.6 (16 bit)	55.8	22.4	67.47	8.02
1.2 (16 bit)	31.7	19.5	37.34	5.96
1.2 (16 bit)	3.8	6.3	4.23	2.72
1.0 (16 bit)	1.7	4.0	2.02	1.90

Table 4.3 Comparison between Fang and Wong algorithm and the combined scheme for 3-step FIR filter

Fabrication Technology μm	Fang and Wong		combined scheme	
	Area mm^2	Wiring delay(ns)	Area mm^2	Wiring delay(ns)
1.6 (16 bit)	44.7	15.9	53.94	7.13
1.2 (16 bit)	25.1	12.0	31.61	5.39
1.2 (8 bit)	2.9	5.5	3.59	2.56
1.0 (8 bit)	1.3	3.4	1.74	1.84

4.6.2 Comparison between the combined scheme and a two-step scheme which uses a constructive floorplanning algorithm

In this section, a comparison between the combined scheme and a two-step scheme which performs the combined scheduling and binding algorithm described in section 4.5 followed by the constructive floorplanning algorithm described in section 4.3, is presented. Tables 4.4, 4.5, 4.6 and 4.7 shows the comparison between the designs generated by the combined scheme and those generated by the two-step algorithm. Wiring delay refers the overhead in the cycle time due to interconnect delays. The results show that the combined scheme gives better results in almost all cases when compared to the two-step algorithm. It was able to arrive at a lower wiring delay than that for the two-step algorithm in most cases with little or no degradation in area (in fact, the area of designs generated by the combined scheme was better, in many cases).

Table 4.4 Comparison between two-step (using constructive floorplanner) and combined schemes for synthetic benchmark

Fabrication Technology μm	two-step			combined scheme			% reduction Wiring delay
	Area mm^2	Latency ns	Wiring delay(ns)	Area mm^2	Latency ns	Wiring delay(ns)	
1.6 (16 bit)	58.61	724.05	15.90	58.48	698.84	7.45	53.14
1.2 (16 bit)	32.95	546.43	11.64	32.60	526.98	5.16	55.67
1.2 (8 bit)	0.329	162.81	3.93	0.334	156.65	1.88	52.16
1.0 (8 bit)	0.156	145.74	2.84	0.159	141.36	1.38	51.41

4.6.3 Comparison between the combined scheme and a two-step scheme which uses a simulated annealing based floorplanning algorithm

In this section, a comparison between the combined scheme and a two-step algorithm which performs the combined scheduling and binding algorithm described in section 4.5 followed by a simulated annealing based floorplanning algorithm, is presented. The floorplanning algorithm

Table 4.5 Comparison between two-step (using constructive floorplanner) and combined schemes for 2-step FIR

Fabrication Technology μm	two-step			combined scheme			% reduction Wiring delay
	Area mm^2	Latency ns	Wiring delay(ns)	Area mm^2	Latency ns	Wiring delay(ns)	
1.6(16 bit)	54.34	583.08	12.04	57.19	574.40	7.70	36.05
1.2(16 bit)	30.21	444.04	12.52	31.38	430.47	5.74	54.15
1.2(8 bit)	3.18	186.22	3.77	3.38	183.76	2.54	32.63
1.0(8 bit)	1.49	168.22	2.37	1.56	167.23	1.87	21.10

Table 4.6 Comparison between two-step (using constructive floorplanner) and combined schemes for 3-step FIR

Fabrication Technology μm	two-step			combined scheme			% reduction Wiring delay
	Area mm^2	Latency ns	Wiring delay(ns)	Area mm^2	Latency ns	Wiring delay(ns)	
1.6 (16 bit)	46.16	761.28	10.26	46.87	753.7	6.98	31.97
1.2 (16 bit)	25.36	573.77	7.76	26.24	568.60	5.28	31.96
1.2 (8 bit)	2.83	216.98	8.99	2.81	198.99	2.32	74.19
1.0 (8 bit)	1.37	181.46	2.75	1.27	179.47	1.71	37.82

is similar to that proposed by Wong and Liu [13]. The cost function used in the experiments is a linear combination of the area and the cycle time, in contrast to the original scheme proposed in [13], which uses a linear combination of the area and the total wire length as the cost function. The following section summarizes the simulated annealing based floorplanning algorithm.

4.6.3.1 Floorplanning using simulated annealing

The floorplan is represented using a postfix notation, which is generated by traversing the slicing tree of the floorplan in post order [13]. The leaf nodes are the operands in the postfix expression, and the internal nodes are the operators. The two operators are + and *, representing a vertical and horizontal slice, respectively. The different moves considered by the annealing algorithm are as follows:

- Interchange two consecutive operands

Table 4.7 Comparison between two-step (using constructive floorplanner) and combined schemes for 8-step Elliptic filter

Fabrication Technology μm	two-step			combined scheme			% reduction Wiring delay
	Area mm^2	Latency ns	Wiring delay(ns)	Area mm^2	Latency ns	Wiring delay(ns)	
1.6 (16 bit)	15.47	392.28	5.54	15.61	371.19	2.90	47.65
1.2 (16 bit)	8.24	301.35	4.17	8.91	284.90	2.11	49.40
1.2 (8 bit)	0.851	274.6	3.33	0.896	256.10	1.01	69.67
1.0 (8 bit)	0.426	234.86	1.62	0.419	229.41	0.939	42.04

- Complement an operator chain
- Interchange a neighboring operand-operator pair.

The moves makes sure that the resulting expression is still a ballot sequence [13].

The area A used for evaluating the cost function is given by the area of the floorplan constructed by the floorplanner, and the latency L is given by the product of the number of cycles and the delay of the longest step. In this step, the delay also considers the interconnect delays between modules (calculated from the floorplan) in addition to functional unit, mux and register latencies.

Tables 4.8,4.9,4.10 and 4.11 show the comparison between the combined scheme and the two step scheme which uses the annealing based floorplanning algorithm. It can be seen that the combined scheme was able to reduce the interconnection delay along the critical path by up to around 46 percent in comparison to the two-step algorithm. The areas are either lower or almost the same for the combined scheme.

4.6.3.2 Disclaimer

The results generated by the simulated annealing based floorplanning algorithm are sensitive to various parameters like the starting configuration, the initial temperature, the random number seed, the cooling schedule, moves per temperature and the implementation of the annealing algorithm. The results in this section are for a particular starting configuration and a cooling

schedule which decreases the temperature by a factor of 0.9 after each temperature. The initial temperature was selected such that 95 percent of moves are accepted at that temperature. Experiments were conducted, with three different random number seeds and three different moves per temperature (i.e., nine per benchmark), and the results reported are the minimum of those generated by those experiments.

