

POWER AND MEMORY OPTIMIZATION TECHNIQUES IN EMBEDDED SYSTEMS DESIGN

A Dissertation
Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Electrical and Computer Engineering

by
Atef Khalil Allam
B.S., Assiut University, 1994
M.S., Assiut University, 1998
August 2005

*To my LORD
WHO created me and created every thing*

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor Dr. Ramanujam for his support, encouragement, and technical advice during the course of this work. His valuable hints when I am stuck will always be appreciated. I would also like to thank Dr. R. Vaidyanathan, Dr. D. Carver, Dr. J. Tyler, Dr. J. Trahan, Dr. J. Hong, and Dr. A. Lisan for serving on my committee.

Finally, I would like to express my gratitude to my parents, my eldest brother Eng. El Badry, and my wife for their marvelous support that makes this work eventually possible.

TABLE OF CONTENTS

DEDICATION	ii
ACKNOWLEDGEMENTS	iii
LIST OF TABLES	vii
LIST OF FIGURES	ix
LIST OF NOTATIONS	xi
ABSTRACT	xii
CHAPTER	
1. INTRODUCTION	1
1.1 Embedded Systems	1
1.2 Low-Power Design	2
1.3 High-Level Synthesis	4
1.4 Motivating Example	5
1.5 Memory System	5
1.6 Related Work	8
1.6.1 High-Level Synthesis	8
1.6.2 Memory System	11
1.7 Dissertation Outline	12
2. DSE FOR PEAK AND AVERAGE POWER OPTIMIZATION USING MULTIPLE SUPPLY-VOLTAGES	13
2.1 Problem Definition	13
2.2 Exact Solution	15
2.2.1 TCS Problem	15
2.2.2 TRCS Problem	15
2.3 Iterative LP Relaxation	18
2.4 Power- Resources Saving	19
2.4.1 Time Complexity of Power-Resources Saving	20
2.5 Experimental Results	22
2.6 Chapter Summary	26
3. MODIFIED FORCE-DIRECTED SCHEDULING FOR PEAK AND AVERAGE POWER OPTIMIZATION USING MULTIPLE SUPPLY- VOLTAGES	27
3.1 Problem Definition	27
3.2 Basic Force-Directed Scheduling	28

3.3	Modified Power-Force-Directed Scheduling (MPFDS)	29
3.3.1	Time Complexity of MPFDS	32
3.4	Power-Resources Saving	33
3.5	Experimental Results	34
3.6	Chapter Summary	36
4.	MODULE SELECTION AND SCHEDULING UNIFICATION FOR PEAK AND AVERAGE POWER OPTIMIZATION	37
4.1	Problem Definition	37
4.2	Exact Solution	39
4.2.1	TCMSS Problem	39
4.2.2	TACMSS Problem	40
4.3	Iterative LP Relaxation	41
4.4	Power-Area Saving	44
4.5	Experimental Results	46
4.6	Chapter Summary	50
5.	SIMULTANEOUS PEAK AND AVERAGE POWER OPTIMIZATION IN SYNCHRONOUS SEQUENTIAL CIRCUITS USING RETIMING AND MULTIPLE SUPPLY-VOLTAGES	51
5.1	Synchronous Circuit Representation	51
5.2	Basics of Retiming	52
5.3	Unifying Retiming and Power Optimization	53
5.4	The Two-Stage Power Optimization Solution	55
5.4.1	Motivating Example	55
5.4.2	Power-Oriented Retiming	56
5.4.3	Peak and Average Power Optimization for DAGs	58
5.5	Extensions	59
5.6	Experimental Results	60
5.7	Chapter Summary	63
6.	DYNAMIC MEMORY USAGE OPTIMIZATION	65
6.1	Preliminaries	66
6.2	Evaluation Order and Performance-Constrained Evaluation (EOPCE)	67
6.3	Memory-Constrained Evaluation (MCE)	68
6.4	Example for PCE Problem	69
6.5	Memory Force-Directed Scheduling for Memory Minimization (MFDS)	71
6.6	Experimental Results	74
6.7	Chapter Summary	78
7.	LOOP FUSION USING ILP FOR MEMORY MINIMIZATION	79
7.1	Problem Definition and Formulation	79
7.1.1	Modified Fusion Graph	80
7.2	Legality of Fusion	82
7.3	ILP Formulation	84
7.3.1	Fusion Constraints	84

7.3.2 Fusion Objective Function	84
7.3.3 Objective Function Linearization	85
7.4 Example	87
7.5 Fusion in the Case of DAGs	88
7.6 Experimental Results	94
7.7 Chapter Summary	95
8. CONCLUSION AND FUTURE WORK	96
8.1 Contributions of This Work	96
8.2 Future Work	99
8.2.1 Power Optimization for Control-Flow Applications	99
8.2.2 Tight Time-Frames in the MILP Formulation for Unified Peak Power and Retiming	100
8.2.3 Modeling Leakage Power	101
8.2.4 Incorporating Disk Access Cost with Fusion	101
REFERENCES	102
VITA	112