Table 4.8 Comparison between two-step (using annealing based floorplanner) and combined schemes for synthetic benchmark

Fabrication Technology μm	two-step			combined scheme			% reduction Wiring delay
	Area mm^2	Latency ns	Wiring delay(ns)	Area mm^2	Latency ns	Wiring delay(ns)	
1.6 (16 bit)	59.30	712.37	11.96	58.48	698.84	7.45	37.71
1.2 (16 bit)	34.61	536.28	8.26	32.60	526.98	5.16	37.53
1.2 (8 bit)	0.362	158.32	2.43	0.334	156.65	1.88	22.63
1.0 (8 bit)	0.170	142.51	1.76	0.159	141.36	1.38	21.59

Table 4.9 Comparison between two-step (using annealing based floorplanner) and combined schemes for 2-step FIR

Fabrication Technology μm	two-step			combined scheme			% reduction Wiring delay
	Area mm^2	Latency ns	Wiring delay(ns)	Area mm^2	Latency ns	Wiring delay(ns)	
1.6(16 bit)	78.16	586.63	13.82	57.19	574.40	7.7	44.28
1.2(16 bit)	43.98	434.30	7.65	31.38	430.47	5.74	24.97
1.2(8 bit)	3.83	184.84	3.08	3.38	183.76	2.54	17.53
1.0(8 bit)	1.58	167.84	2.18	1.56	167.23	1.87	14.22

4.7 Summary

With small device features in sub-micron technologies, since the interconnect delays play a dominant role in the cycle time, it is important to consider the effect of physical design on high-level synthesis. We have presented a new algorithm which combines physical design and

Table 4.10 Comparison between two-step (using annealing based floorplanner) and combined schemes for 3-step FIR

Fabrication Technology μm	two-step			combined scheme			% reduction Wiring delay
	Area mm^2	Latency ns	Wiring delay(ns)	Area mm^2	Latency ns	Wiring delay(ns)	
1.6 (16 bit)	57.36	769.44	12.98	46.87	753.7	6.98	46.22
1.2 (16 bit)	28.45	580.21	9.90	26.24	568.6	5.28	46.67
1.2 (8 bit)	2.93	199.75	3.24	2.81	198.99	2.32	28.40
1.0 (8 bit)	1.56	181.28	2.69	1.27	179.47	1.71	36.43

Table 4.11 Comparison between two-step (using annealing based floorplanner) and combined schemes for 8-step Elliptic filter

Fabrication Technology μm	two-step			combined scheme			% reduction Wiring delay
	Area mm^2	Latency ns	Wiring delay(ns)	Area mm^2	Latency ns	Wiring delay(ns)	
1.6 (16 bit)	15.61	375.81	3.48	15.61	371.19	2.90	16.67
1.2 (16 bit)	8.58	295.92	3.49	8.91	284.90	2.11	39.54
1.2 (8 bit)	0.994	261.61	1.70	0.896	256.1	1.01	40.59
1.0 (8 bit)	0.483	232.26	1.29	0.419	229.41	0.939	27.21

high level synthesis. Experimental results show that the approach is quite effective in arriving at solutions of better quality in comparison to a traditional approach of performing high-level synthesis and physical design one after the other.

CHAPTER 5

SIMULTANEOUS SCHEDULING, BINDING AND FLOORPLANNING IN HIGH-LEVEL SYNTHESIS FOR INTERCONNECT ENERGY MINIMIZATION

5.1 Introduction

The interconnect energy dissipation is proportional to the product of the switching activity and the capacitance of the interconnects. The switching activity on an interconnect is determined by the sequence of variables assigned to it, which in turn depends on the schedule and binding. The capacitance of an interconnect is directly dependent on its length, which is determined by the floorplan. Thus, to minimize the interconnect energy dissipation, it is important to consider the switching as well as the floorplan information during scheduling and binding. Similarly, the floorplanner should take the schedule and binding into account to minimize the interconnect energy dissipation. It should try to minimize the length of interconnects which have high switching activity, for example.

A simulated annealing algorithm which simultaneously performs scheduling, binding and power-driven floorplanning with the objective of optimizing the interconnect energy dissipation, while trying to minimize the impact on latency and area is presented in this chapter. We compare the results with that of an algorithm which considers only latency and area, and ignores power

and show that it is possible to reduce interconnect energy dissipation by around 30 to 60 percent for several benchmark circuits.

In Section 5.2, the power estimation algorithm is described. In Section 5.3, a constructive power-driven floorplanning algorithm is presented whose primary objective is to reduce interconnect energy dissipation. It also tries to reduce delays along critical paths as a secondary objective. In Section 5.4, the algorithm for combined scheduling, binding and floorplanning which minimizes interconnect energy is described. In Section 5.5, the results are presented for a set of high-level synthesis benchmarks.

5.2 Power estimation using switching tables

The power dissipation as a result of switching can be calculated using the equation $P = \frac{1}{2T} \sum_{i \in \text{nodes}} V_{DD}^2 s_i c_i$ where s_i is the number of times node i switches during time T , c_i is the capacitance of node i and V_{DD} is the supply voltage.

The estimation of switching on a resource can be efficiently performed using the idea of the switching table [8], [5]. A switching table for a resource r , for a signal flow graph without conditional statements, is a two-dimensional matrix S where the row i and column j stands for the i^{th} operation, and $S(i, j)$ is the switching that results if i and j are scheduled on r one after the other with no other operation in between. For interconnection links, an operation is essentially the transmission of a variable from one module to another.

The switching table for resource r can be expressed as the sum of two switching tables [1]. One is the intra-iteration switching table M_r and the other is the inter-iteration switching table N_{rk} . Intra-iteration switching $M_r(i, j)$ represents the switching incurred when operation i is scheduled immediately before j on resource r . Inter-iteration switching $N_{rk}(i, j)$ represents the switching incurred when operation j is scheduled first in loop k on resource r and i is the last operation scheduled in loop k on resource r .

The switching table can be calculated by simulation of the operations in the signal flow graph using representative input vectors. Each input vector results in a particular value for each vari-

able, and the switching can be calculated using the switching function for the resource. Details of switching table calculation can be found in [1].

Consider a particular schedule, binding and floorplan for a signal flow graph. Let l_{ij} be the interconnection in the floorplan between modules i and j . Let $v_1, v_2, \dots, v_k$ be the variables assigned in sequence on l_{ij} as a result of communications between modules i and j to perform the computations assigned to i and j by the schedule and binding. Then the switching s_{ij} on the link is determined by pairs of variables $(v_1, v_2), (v_2, v_3) \dots (v_{k-1}, v_k)$ as given by the switching table. The capacitance c_{ij} is proportional to the distance between i and j in the floorplan. Thus, the energy dissipated by l_{ij} is proportional to $s_{ij}c_{ij}$.

5.3 The constructive power-driven floorplanning algorithm

The constructive power-driven floorplanning algorithm is similar to the timing-driven floorplanning algorithm described in the previous chapter, except that it considers the switching activity on the interconnections between modules to decide which modules have to be adjacent in the floorplan.

Figure 5.1 shows an example of the communication between functional modules (shown as arcs between modules) for a particular cycle. The numbers next to each arc represents the switching on that link. Consider modules r_1, M_1, M_2 and r_9 in the figure. Since the links between them exhibit high switching activity, the capacitance of the interconnections between the module pairs (r_1, M_1) , (M_1, M_2) and (M_2, r_9) could have a major impact on the overall power. Thus, it is desirable to have them as close as possible to each other in the floorplan. However, since the link between modules r_5 and M_3 have low switching activity, they can be farther away in the floorplan without possibly affecting the power consumption.

As explained in the previous chapter, the timing-driven floorplanning algorithm constructs an urgency-based cost matrix using the slacks of the interconnection between modules, and uses it for clustering, bounding curve construction and reorientation. The power-driven floorplan-

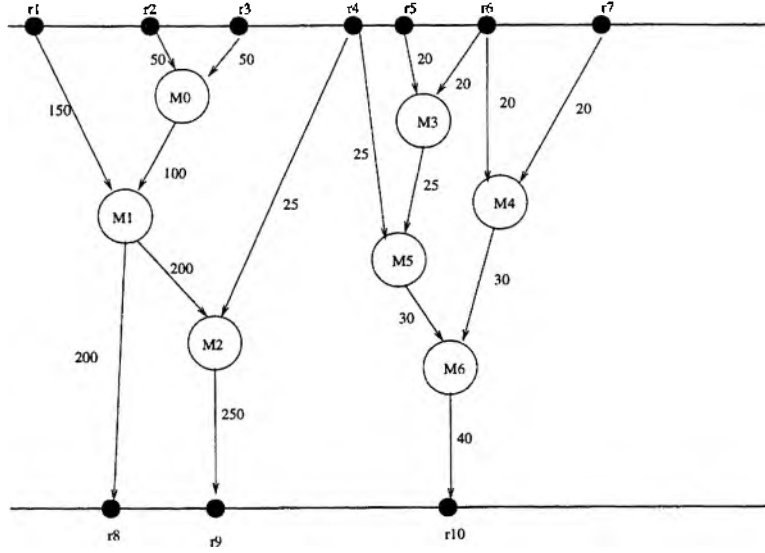


Figure 5.1 Example showing switching along edges between modules

ner also proceeds in a similar manner, but with the difference that instead of the urgency-based cost matrix, a cost matrix C_p , in which the entry (i, j) is a function of the switching activity on the link between modules i and j in addition to the slack of connection between the modules, is constructed. The higher the value of the entry $C_p(i, j)$, the closer the modules i and j should be in the floorplan, in order to minimize the capacitance of the link between them and also to minimize the communication delay on the link. The construction of the cost matrix for the power-driven floorplanning algorithm is described in detail in the following section.