LIST OF TABLES

2.1 DFG nodes and their associated 0-1 variables for Figure 2.3-(a)	17
2.2 Gradually iterative LP solution for the DFG in Figure 2.3-(a)	20
2.3 Module library for MVS	24
2.4 HAL under TCS	24
2.5 ARF under TCS	24
2.6 EWF under TCS	24
2.7 HAL [3*, 3+] under TRCS	25
2.8 HAL [2*, 2+] under TRCS	25
2.9 ARF [6*, 3+] under TRCS	25
2.10 ARF [4*, 2+] under TRCS	25
2.11 EWF [3*, 5+] under TRCS	26
2.12 EWF [2*, 3+] under TRCS	26
3.1 Peak and average power results for ILP solution and the two-phase heuristic	35
3.2 Comparison between power-area saving with and without resource constraints	35
4.1 DFG nodes and their associated 0-1 variables for Figure 4.4-(a)	43
4.2 Gradually iterative LP solution for the DFG in Figure 4.4-(a)	44
4.3 Module library for module selection	47
4.4 HAL under TCMSS	47
4.5 ARF under TCMSS	47
4.6 EWF under TCMSS	48
4.7 FDCT under TCMSS	48
4.8 HAL 120 under TACMSS	48

4.9	HAL 100 under TACMSS	48
4.10	ARF 260 under TACMSS	49
4.11	ARF 170 under TACMSS	49
4.12	EWF 120 under TACMSS	49
4.13	EWF 100 under TACMSS	49
4.14	FDCT 315 under TACMSS	50
4.15	FDCT 280 under TACMSS	50
5.1	Peak and average power under varying objectives for different benchmarks	62
5.2	Power results of both power-oriented and unit-weight retiming	63
6.1	Memory sizes, ASAP and ALAP for the DFG in Figure 6.1	69
6.2	Output memory schedule for the ILP formulation in Figure 6.2	70
6.3	Performance constrained evaluation with unit delay	76
6.4	Performance constrained evaluation	77
6.5	ILP memory constrained evaluation	77
7.1	Potential fusion edges and their associated 0-1 variables	87
7.2	The associated paths and their pairs of loop indices for the constraints in Figure 7.5	88
7.3	Memory usage of some benchmarks after fusion	95

LIST OF FIGURES

1.1	Embedded system structure	2
1.2	Effect of peak power consideration	6
2.1	Flow-chart for the DSE using multiple voltage scheduling.	14
2.2	Procedure of the multiple supply-voltages scheduling.	15
2.3	Example of the MILP formulation	17
2.4	Complete listing of constraints and the objective for the DFG in Figure 2.3-(a)	17
2.5	Iterative LP procedures	19
2.6	Power-resources saving algorithm	21
2.7	An example demonstrating the effect of power-resources saving procedure	22
3.1	Potential schedule of sample operation	31
3.2	MPFDS algorithm	32
3.3	Effect of power-resources saving procedure	33
4.1	Flow-chart for the scheduling and module selection process	38
4.2	Unified module selection and scheduling for power minimization	39
4.3	Iterative LP procedures	42
4.4	A DFG example	43
4.5	Complete listing of constraints and the objective for the DFG in Figure 4.4-(a)	44
4.6	Power-area saving algorithm	45
4.7	Power-area saving procedure example	46
5.1	An example for SDFG	52
5.2	Results of retiming Figure 5.1 with two different retiming vectors for clock period = 4	56
5.3	Procedure for extracting the minimal power schedule	60

6.1	Sample DFG for memory evaluation	67
6.2	Complete listing of all constraints of the example at Table 6.1	70
6.3	Portion of a sample DFG	72
6.4	Procedure for probability computation	72
6.5	Sample scheduling for Figure 6.3	74
6.6	Flow chart for DSE strategy	75
6.7	Sample examples for memory evaluation	76
7.1	An example of multi-dimensional integral	79
7.2	Fusion graph for operation-minimal sequence in Figure 7.1	81
7.3	Illegal fusion configurations	82
7.4	Legal fusion configurations	82
7.5	Complete ILP formulation for the fusion graph in Figure 7.2-(a)	87
7.6	An example of multi-dimensional summation with common sub-expression	88
7.7	Examples of illegal fusion graphs for a DAG	89
7.8	An example of common sub-expression with unifiable sets of index-variables	90
7.9	Illegal fusion graphs and their corresponding legal graphs for Figure 7.8	92
7.10	Test examples	94

LIST OF NOTATIONS

$p(i, v)$	Power consumed by operation i using voltage level v .
$d(i, v)$	Delay (in # of control steps) of operation i using voltage level v .
P_j	Power consumed by all functional units or module instances at step j .
P_{peak}	Maximum power consumed by all functional units at any step
FU_k	Functional unit of type k
M_k	Maximum number of functional units of type k
$cstep$	Control step
λ	Total number of control steps.
$ps(i)$	Predecessors/successors of an operation i .
$p(i, m)$	Power consumed by operation i executed on module type m .
$d(i, m)$	Delay (in # of control steps) of operation i when executed on module type m .
r_m	Number of instances of module type m .
$area(m)$	Area cost of module type m .
A	Total area constraint.
$M(i)$	Memory size needed for data object i .
$D(i)$	Execution time (in number of time units) needed for data object i .
$ex-M(i)$	Extended memory size needed for data object i and all of its children.
mem_j	Total memory usage by all active nodes at step j .
$mem_constraint$	Maximum memory space allowed
$R(i)$	Time-frame of node i , which is the set of time-steps that start at its earliest time, <i>ASAP</i> , and end at its latest time, <i>ALAP</i>

ABSTRACT

Embedded systems incur tight constraints on power consumption and memory (which impacts size) in addition to other constraints such as weight and cost. This dissertation addresses two key factors in embedded system design, namely minimization of power consumption and memory requirement. The first part of this dissertation considers the problem of optimizing power consumption (peak power as well as average power) in high-level synthesis (HLS). The second part deals with memory usage optimization mainly targeting a restricted class of computations expressed as loops accessing large data arrays that arises in scientific computing such as the coupled cluster and configuration interaction methods in quantum chemistry.

First, a mixed-integer linear programming (MILP) formulation is presented for the scheduling problem in HLS using multiple supply-voltages in order to optimize peak power as well as average power and energy consumptions. For large designs, the MILP formulation may not be suitable; therefore, a two-phase iterative linear programming formulation and a power-resource-saving heuristic are presented to solve this problem. In addition, a new heuristic that uses an adaptation of the well-known force-directed scheduling heuristic is presented for the same problem. Next, this work considers the problem of module selection simultaneously with scheduling for minimizing peak and average power consumption. Then, the problem of power consumption (peak and average) in synchronous sequential designs is addressed. A solution integrating basic retiming and multiple-voltage scheduling (MVS) is proposed and evaluated. A two-stage algorithm namely power-oriented retiming followed by a MVS technique for peak and/or average power optimization is presented.

Memory optimization is addressed next. Dynamic memory usage optimization during the evaluation of a special class of interdependent large data arrays is considered. Finally, this dissertation develops a novel integer-linear programming (ILP) formulation for static memory optimization using the well-known fusion technique by encoding of legality rules for loop fusion

of a special class of loops using logical constraints over binary decision variables and a highly effective approximation of memory usage.

CHAPTER 1

INTRODUCTION

1.1 Embedded Systems

Embedded systems nowadays are present in most of the electronic devices and instruments used in daily life; they can be found in consumer electronic devices (calculator, digital cameras, cell phones, etc.), office equipment (printers, copy machines, fax machines, etc.), home appliances (microwave ovens, washing machines, alarms, etc.), and automobiles (cruise control, transmission control, fuel injection, etc.).

Loosely speaking, an embedded system is any computing system other than a desktop computer [VG02]. An embedded system typically consists of four main components: an embedded processor, synthesized circuit for dedicated hardware units, memory, and I/O interface. All of these are typically implemented in one chip constituting what is called a *system-on-chip* (SOC) as shown in Figure 1.1.

Embedded systems exhibit certain characteristics that distinguish them from other computing systems. These characteristics are:

- **Single function:** An embedded system usually executes a certain task (or program) repeatedly.
- **Real-time operation:** Time constraint is very crucial in execution of tasks on embedded systems. Even a small execution delay might cause a serious malfunctioning or total failure.
- **Tight constraints:** Because of the nature of embedded systems, their design metrics such as size, performance and power impose tight constraints.

Embedded systems are specified in different levels of abstractions. Gajski's famous Y-chart [GK83] has identified different views of these abstraction levels and the relationship among

them. Typical levels of abstraction are the system, the behavioral, the register-transfer (RT), the logic, and the physical level. Each level of specification is a refinement of the level above it. At the system level, the design is described as a set of interacting subsystems (processes) to be mapped to either hardware or software components. These subsystems can be implemented using processors (software), application-specific IC's (ASICs), memories and dedicated hardware. At the behavioral level, each subsystem is specified in its algorithmic (or functional) form. At the RT level, the system is specified as a collection of communicating register transfer logic (RTL) units such as ALUs, registers, and multiplexers. The logic level specification is the hardware implementation of the logic functions given as a netlist of logic gates and flipflops. The physical level description is the physical implementation given as a netlist of transistors, capacitors, and resistors on a board. Register-transfer (RT), logic, and physical levels belong to the hardware side. Module level and block level specifications are the typical levels of abstraction on the software side.