5.3.1 Cost matrix

The cost matrix C_p is an $n \times n$ matrix where n is the number of modules. It is computed as

$$C_p = \lambda T + (1 - \lambda)C_l$$

where T is the transition matrix where $T(i, j)$ is the total switching on the edge between module i and j , C_l is the urgency matrix which is calculated as in the case of the timing-driven floorplanning algorithm, and λ is a user defined parameter which indicates the relative importance of power minimization over latency minimization. Computation of T is performed as follows.

5.3.2 Transition matrix calculation

Each edge of the extended signal flow graph represents a variable computed by the source node of the edge and consumed by the sink node of the edge. As indicated before, the switching on an edge has two components, intra-iteration switching and inter-iteration switching.

Let $V_s(i, j)$ be the variable assigned to edge (i, j) during step s and let t be the earliest step after s during which a different variable is assigned to edge (i, j) . Then intra-iteration switching on edge (i, j) is given by

$$Intra(i, j) = \sum_s S_{intra}(V_s(i, j), V_t(i, j))$$

where S_{intra} is the intra-iteration switching table.

Inter-iteration switching, $Inter(i, j)$ is calculated as follows. Let $First(i, j)$ be the first variable assigned to edge (i, j) from the beginning of the schedule. Let $Last(i, j)$ be the last variable assigned to (i, j) . Then

$$Inter(i, j) = S_{inter}(Last(i, j), First(i, j))$$

where S_{inter} is the inter-iteration switching table.

The transition matrix T , which stores the total switching between every module pair i and j is given by

$$T(i, j) = Intra(i, j) + Inter(i, j)$$

where $Intra(i, j)$ and $Inter(i, j)$ denote the intra and inter-iteration switching respectively on the link between modules i and j . $T(i, j)$ is zero if there is no communication between modules i and j .

5.3.3 Estimation of interconnect energy

Let l_{ij} be the interconnect link between modules i and j . Capacitance c_{ij} of the link l_{ij} is estimated as a constant times the Manhattan distance between modules i and j in the floorplan.

```

POWER-DRIVEN-SCHEME()
1  COMPUTE-START-TEMPERATURE()
2  while  $T \geq 0.1$  and not converged
3      do for  $k \leftarrow 1$  to moves-per-temp
4          do GENERATE-MOVE ()
5              ALLOCATE-REGISTERS ()
6              ALLOCATE-MUXES ()
7              CONSTRUCT-FLOORPLAN()
8              ESTIMATE-INTERCONNECT-ENERGY ()
9              CALCULATE-LATENCY ()
10              $C \leftarrow \alpha P + \beta L + \gamma A$ 
11             if  $\Delta C < 0$ 
12                 then ACCEPT-MOVE()
13             else Accept with prob  $e^{-\frac{\Delta C}{T}}$ 
14              $T \leftarrow T \times 0.9$ 

```

Figure 5.2 Simultaneous scheduling, binding and floorplanning for low-power

The switching activity estimate s_{ij} is given by $T(i, j)$, and the interconnect energy estimate P is calculated as $\sum_{i,j} s_{ij} c_{ij}$.

5.4 Overall algorithm

The simultaneous scheduling, binding and floorplanning for low power is as shown in Figure 5.2. It is a simulated annealing based algorithm and the moves are similar to that for the case of the algorithm for simultaneous scheduling, binding and floorplanning for latency optimization. The cost function includes interconnect energy, latency and area, and is calculated as described in the following section.

5.4.1 Cost function

The cost function is a linear function of the interconnect energy, latency and the area of the solution.

$$C = \alpha P + \beta L + \gamma A$$

where the interconnect energy estimate P is as given by the power estimation algorithm, latency L is the product of the number of cycles and the delay of the longest cycle, where the delay is the longest path in a step including the latencies of functional units, mux delays, register delays and the interconnect delays, and area A is given by the layout generated by the constructive floorplanning algorithm.

5.5 Experimental results

We conducted experiments on a 2-step FIR filter, 3-step FIR filter and a 8-step elliptic filter. In the following, 'power-driven scheme' refers to our simultaneous scheduling, binding and floorplanning algorithm which considers interconnect energy, and 'latency-driven scheme' refers to a simulated annealing algorithm which simultaneously performs scheduling, binding and floorplanning with the objective of minimizing latency, but ignoring power [2]. We use two library sets for functional modules as in [15]. The libraries are as shown in Table 4.1. Tables 5.1, 5.2 and 5.3 shows the comparison between the designs generated by the power-driven scheme and the latency-driven scheme. The energy estimate is the sum of the product of switching and interconnect lengths over all interconnect links (denoted by SC). The results show that the power-driven scheme gives better results in almost all cases when compared to the latency-driven algorithm. In many cases, it was able to reduce the interconnect energy dissipation by more than a factor of two, at the expense of increased latency and minimal impact on area.

Table 5.1 Comparison between latency-driven and power-driven schemes for 2-step FIR

Fabrication Technology μm	latency-driven			power-driven scheme			
	Area mm^2	Latency ns	Energy Est ($SC \times 10^{-4}$)	Area mm^2	Latency ns	Energy Est ($SC \times 10^{-4}$)	Energy Reduction (%)
1.6(16 bit)	57.19	574.39	1774.02	55.76	650.15	1078.99	39.18
1.2(16 bit)	31.37	430.47	1298.83	31.34	475.30	829.95	36.10
1.2(8 bit)	3.397	183.76	411.05	3.26	214.88	309.54	24.70
1.0(8 bit)	1.590	167.23	314.12	1.54	203.75	214.66	31.66

Table 5.2 Comparison between latency-driven and power-driven schemes for 3-step FIR

Fabrication Technology μm	latency-driven			power-driven scheme			
	Area mm^2	Latency ns	Energy Est ($SC \times 10^{-4}$)	Area mm^2	Latency ns	Energy Est ($SC \times 10^{-4}$)	Energy Reduction(%)
1.6 (16 bit)	47.12	753.70	3437.90	47.20	1001.68	1106.56	67.81
1.2 (16 bit)	26.24	568.60	2071.82	25.84	735.49	713.87	65.54
1.2 (8 bit)	2.798	198.99	482.03	2.69	299.40	278.42	42.24
1.0 (8 bit)	1.26	179.47	473.79	1.31	270.47	201.87	57.39

Table 5.3 Comparison between latency-driven and power-driven schemes for 8-step Elliptic filter

Fabrication Technology μm	latency-driven			power-driven scheme			
	Area mm^2	Latency ns	Energy Est ($SC \times 10^{-4}$)	Area mm^2	Latency ns	Energy Est ($SC \times 10^{-4}$)	Energy Reduction(%)
1.6 (16 bit)	15.6	371.19	1387.16	15.49	418.08	537.91	61.22
1.2 (16 bit)	8.91	284.90	1054.41	8.43	564.77	390.03	63.00
1.2 (8 bit)	0.896	256.10	387.03	0.84	391.14	180.92	53.25
1.0 (8 bit)	0.420	229.40	291.29	0.41	462.47	284.77	2.24

5.6 Summary

With small device features in sub-micron technologies, since the interconnect power is a major fraction of the power consumed by the circuit, it is important to consider the effect of physical design on high-level synthesis for low-power. We have presented a new algorithm which combines physical design and high level synthesis with the aim of minimizing interconnect energy dissipation. Experimental results show that the approach is quite effective in reducing the interconnect energy in comparison to an approach which does not consider interconnect energy.