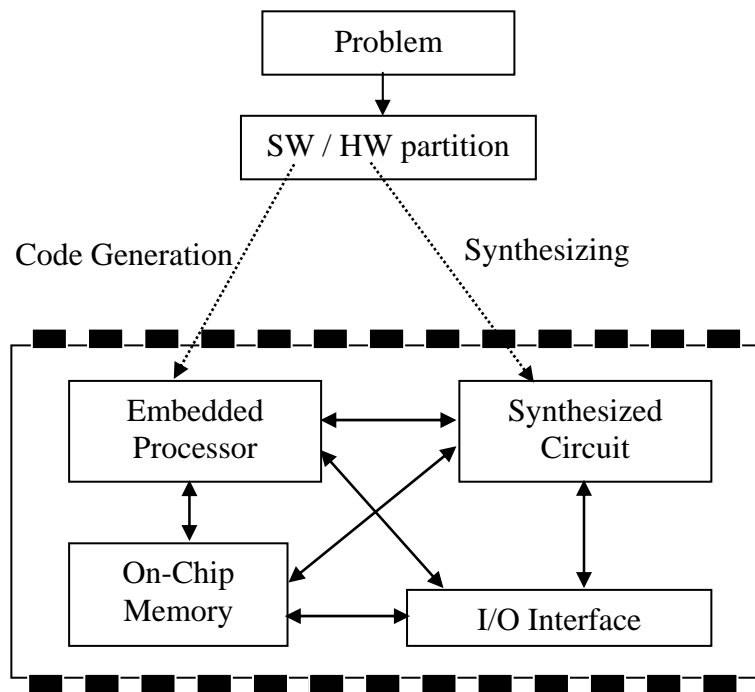


Figure 1.1: Embedded system structure.

1.2 Low-Power Design

The usual design metrics of embedded systems are performance, size, testability, and power. Although conventional design metrics such as performance, size and testability are important, the

most critical design metric nowadays is power. The demand for long-life batteries within tolerable size and weight and the reliability of integrated circuits are the main factors that dictate power-aware design of embedded systems. Reliability of integrated circuits is tightly related to the peak and average power consumption. It has been estimated that every 10°C increase in operating temperature causes component failure rate to approximately double [RD98]. Since most of the power consumed by the integrated circuits is dissipated in the form of heat, complex cooling techniques are needed to maintain the operating temperature of the circuits within the normal level. Moreover, heat dissipation is a limiting factor in increasing the number of transistors in a chip. These and other factors such as cost and size have driven the low power design to be a critical issue. Using low power design techniques helps in solving these problems and alleviates chip failure and system operation degradation.

The main sources of power dissipation in CMOS circuits are the capacitive switching power, P_{sw} , the short-circuit power, P_{sc} , and the power consumption due to leakage current, $P_{leakage}$. These power consumption sources are summarized in the following equation [RD98]:

$$P_{av} = P_{sw} + P_{sc} + P_{leakage}. \quad (1.1)$$

The capacitive switching power together with the short-circuit power is called *dynamic power*, and it is due to charging and discharging in CMOS gate. Dynamic power is considered to be a significant part in the total power consumption and is given by the following equation

$$P_{dynamic} = \frac{1}{2} \alpha C_L V_{dd}^2 f_{clock}, \quad (1.2)$$

where C_L is the load capacitance at the gate output, f_{clock} is the circuit clock frequency, V_{dd} is the supply voltage, and α is the average number of transitions per clock cycle at the gate output, referred to as the *switching activity*.

Power/energy reduction in embedded system can be achieved by carefully designing each of its constituent components targeting low power/energy design. In dedicated hardware units, power/energy reduction is usually achieved through optimizing dynamic power consumption in the main high-level synthesis tasks, namely scheduling, allocation, and binding. In processor cores, dynamic power-management techniques, compilation-based techniques such as code transformations, adoption of domain-specific instructions and specialized addressing modes, and dynamic voltage scaling are used extensively to achieve power/energy reduction within

performance constraint. In memory system, the central idea of temporal locality is exploited to obtain power/energy reduction by reducing memory requirements and data-transfer traffic to and from memory [BM00].

1.3 High-Level Synthesis

High-level synthesis (HLS) is the process of mapping the behavioral specification of a system into register-transfer description. The outcome of the high-level synthesis is a structural view of the data path and a logical view of the control unit. A behavioral specification is represented at the algorithmic level using one of the programming languages such as C or Pascal, or a hardware description language (HDL) such as VHDL or Verilog. A behavioral specification is compiled into an internal representation (a graph representation in most cases) that is suitable for HLS optimization algorithms and captures the behavior, structure, and timing constraints in the input design specification. The internal representation (usually in the form of a graph) along with the input design constraints are fed to the HLS algorithms to obtain a structural implementation (RTL) that satisfies the input specification and optimizes certain design objective such as performance, size and power consumption. During the compilation process, code transformations that support the design goal can be applied as shown in [PS98, LP93, WP95, PR94, Cho93, CR95].

High-level synthesis involves three main tasks: scheduling, allocation (also called module selection), and binding. Scheduling is the process of determining at which control step(s) each operation in the data-flow graph (DFG) executes. Allocation determines the number and the types of the hardware resources (functional units, registers, and interconnections) from an input library. Binding is the process of assigning specific instances of the allocated modules to each computational element and to each storage value. The three tasks are inter-related, which significantly increases the difficulty of a complete exploration of the design space for even small instances of the synthesis problem.

Performance and/or size optimization considered as the traditional design objectives and they have been tackled extensively by a large body of research effort in the past two decades. Scheduling is the most important task in HLS that affects the design of the digital circuits. Scheduling algorithms for size and/or performance optimization can be mainly classified as resources-constrained scheduling heuristics [SD02, SL02], time-constrained scheduling

heuristics [PK89, CT90, NK93, VA95, PS02], time- and resources-constrained scheduling heuristics [MC99], or mathematical formulations using *integer linear programming* (ILP) [HP83, GE93, CHd95, CB97]. Allocation and scheduling are interleaved tasks in which the decision taken during one of them affects the decision by the other. Allocation for size optimization has been addressed in [Jai90, CJ91, KN92, GM92, TH93, AD95, CW96, Bly97].

Dynamic power (capacitive switching power and short circuit power) is a significant source of power consumption in CMOS circuits. Because of the quadratic relationship of the supply voltage to dynamic power consumption, lowering the supply voltage is considered to be the most effective approach for minimizing power/energy consumption in circuits. However, lowering the supply voltage increases the circuit delay. Parallelization and pipelining can compensate for the increased delay but at the cost of increased area overhead. Using multiple supply voltages in the data-flow graph (DFG) scheduling solves the problem and minimizes the power/energy consumption under latency constraint without an increase in area. The idea is to assign the highest voltage level to the operations on the critical path to meet the time constraint (or to achieve minimum latency in resource-constrained scheduling) and to use a lower voltage level for operations not on the critical path to achieve power/energy minimization.

1.4 Motivating Example

Peak power minimization plays as substantial a role as or even more than average power minimization in low power design. It has a direct effect on battery lifetime as well as on the reliability of integrated circuits. Consider the DFG in Figure 1.2. Assume that the delays of the DFG operation are 2 and 1 time steps if it is scheduled with a high and low voltage-level, respectively. Let each operation consume 20 μ watt and 8 μ watt average power if it is scheduled with a high and low voltage-level, respectively. There are two schedules for the DFG in Figure 1.2-(a) with the same average power consumption but with different values of peak power consumptions as shown in Figures 1.2-(b) and 1.2-(c). The schedule in Figure 1.2-(b) has 18 μ watt average power and 28 μ watt peak power consumptions; the schedule in Figure 1.2-(c) is an optimal schedule with 18 μ watt average power and 20 μ watt peak power consumption. This example shows that considering peak power minimization in the scheduling process can result in a power efficient schedule.

1.5 Memory System

Memory is an integral part of embedded system and it is the most dominant component in its size. Thus minimizing memory requirement helps in reducing the size and so the cost of an embedded system [BC95]. In addition, minimizing the amount of memory has an impact on reducing the amount of data traffic between the SOC and the off-chip storage that reflects in power minimization. Memory optimization is a prevalent goal of the most compilation techniques in addition to performance optimization especially for the special-purpose processors used in embedded systems. In the scientific computations field, a fragment of the application program is repeatedly executed accessing large data arrays. Most of the compilation techniques use loop transformations for both performance optimization and memory minimization. Fusion is a loop restructuring technique for loop optimization in which the producer loop nest of an array is merged with its consumer loop nest inside a common loop. By fusing a common loop between producer loop and consumer loop nest, the required storage of the intermediate array is reduced by the range of the fused loop.

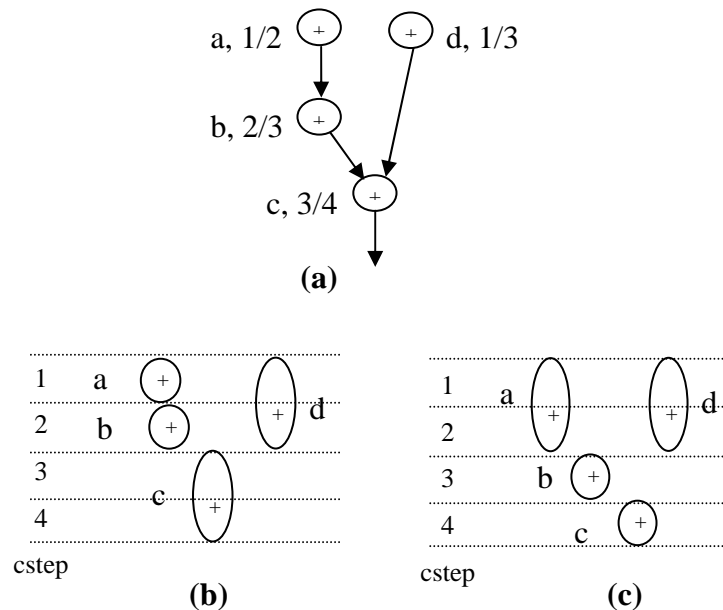


Figure 1.2: Effect of peak power consideration: (a) the DFG annotated with ASAP/ALAP values for $\lambda = 4$, (b) average power minimization alone, and (c) simultaneous peak and average power minimization.