CHAPTER 6

PARALLEL ALGORITHMS FOR SIMULTANEOUS SCHEDULING, BINDING AND FLOORPLANNING

6.1 Introduction

In the previous chapters, it was shown that simultaneous scheduling, binding and floorplanning is capable of arriving at solutions of better quality in comparison to the traditional approach of performing scheduling and binding followed by floorplanning. However, that approach leads to increased run times. In this chapter, two parallel algorithms for simultaneous scheduling, binding and floorplanning are presented, which are able to arrive at almost the same quality of solution as the sequential approach and also giving close to linear speedups.

Multiple Markov chains [37], [36], [38] have been shown to be an effective way to perform simulated annealing in parallel. Simulated annealing can be thought of as a search path whose moves are accepted or rejected depending on particular cost evaluations. This search path can be considered as a Markov chain. Given p processors, a straightforward way of running simulated annealing in parallel is to initiate simulated annealing with a different seed on different processors and let each processor perform moves independently. Finally the best solution from those computed by all the processors is selected.

Although this approach could potentially result in better solution, it may not result in any speedup. We use an approach similar to the multiple Markov chain approach proposed by Lee and Lee [36] for graph partitioning and Chandy and Banerjee [38] for VLSI cell placement. Let

N be the number of moves performed by the simulated annealing algorithm. The overall idea is to make each processor perform an independent simulated annealing with a different seed, but make it perform only N/p moves. Periodically, the processors exchange solutions and each processor will either continue with its own solution or with a different processor's solution. The number of moves performed between these exchanges should be such that each processor should get sufficient time to advance its own annealing process by a considerable amount. Based on the way this exchange is performed, the multiple Markov chain approach can be synchronous or asynchronous [36], [38].

In the synchronous approach, all processors synchronize at regular intervals and determine the best solution found so far. Each processor then updates its local database with the best solution and continues. A disadvantage of this approach is that it could get trapped in a local minimum. In the asynchronous approach, a processor is designated as the master processor which stores the globally best solution. The other processors periodically check the global best solution and perform an update of its local solution if necessary. In this case, the processors do not have to synchronize and could potentially escape local minima.

6.2 Parallel algorithms for combined scheduling, binding and floorplanning

In this section, we describe two parallel algorithms for simultaneous scheduling, binding and floorplanning all of which are based on the simulated annealing algorithms described in chapters 4 and 5. The first scheme utilizes synchronous multiple Markov chains and the second scheme uses a variation of the asynchronous multiple Markov chain approach. Let P denote the number of processors.

6.2.1 Synchronous multiple Markov chains

In this algorithm, each processor performs N/P moves per temperature where N is the number of moves performed by the sequential algorithm per temperature. Periodically, each proces-

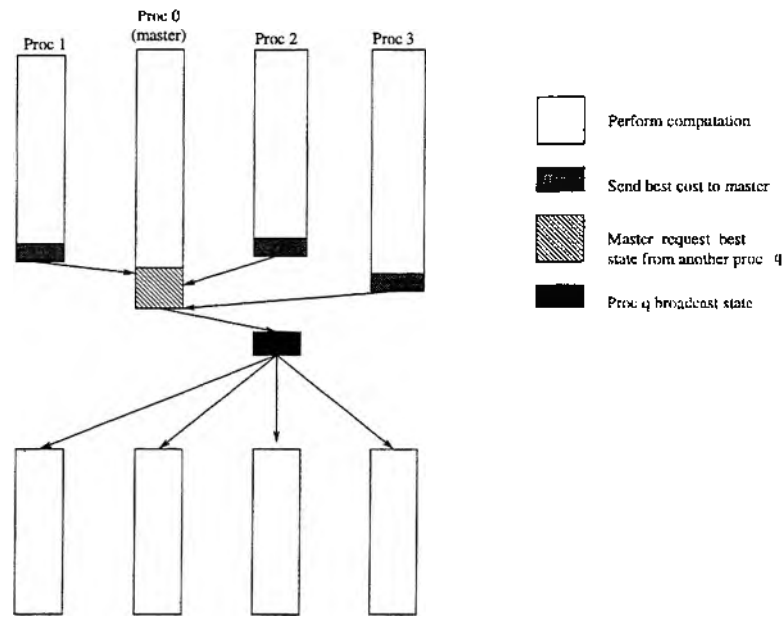


Figure 6.1 Synchronous multiple Markov chains

sor sends a cost measure of its current solution to processor 0. Processor 0 selects the best cost of the ones received from all other processors and compares it with the cost of the current global best solution. If it received a better cost than the global best, it sends a message to the processor which has the current best global solution. When that processor receives that message, it broadcasts the solution to all other processors. All the processors update their local solution with that received from the broadcast, and continues annealing from that solution. A barrier synchronization is implicitly achieved as a result of the above step. Figure 6.1 shows a typical scenario during the synchronous multiple Markov chain based approach. The detailed algorithm is given in Figure 6.2

6.2.2 Asynchronous multiple Markov chains

In this section we describe the asynchronous multiple Markov chain approach. Processor 0 is designated as the master processor which stores the global best solution. Each processor is assigned to perform N/P moves per temperature. At fixed intervals the slave processors send the cost of its current solution to the master processor. The master processor periodically probes for any messages from other processors (a natural place to do it is before every move). If there

```

SYNCHRONOUS MMC MASTER PROCESS()
1  while  $T \geq 0.1$  and not converged
2      do Perform N/P Moves
3          Get best cost from other procs
4          Broadcast overall best cost
5          if own solution is the best
6              then broadcast best solution
7              else get best solution from the processor having it
8              Update own solution  $T \leftarrow T \times 0.9$ 

```

```

SYNCHRONOUS MMC SLAVE PROCESS()
1  while  $T \geq 0.1$  and not converged
2      do Perform N/P moves
3          Send current solution to Master
4          Receive overall best cost from Master
5          if own solution is the best
6              then broadcast best solution
7              else get best solution from the processor having it
8              Update own solution  $T \leftarrow T \times 0.9$ 

```

Figure 6.2 Synchronous multiple Markov chains

are no messages, the master processor continues the annealing process. Otherwise, it receives the message and compares the cost with the cost of the global best solution and performs the following action. If the slave processor's solution is better than the current global best, the master processor sends a message to that processor requesting it to send its solution. The slave processor sends its solution to the master and the master updates the global best solution. The slave processor continues annealing from its current solution. If the cost of the current global best solution is better than the one received from the slave processor, the master sends the global best solution to the slave processor, which updates its current state with that received from the master and then continues from that point. Figure 6.3 shows the flow of messages for asynchronous multiple Markov chain based approach. The detailed algorithm is given in Figure 6.4

6.3 Experimental results

Experiments were performed for two high-level synthesis benchmarks using the 16-bit design library set given in table 4.1. The parallel algorithms were run on 1,2,4 and 8 processors on an IBM SP-2 message-passing multiprocessor. The following sections give the results for the latency-driven and power-driven schemes. In the tables, Q is given by $Cost_{seq}/Cost_{par}$ where $Cost_{seq}$ is the cost of the solution generated by the sequential algorithm and $Cost_{par}$ is that of the solution generated by the parallel algorithm. The speedups reported are relative to run times of the sequential algorithm on IBM SP-2. It can be seen that both synchronous and asynchronous schemes give almost the same quality of solution as the sequential approach. Asynchronous scheme was able to give better solution than the synchronous scheme in many cases. This is due to the fact that it has higher probability of avoiding being trapped in local minima when compared to the synchronous algorithm. In most cases, the speedups are close to linear. In some cases, we get super-linear speedup. This is due to the fact that simulated annealing can be considered to be a search process. When multiple threads of search are initiated in different parts of the solution space in parallel, the expected time for one of them to converge to the global minimum is smaller than that for a single processor executing those threads one after the other.