For data-dependent program regions, there are many different orders of evaluation that vary widely in the maximum memory usage required when dynamic memory allocation is used. Note

that with dynamic memory allocation, a data object is allocated memory when it is needed (i.e., memory is allocated at the start of the computation that creates the data values and is used until all its parents are evaluated) and then it is de-allocated after its last use. The method that finds the order of evaluation for the least memory requirement is called the “memory evaluation order” technique. This technique can also be applied for loop optimization on the fused loops with careful consideration of the allocation and de-allocation of arrays inside some fused loop structure.

The class of computations considered in this work arises in the field of correlated electronic structure methods such as coupled cluster and configuration interaction methods in quantum chemistry [Aul96, HL86, RG95]. In this class of computations, loop computations are specified as multi-dimensional integrals of products of many input arrays. These computations can be expressed numerically as multi-dimensional sums of products of input arrays. There are many different ways to get the same final results; the different ways require differing number of arithmetic operations due to operator properties such as commutativity, associativity and distributivity. Lam et al. [LS97] have devised an optimization procedure to do loop computations using the minimum number of floating point operations through determining an equivalent sequence of multiplication and summation formulas; the resulting optimal sequence of formulas is called an *operation-count-optimal* formula sequence. The intermediate result from each formula is stored in an intermediate array that can be used many times without the need for re-computing these results.

Resulting formulas can be implemented as separate sets of perfectly nested loops, one set for each formula. In this way, intermediate arrays have to be stored in full; in most cases the intermediate arrays are huge with sizes often exceeding the available memory on most machines. Loop fusion [GO92, KM93, LS97, Lam99, MA95, SM96] is a candidate solution for reducing memory usage. As stated earlier, fusing loops between the producer loop nest and consumer loop nest reduces the required storage of intermediate arrays. Because resultant loops have to be legal after fusion, fusing some loops precludes fusing others. In this work, we consider the problem of deciding which loops are to be fused to achieve minimal memory usage (called the *optimal memory usage* problem). We consider memory usage optimization in a restricted class of computations that arises in scientific computing field such as electronic structure calculations. In

addition, we address memory minimization problem in a multi-processing elements with shared memory model.

1.6 Related Work

1.6.1 High-Level Synthesis

Since dynamic power (capacitive switching power and short circuit power) is considered to be a significant source of power consumption in CMOS circuits, most of the research work has focused on minimizing dynamic power consumption in HLS [KK95, LM96, SG97, MP98]. Some of the research has tackled the dynamic power minimization problem by reducing the switching activity of allocated resources [RJ95a, RJ95b, CP95, MC95, LM01, HC02, HR03, LK03]. Others have developed techniques for dynamic power management at system level using clock gating [FS95, MD96, LR99]. The idea is to turn sections of the clock tree on and off during a module's active and idle modes, respectively, in order to save power.

In recent years, a lot of research work has been done to solve the multiple supply voltages scheduling (MVS) problem. Some of these research works addressed the MVS problem using heuristics [CP97, Lin97, SC00a, RV03, WJ04], while others addressed it using *integer linear programming* (ILP) [Lin97, JR97, SC00b, MC02, CC03a]. Lin et al. [Lin97] proposed an ILP formulation and a heuristic for solving the scheduling with power-minimization problem using variable supply voltages technique. They formulate the problem as minimization of power under timing, resource, and combined timing and resources constraints. Their heuristic is an iterative list-based scheduler with $O(n^3 \log n)$ time complexity. It uses the *delaying gain* of an operation as the priority function; they define the delaying gain of an operation as the weighted sum of the power gain, the mobility, and the computation density of an operation. In their ILP formulation, they modeled the cost function to minimize power as well as other factors such as time and/or number of resources. The problem with their ILP formulation is that power modeling in the cost function is not accurate. Johnson and Roy [JR97] have introduced a scheduling technique called MOVER (Multiple Operating Voltage Energy Reduction) to obtain a data path energy-minimized schedule using multiple supply voltages. The core of the MOVER technique is an ILP model. The algorithm assumes a given number of voltage levels (not the actual values of these voltage levels) and it searches a continuous range of voltages when seeking a minimum voltage level to achieve energy minimization. On the other hand, Chang and Pedram [CP97] have