6.3.1 Experimental results for the latency-driven scheme

In this section, the experimental results are presented for the latency-driven scheme. The cost function used for comparison of quality of the solutions is given by $\alpha L + \beta A$ where L is the latency of the schedule, considering the interconnect delays, and A is the area of the floorplan, and α, β are user-defined positive constants. A lower value for the cost function indicates that the solution is of better quality. The quality and run times for the sequential algorithm for the latency driven scheme are shown in table 6.1. The quality of the solution generated by 2, 4 and 8 processors for different benchmarks are shown in tables 6.2 and 6.3, and their speedups in Figures 6.5, 6.6 and 6.7 for asynchronous and synchronous Markov chain schemes.

Table 6.1 Quality measures and run times for the sequential implementation for the latency driven algorithm

Circuit	Fabrication Technology	Latency (ns)	Area (sqmm)	Cost	Run time (seconds)
FIR 2-step	1.6(16 bit)	574.40	57.19	947.03	1.01×10^4
	1.2(16 bit)	430.69	31.75	542.95	1.01×10^4
FIR 3-step	1.6(16 bit)	753.70	46.87	669.97	6.63×10^3
	1.2(16 bit)	568.60	25.86	391.47	7.11×10^3
Elliptic 8-step	1.6(16 bit)	371.19	15.60	232.54	3.45×10^3
	1.2(16 bit)	284.90	8.91	141.65	3.56×10^3

Table 6.2 Quality comparisons for asynchronous Markov-chain approach for the latency driven algorithm

Circuit	Fabrication Technology	2 proc	4 proc	8 proc
FIR 2-step	1.6(16 bit)	1.000	1.000	1.000
	1.2(16 bit)	1.000	1.000	1.000
FIR 3-step	1.6(16 bit)	1.000	1.000	1.000
	1.2(16 bit)	1.000	1.000	1.000
Elliptic 8-step	1.6(16 bit)	1.000	0.990	0.988
	1.2(16 bit)	1.007	1.000	0.975

Table 6.3 Quality comparisons for synchronous Markov-chain approach for the latency driven algorithm

Circuit	Fabrication Technology	2 proc	4 proc	8 proc
FIR 2-step	1.6(16 bit)	1.000	1.000	1.00
	1.2(16 bit)	1.000	1.000	1.000
FIR 3-step	1.6(16 bit)	1.000	1.000	1.000
	1.2(16 bit)	1.000	1.000	1.000
Elliptic 8-step	1.6(16 bit)	0.978	0.979	0.979
	1.2(16 bit)	0.982	0.971	0.965

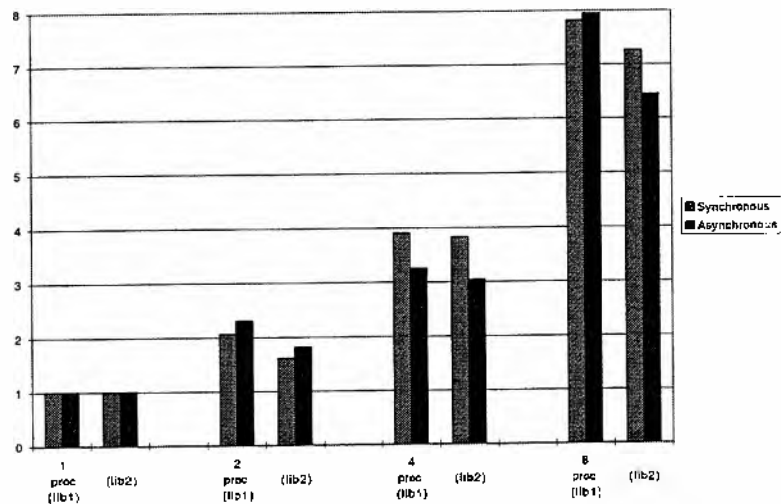


Figure 6.5 Speed-up comparisons for FIR-2step filter for the latency driven algorithm

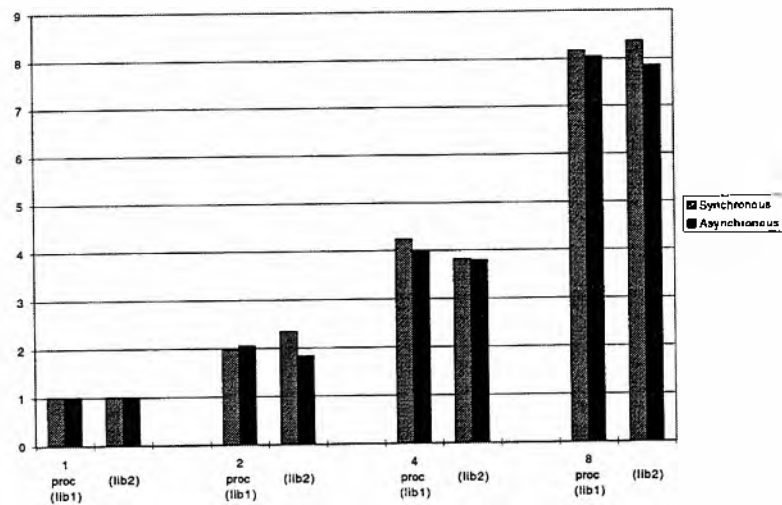


Figure 6.6 Speed-up comparisons for FIR-3step filter for the latency driven algorithm

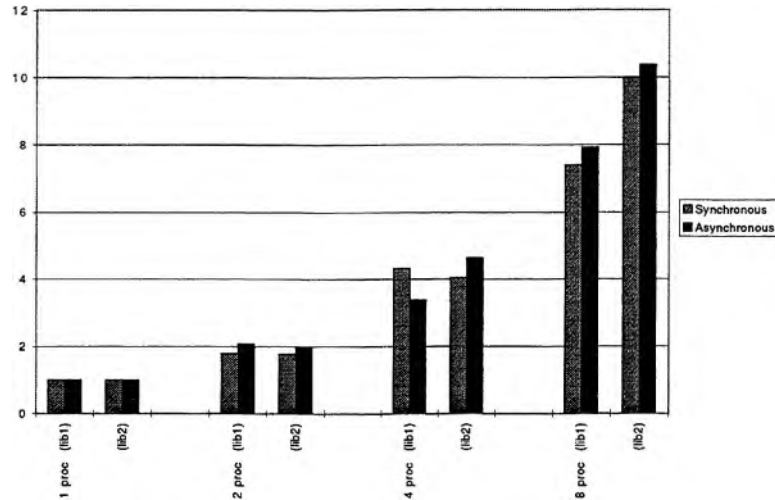


Figure 6.7 Speed-up comparisons for Elliptic filter for the latency driven algorithm

6.3.2 Experimental results for the power-driven scheme

In this section, the experimental results for the power-driven scheme are presented. The cost function used for comparison of quality of the solutions is given by $\alpha P + \beta L + \gamma A$ where L is the latency of the schedule, considering the interconnect delays, and A is the area of the floorplan, P is the interconnect energy estimate and α , β and γ are user-defined positive constants. As in the case of the latency-driven approach, a lower value for the cost function indicates that the solution is of better quality. Table 6.4 gives the results and run times for the sequential algorithm for the power-driven scheme. Tables 6.5 and 6.6 give the quality of solutions for the benchmark circuits for 2, 4 and 8 processors on an IBM SP-2, and Figures 6.8, 6.9 and 6.10 give the speedups for asynchronous and synchronous schemes.