presented a dynamic programming technique to solve the problem of multiple supply voltage scheduling. Their technique assigns a voltage level (selected from a given fixed number of voltage levels) to each operation in the DFG to minimize the energy consumption under time constraint. The technique takes into consideration the switching activity as well as the delay and area overhead of the level converters. The algorithm is pseudo-polynomial and gives optimal results for trees, but is suboptimal for a general directed acyclic graph. Recently, Manzak and Chakrabarti [MC02] introduced a list-based scheduling algorithm that minimizes power/energy consumption under time and resource constraints when the resources operate at multiple supply voltages. The proposed algorithm operates in two passes. Resource-constrained scheduling to obtain a minimum time is performed in the first pass. In the second pass, the slack time (the time difference between the given latency and the time obtained from the first pass) is distributed among the nodes of the DFG using the energy delay ratio of the nodes (using the Lagrange multiplier method) to minimize the power/energy consumption. Gupta and Katkooi [GK02] presented a latency-constrained scheduling algorithm for low-power design based on the force-directed scheduling approach. Data profiling is used to collect information about the switching activities, the switching capacitances inside modules are modeled, and the concept of force is used to make a power-optimal scheduling decision. Mohanty and Raganathan [MR03] introduced an ILP based optimization technique for simultaneous minimization of peak and average power using a multiple supply voltages scheme. They introduced two datapath scheduling schemes, one using multiple supply voltages and dynamic clocking and the other using multiple supply voltages and multicycling. Shiue [Shi00] has presented an ILP model and a modified force-directed scheduling (MFDS) heuristic that minimizes peak power under latency constraint using a single supply voltage. He considered multicycling and pipelining but he did not consider multiple supply voltages. Our presented MILP formulation differs from the one presented in [Shi00] that deals with a single voltage level: our formulation considers multiple voltages. In addition, the variables used in the MILP formulation by [MR03] are 4-dimensional variables, while ours are 3-dimensional, which has a big impact on decreasing the solution run-time.

Module selection is interdependent on the scheduling task in HLS. Thus, some of the research works consider them simultaneously, while others considered module selection and scheduling in an iterative fashion to cope with the high complexity if they are addressed

simultaneously. Timmer et al. [TH93] have introduced a two-step algorithm for module selection and scheduling targeting area minimization. Module selection is done first using an MILP formulation, followed by a resource-constrained list scheduling heuristic to check for time violation. If there is available time, the algorithm reconsiders the module selection phase again and so on until the time constraint is met. Jain [Jai90] has presented an ILP formulation for module selection in pipelined designs. The formulation does not consider scheduling and only considers the more restricted form of the module selection problem where all the instances of the same operation type are implemented using the same module type. These two algorithms did not consider power consumption in the module selection process. Power consumption is considered in the work presented in [CP96, Shn97, GS97, CS00]. Shen and Jong [Shn97] presented a heuristic for module selection design space exploration targeting power consumption. Module selection is formulated as a multi-objective optimization problem in which a cost function is iteratively improved each time a better configuration is found; they used a branch and bound technique to prune off inferior designs. In their algorithm, module selection is separated from the scheduling phase and they do not consider resource constraints. Chantrapornchai et al. [CS00] proposed an approach based on fuzzy logic for solving the combined scheduling, binding, and module selection under latency and power constraint. Their approach is a two-phase heuristic, the configuration estimation phase and the module selection phase for the derived resource configuration. The design objective is achieved using what they call the acceptability function, which evaluates the usefulness of a module based on its utility. Our MILP formulation addresses peak power consumption, which is not considered in the previous work (to the best of our knowledge). In addition, our work considers scheduling and module selection simultaneously for power minimization. Also, our formulation adopts an unrestricted library in which a computational element can be mapped to many module types and some module types can be used to realize several operation types.

Retiming was introduced by Leiserson and Saxe [LS86] as an optimization technique for synchronous digital circuits in which the registers in the circuit are redistributed in such a way as to achieve a certain objective while preserving the circuit original functionality [Cho93, CM98, NT99, NS01]. Since retiming can be used to shorten the critical path delay and to increase parallelism among the computation nodes, it can be used to increase the number of computational elements that are candidates for voltage scaling. Devadas [MD93] presented a

heuristic for power minimization in synchronous sequential circuits, in which the flip-flops are positioned at the output of computational nodes rather than at the output of registers in order to minimize the switching activity. Chabini et al. [CC03b] have addressed the problem of minimizing the dynamic power (only the average power) consumption in synchronous sequential circuits using a unification of basic retiming and supply voltage scaling. They formulated the problem as an MILP problem for a given clock period. Also, Chabini et al. [CW04] have devised a heuristic for minimum dynamic power consumption. They first presented a retiming procedure using an LP formulation to maximize the total number of non-zero-delay (NZD) edges in order to maximize the parallelization among the graph nodes. Then, it is followed by an MILP formulation for the resultant circuit with the objective to minimize the average dynamic power consumption. We present a mathematical MILP formulation for the combined retiming and multiple voltages scheduling problem for power (peak and/ or average) optimization. In addition, we propose an efficient retiming procedure called *power-oriented retiming*, which exploits the circuit structure to impose control over edge selection in the retiming process in order to achieve power saving.