6.4 Summary

With small device features in sub-micron technologies, since the interconnect delays play a dominant role in the cycle time, it is important to consider the effect of physical design on high-level synthesis. We have presented parallel algorithms which combines physical design and high level synthesis. Experimental results show that the approach is quite effective in obtaining

Table 6.4 Quality measures and run times for the sequential implementation of the power driven algorithm

Circuit	Fabrication Technology	Latency (ns)	Area (sqmm)	Energy Est	Cost	Run time (seconds)
FIR 2-step	1.6(16 bit)	650.20	55.76	107.90	49.90	8.576×10^3
	1.2(16 bit)	475.30	31.35	83.00	28.37	8.307×10^3
FIR 3-step	1.6(16 bit)	1001.68	47.20	110.7	64.75	3.88×10^3
	1.2(16 bit)	735.50	25.84	71.39	30.96	3.62×10^3
Elliptic 8-step	1.6(16 bit)	418.10	15.49	53.79	19.9	3.65×10^3
	1.2(16 bit)	564.80	8.43	39.00	11.67	4.11×10^3

Table 6.5 Quality comparisons for asynchronous Markov-chain approach for the power driven algorithm

Circuit	Fabrication Technology	2 proc	4 proc	8 proc
FIR 2-step	1.6(16 bit)	0.92	1.02	1.02
	1.2(16 bit)	1.04	1.00	1.04
FIR 3-step	1.6(16 bit)	1.11	1.20	1.12
	1.2(16 bit)	0.97	1.05	0.97
Elliptic 8-step	1.6(16 bit)	0.89	1.01	0.97
	1.2(16 bit)	1.06	1.09	1.05

Table 6.6 Quality comparisons for synchronous Markov-chain approach for the power driven algorithm

Circuit	Fabrication Technology	2 proc	4 proc	8 proc
FIR 2-step	1.6(16 bit)	0.94	1.02	0.93
	1.2(16 bit)	1.03	0.97	1.05
FIR 3-step	1.6(16 bit)	1.05	1.20	0.90
	1.2(16 bit)	1.04	0.99	0.95
Elliptic 8-step	1.6(16 bit)	0.93	0.92	0.85
	1.2(16 bit)	1.09	1.05	1.09

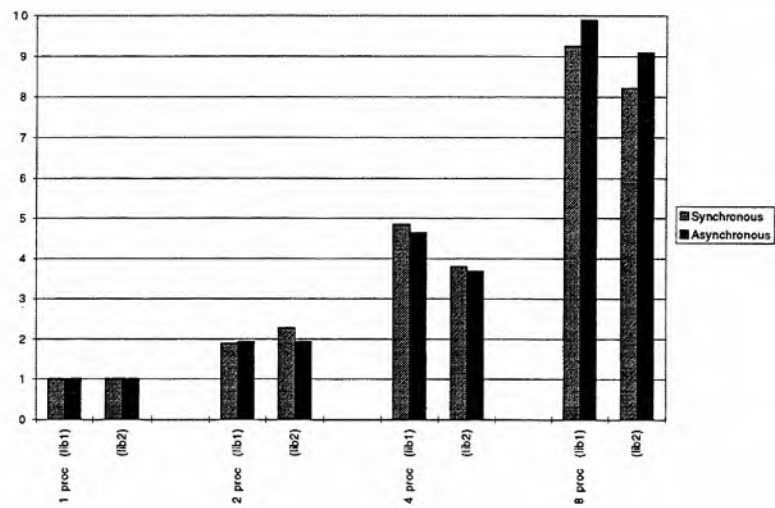


Figure 6.8 Speed-up comparisons for FIR-2step filter for the power driven algorithm

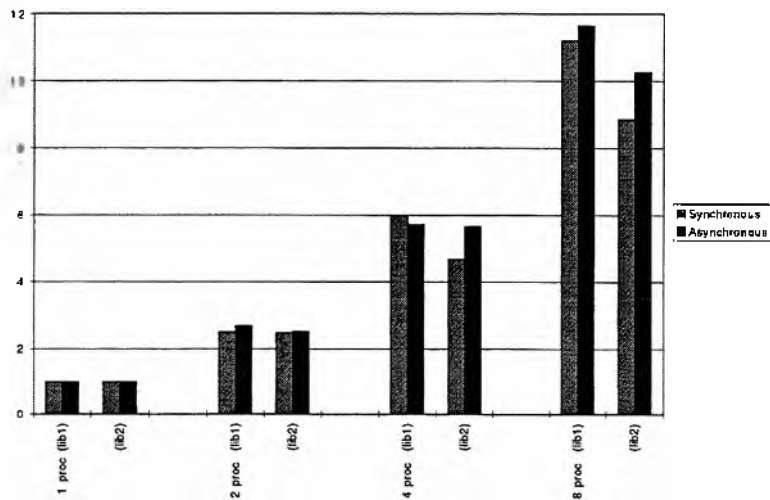


Figure 6.9 Speed-up comparisons for FIR-3step filter for the power driven algorithm

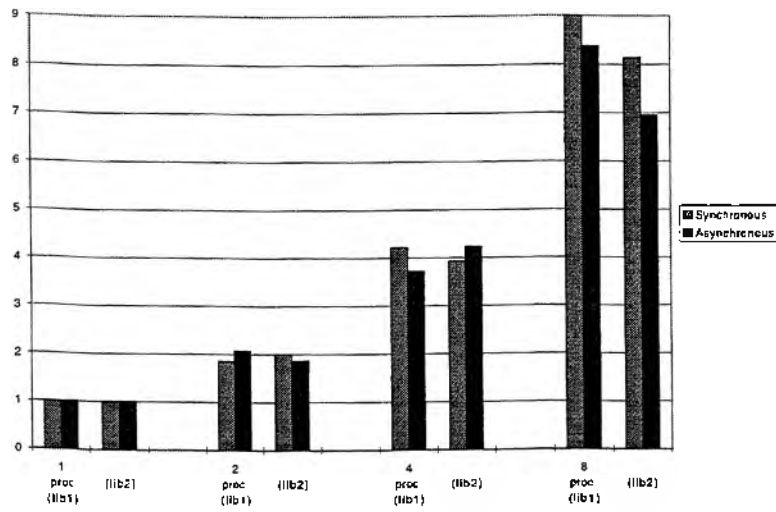


Figure 6.10 Speed-up comparisons for Elliptic filter for the power driven algorithm

solutions of almost the same or better quality as the sequential algorithm, while giving close to linear speedups.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Thesis summary

In the first part of this thesis, we presented novel algorithms for force directed scheduling. First, an extension of the force directed scheduling problem which naturally handles loops and conditionals by coming up with a scheme of scheduling *hierarchical* signal flow graphs was developed. Second, three novel parallel algorithms for the scheduling problem was presented. The parallel algorithms are portable across a wide range of parallel platforms. The results are reported on an SGI Origin shared memory multiprocessor, and an IBM SP-2 distributed memory multicomputer. While some parallel algorithms for VLSI CAD reported by earlier researchers have reported on loss of qualities of results, the parallel algorithms presented in this thesis produced exactly the same results as the sequential algorithms on which they are based.

In the second part of the thesis, a new algorithm which combines physical design and high level synthesis was presented. Experimental results show that the approach is quite effective in arriving at solutions of better quality in comparison to a traditional two-step approach of performing high-level synthesis and floorplanning one after the other. A combined scheduling, binding and power-driven floorplanning algorithm was also presented, which is able to reduce the interconnect energy dissipation considerably.

Finally, two parallel algorithms which combines physical design and high level synthesis were presented. Experimental results show that the approach is quite effective in obtaining so-

lutions of almost the same or better quality as the sequential algorithm, while giving close to linear speedups.

7.2 Future work

There are several ways of extending the algorithms presented in this thesis. They are as follows

- The quality and run times of the simultaneous scheduling, binding and floorplanning depends very much on the quality and run time of the floorplanning algorithm. An area of future work is to arrive at better floorplanning algorithms which can also run faster.
- Inclusion of features to perform pipelining, multi-cycle operations, and to incorporate schemes to tradeoff latency and area using different types of functional units, e.g. carry-lookahead adder versus ripple-carry adder.
- Handle signal-flow graphs containing conditionals and loops
- Even though simulated annealing is effective in arriving at solutions of good quality, it also requires large run times. An area of research is to evaluate alternate frameworks of performing combined scheduling, binding and floorplanning, other than simulated annealing.

REFERENCES

- [1] J. Crenshaw and M. Sarrafzadeh, "Accurate High Level Datapath Power Estimation," in *European Design and Test Conference*, 1997.
- [2] P. Prabhakaran and P. Banerjee, "Simultaneous Scheduling, Binding and Floorplanning in High-level Synthesis," *International Conference on VLSI Design*, 1998.
- [3] P. Prabhakaran and P. Banerjee, "Simultaneous Scheduling, Binding and Floorplanning in High-level Synthesis," *Technical Report CRHC-97-16*, 1997.
- [4] A. Raghunathan and N. K. Jha, "An Iterative Improvement Algorithm for Low Power Data Path Synthesis," in *International Conference on Computer-Aided Design*, 1995.
- [5] A. Raghunathan, and N. K. Jha, "Behavioral Synthesis for Low Power," in *International Conference on Computer Design*, 1994.
- [6] A. Das and R. Karri, "Simultaneous Scheduling and Binding for Power Minimization During Microarchitecture Synthesis", in *International Symposium on Low Power Design*, 1995.
- [7] K. Chao and D. F. Wong, "Low Power Considerations in Floorplan Design", in *International Workshop on Low-Power Design*, 1994.
- [8] P. E. Landman and J. M. Rabaey, "Power Estimation for High-level Synthesis", in *European Design Automation Conference*, 1993.
- [9] A. Chandrakasan, M. Potkonjak, J. Rabaey and R. Brodersen, "HYPER-LP: A system for power minimization using architectural transformations", in *International Conference on Computer-Aided Design 1992*.
- [10] M. Pedram, "Power Minimization in IC Design: Principles and Applications", in *ACM Transactions on Design Automation of Electronic Systems*, 1996.
- [11] C. Svenson and D. Liu, "A Power Estimation tool and Prospects of Power Savings in CMOS VLSI chips", in *International Workshop on Low-Power Design*, 1994.
- [12] S. Devadas and A. R. Newton, "Algorithms for Hardware Allocation in DataPath Synthesis," *IEEE trans. on CAD*, vol. 8, No. 7, 1989
- [13] D. F. Wong and C. L. Liu, "A New Algorithm for FloorPlan Design," in *Proc. of 23rd DAC*, 1986.

- [14] T. Yamanouchi, K. Tamakashi and T. Kambe, "Hybrid Floorplanning Based on Partial Clustering and Module Restructuring", *International Conference on Computer Aided Design*, 1996.
- [15] Y. Fang and D. F. Wong, "Simultaneous Functional Unit Binding and Floorplanning," *Proc. of ICCAD*, pp. 317-321, 1994.
- [16] V. Moshnyaga, H. Mori, H. Onodera, K. Tamary, "Layout-Driven Module Selection for Register-Transfer Synthesis of Sub-micron ASICs," *Proc. of ICCAD*, pp. 100-103, 1993.
- [17] J. Weng and A. C. Parker, "3D Scheduling : High-Level Synthesis with Floorplanning," *Proc. of 28th DAC*, pp. 668-673, 1991.
- [18] D. W. Knapp, "Fasolt: A Program for Feedback-Driven Data-Path Optimization," *IEEE Trans. on CAD*, vol. 11, No. 6, pp. 677-695, 1992.
- [19] M. C. McFarland, "Using Bottom-Up Design Techniques in the Synthesis of Digital Data-Path Optimization," *Proc. of 23rd DAC*, pp. 474-480, 1986.
- [20] F. Brewer and D. Gajski, "Chippe: A system for constraint driven Behavioral Synthesis," *IEEE Trans. on CAD*, Vol. 9, No. 7, pp. 681-695, 1990.
- [21] E. Girczyc, "Automatic Generation of Microsequenced Data Paths to Realize ADA Circuit Descriptions," *PhD thesis, Carleton University, July 1984*
- [22] W. Dai and E. S. Kuh, "Simultaneous Floor Planning and Global Routing for Hierarchical Building Block Layout," *IEEE trans on CAD*, 1987
- [23] D. P. La Potin, and S. W. Director, "Mason: A Global Floorplanning Approach for VLSI Design," *IEEE trans on CAD*, 1986
- [24] L. StockMeyer, "Optimal Orientations of Cells in Slicing Floorplan Designs", *Information and Control*, Vol 59,(1983), 510-522.
- [25] R. H. J. M. Otten, "Automatic Floorplan Design," *Proc. DAC 82*, pp 261-267.
- [26] R. H. J. M. Otten, "Efficient Floorplan Optimization," *Proc. ICCD 83*, pp 96-98.
- [27] D. F. Wong and P. S. Sakhamuri, "Efficient Floorplan Area Optimization", *Design Automation Conference* 1989
- [28] T. W. Wang and D. F. Wong, "An Optimal Algorithm for Floorplan Area Optimization", *Design Automation Conference* 1990
- [29] R. Nair, C. L. Berman, P. S. Hauge and E. J. Yoffa, "Generation of Performance Constraints for Layout," *IEEE Trans. Computer-Aided Design*, 1989.
- [30] A. Mujumdar, M. Rim, R. Jain and R. De Leone, "BITNET: An Algorithm for Solving The Binding Problem," *International Conference on VLSI Design*, pp. 668-673, 1991.

- [31] H. Jang and B. Pangrle, "A Grid-Based Approach for Connectivity Binding with Geometric Costs," *Proc. of ICCAD*, pp. 94-99, 1993.
- [32] H. B. Bakoglu, "Circuits, Interconnections and Packaging for VLSI," pp. 194-225, 1990.
- [33] J. Cong, Z. Pan, L. He, C. Koh and K. Khoo, "Interconnect Design for Deep Submicron ICs," in *International Conference on Computer-Aided Design 1997*
- [34] Semiconductor Industry Association, "National Technology Roadmap for Semiconductors," 1994.
- [35] D. Zhou, F. P. Preparata and S. M. Kang, "Interconnection Delay in Very High-Speed VLSI," *IEEE Trans. on Circuits and Systems*, Vol 38, No. 7, pp.779-790, 1991.
- [36] S. Y. Lee and K. G. Lee, "Asynchronous communication of multiple Markov chains in parallel simulated annealing," in *Proceedings of the International Conference on Parallel Processing*, Aug.1992.
- [37] E. H. L. Aarts, F. M. J.de Bont, E. H. A. Habers, and P. J. M. van Laarhoven, "Parallel implementations of the statistical cooling algorithm," *Integration, the VLSI journal*, vol. 4 Sept. 1986.
- [38] J. A. Chandy and P. Banerjee, "Parallel Simulated Annealing Strategies for VLSI Cell Placement," *Proceedings of the 9th International Conference on VLSI Design*, Jan. 1996
- [39] P. Banerjee, "Parallel Algorithms for VLSI Computer-aided Design Applications," *Englewood-Cliffs, NJ:Prentice Hall*, 1994.
- [40] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICS," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 6, pp. 661-679, 1989.
- [41] V. Kumar, A. Grama, A. Gupta and G. Karypis, "Introduction to Parallel Computing", *Benjamin/Cummings Publishing Company*, 1994.
- [42] W. Gropp, E. Lusk, and A. Skellum, "Using MPI: Portable Parallel Programming with the Message Passing Interface", *MIT Press*, 1994.
- [43] R. Haupt, "A survey of priority-rule based scheduling," *OR Spektrum*, vol. 11, no. 1, pp. 3-16, 1989.
- [44] M. C. McFarland, "Using bottom-up design techniques in the synthesis of digital hardware from abstract behavioral descriptions," *Proc. 23rd DAC*, Las Vegas, NV, June 1986, pp. 474-480.
- [45] M. C. McFarland, A. C. Parker, and R. Camposano, "The high-level synthesis of digital systems," *Proc. IEEE*, vol. 78, no. 2, pp. 301-318, 1990.
- [46] C. H. Gebotys and M. I. Elmasry, "A global optimization approach for architectural synthesis," in *Proc. ICCAD*, Santa Clara, CA, Nov. 1990, pp. 258-261.

- [47] C. T. Hwang, T. H. Lee, and Y. C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE Trans. CAD*, vol. 10, no. 4, pp. 464-475, 1991.
- [48] R. J. Cloutier and D. E. Thomas, "The combination of scheduling, allocation, and mapping in a single algorithm," *Carnegie Mellon Univ. Res. Rep CMUCAD-90-17*, May 1990.
- [49] T. Kim, J. W. S. Liu, and C. L. Liu, "A scheduling algorithm for conditional resource sharing," in *Proc ICCAD*, Santa Clara, CA, Nov 1991, pp. 84-87.
- [50] A. Oláh, S. H. Gerez, and S. M. Heemstra de Groot, "Scheduling and allocation for the high-level synthesis of DSP algorithms by exploitation of transfer mobility", in *Proc. CompEuro*, Delft, The Netherlands, 1992, pp. 145-150.
- [51] L. Stok, "Architectural synthesis and optimization of digital systems," Ph. D dissertation, Eindhoven Univ. of Technol., Eindhoven, The Netherlands. Apr. 1990
- [52] Wim F. J. Verhaegh, Paul E. R. Lippens, Emile H. L. Aarts, Jan H. M. Korst, Jef L. van Meerbergen, Albert van der Werf, "Improved force-directed scheduling in high-throughput digital signal processing", in *IEEE trans on CAD*, vol. 14, no. 8, August 1995.
- [53] J. Roy, N. Kumar, R. Dutta, R. Vemuri, "DSS: A distributed high-level synthesis system", in *IEEE design and test of computers*, June 1992.
- [54] R. Lipsett, C. Schaefer and C. Ussery, "VHDL: Hardware Description and Design", in *Kluwer Academic Publishers, Boston, MA*, 1991.
- [55] D. Thomas and P. Moorby, "The Verilog Hardware Description Language", in *Kluwer Academic Publishers, Boston, MA*, 1991.
- [56] D. Ku and G. DeMicheli, "High-level Synthesis of ASICs under Timing and Synchronization Constraints", *Kluwer Academic Publishers, Boston, MA*, 1992.
- [57] B. Ramkumar and P. Banerjee, "ProperCAD: A portable object-oriented parallel environment for VLSI CAD," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 829-842, July 1994.
- [58] S. Kim, B. Ramkumar, J. Chandy, S. Parkes, and P. Banerjee, "ProperPLACE: A portable parallel algorithm for standard cell placement. *Proc. 8th Int. Parallel Processing Symp. (IPPS-94)*, Apr. 1994.
- [59] R. Brouwer and P. Banerjee, "PHIGURE: A parallel hierarchical global router," in *Proc. 27th Design Automation Conference.*, pp. 591-597, June 1990.
- [60] K. P. Belkhale and P. Banerjee. "Parallel algorithms for VLSI circuit extraction," *IEEE Trans. CAD*, 10(2):604-618, May 1991.
- [61] J. S. Rose, W. M. Snelgrove, and Z. G. Vranesic, "Parallel cell placement algorithms with quality equivalent to simulated annealing," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 387-396, Mar. 1988.

- [62] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, "A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells", in *IEEE Trans. Computer-aided Design Integrated Circuits Systems*, pages 838-847, Sept. 1987.
- [63] A. Casotto and A. Sangiovanni-Vincentelli, "Placement of standard cells using simulated annealing on the Connection Machine," in *Digest of Papers, International Conference on Computer-Aided Design*, (Santa Clara, CA), pp. 350-353, Nov. 1987.
- [64] S. Patil, P. Banerjee, and T. Patel, "Parallel Test Generation for Sequential Circuits on General Purpose Multiprocessors," in *Proc. 28th Design Automation Conf (DAC-91)*, June. 1991.
- [65] P. Agrawal, V. D. Agrawal, J. Villodo, "Sequential Circuit Test Generation on a Distributed System," in *Proc 30th Design Automation Conf, (DAC-93)*, June. 1993.
- [66] K. De, B. Ramkumar and P. Banerjee, "ProperSYN: A portable parallel algorithm for logic synthesis," in *IEEE Trans. Computer Aided Design(ICCAD-92)*, Nov. 1992.
- [67] B. Ramkumar and P. Banerjee, "ProperTEST: A portable parallel test generator for sequential circuits," to appear in *IEEE Trans. on CAD*, 1996.
- [68] K. MacPherson, "Parallel algorithms for layout verification," *MS thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana-Champaign, Technical report*, 1995.
- [69] K. De, J. A. Chandy, S. Roy, S. Parkes and P. Banerjee, "Parallel algorithms for logic synthesis using the MIS approach," *Proceedings of IPPS*, Santa Barbara, April 1995.
- [70] S. Parkes, P. Banerjee, and J. Patel, "A parallel algorithm for fault simulation based on PROOFS," *Proceedings of the ICCD*, Austin, TX, Oct. 1995.
- [71] S. Parkes, P. Banerjee, and J. Patel "ProperHITEC: A portable, parallel, object-oriented approach to sequential test generation," in *Proc. 31st Design Automation Conf.*, June 1994.
- [72] R. Galivanche and S. M. Reddy, "A Parallel PLA Minimization Program," in *Proc. Design Automation Conf.*, pages 600-607, 1987.
- [73] H. D. Hachtel and P. H. Moceyunas, "Parallel Algorithms for Boolean Tautology Checking," in *Proc. Int. Conf. Computer-aided Design*, pp. 422-425, Nov. 1987.

VITA

Pradeep Kumar Prabhakaran received the B. Tech degree in Computer Science from the University of Kerala, India in 1990. He received the M. E degree in Computer Science from the Indian Institute of Science, Bangalore, India in 1992 and an M. S. degree in Computer Science from the Ohio State University in 1993. In the summer of 1997, he was an intern at the IBM T. J. Watson research center, Yorktown Heights, New York. He will receive the Ph. D degree in 1998. His research interests are in High-level Synthesis, Logic Synthesis, Physical Design, Parallel Processing and Computer Graphics.