1.6.2 Memory System

Reduction of arithmetic operations has been traditionally done by compilers using the technique of common sub-expression elimination [FL91]. Chatterjee et al. [CG93, CG95] consider the optimal alignment of arrays in evaluating array expressions on massively parallel machines, but they do not consider distribution and replication of arrays. Much work has been done on improving locality and parallelism by loop fusion [KM93, MA95, SM96]. However, this work considers a different use of loop fusion, which is to reduce array sizes and memory usage of automatically synthesized code containing nested loop structures. Traditional compiler research does not address this use of loop fusion because this problem does not arise with manually-produced programs. The contraction of arrays into scalars through loop fusion is studied in [GO92] but is motivated by data locality enhancement and not memory reduction. Guibas and Wyatt [GW78] discussed loop fusion in the context of delayed evaluation of array expressions in APL programs but their work is also not aimed at minimizing array sizes. We are unaware of any work on fusion of multi-dimensional loop nests into imperfectly-nested loops as a means to reduce memory usage other than that of Lam et al. [Lam99, LS99b, LS99a], which

uses a search procedure. In addition, our presented mixed integer linear programming (MILP) formulation for the memory usage optimization problem is not presented elsewhere.

1.7 Dissertation Outline

This thesis addresses two important factors in embedded system design, power consumptions minimization and memory optimization.

The first four chapters (Chapters 2, 3, 4 and 5) consider the problem of power consumption (peak power as well as average power) minimization in high-level synthesis. The developed techniques target the applications that do not include control-flow structures such as those in signal and image processing applications.

Chapter 2 presents an MILP formulation for the scheduling problem using multiple supply-voltages in order to optimize peak power as well as average power and energy consumption under two sets of constraints, time-constraint alone (TCS), and time and resource constraints (TRCS). Then, a two-phase iterative LP and power-resource saving heuristic is devised to solve the MVS for large design problems. Chapter 3 tackles the same problem, MVS for peak and average power optimization, using a modified force-directed scheduling technique. Chapter 4 considers the problem of module selection simultaneously with the scheduling task targeting power consumption (peak and average) optimization. Chapter 5 addresses the problem of reducing power consumption (peak and average) in synchronous sequential circuit designs (where the data-flow graphs representation contains cycles across iterations). We present a combination of basic retiming and multiple voltage scheduling (MVS) techniques in a unified fashion. Then a two-stage algorithm (power-oriented retiming followed by a MVS technique similar to the one in chapter 2) for peak and/or average power optimization is presented.

The second part of the thesis (Chapters 6 and 7) deals with memory usage optimization targeting (mainly) a restricted class of computations that arise in scientific computing field such as electronic structure calculations. Chapter 6 considers the DFG evaluation order for dynamic memory usage optimization, while Chapter 7 addresses the static memory optimization through loop fusion technique.

Chapter 8 concludes with a discussion of the contribution of this work and points to possible future work.

CHAPTER 2

DSE FOR PEAK AND AVERAGE POWER OPTIMIZATION USING MULTIPLE SUPPLY-VOLTAGES

High-level synthesis (HLS) is the process of mapping the behavioral specification of the system into register transfer description. The outcome of the high-level synthesis is a structural view of the data path and a logical view of the control unit. High-level synthesis involves three main tasks: scheduling, allocation, and binding. The central task is scheduling, which is the process of determining at which control step(s) each operation in the DFG executes. We define *Scheduling for Low Power and Energy (SLoPE)* in high-level synthesis as the process of determining at which control step(s), and at what voltage level each operation in the DFG executes with the goal of minimizing power and energy.

The supply voltage has a quadratic relationship to the dynamic power consumption. Thus, the most effective approach for minimizing power/energy consumption in circuits is to lower the supply voltage. However, lowering the supply voltage increases the circuit delay. Parallelization and pipelining can compensate for the increased delay but at the cost of increased area overhead. Using multiple supply voltages in the DFG solves the problem. It minimizes the power/energy consumption under latency constraint without resorting to area penalty. The idea is to assign the highest voltage level to the operations on the critical path in order to meet the time constraint and to use a lower voltage level for the operations not on the critical paths to achieve the power/energy minimization.

2.1 Problem Definition

The input to the problem include a DFG representation of the design problem, $G(V, E)$ in which each vertex $v \in V$ represents a computational operation and each edge (u, v) means that operation u has to finish its execution before operation v starts, a set of voltage levels for the operating resources, and a power/delay table that contains the average power consumption and

the delay time needed for each resource operating on each voltage level and the time constraint, λ . The task at hand is to get a schedule (in which each operation is stamped to a control step, $cstep \in (1, 2, \dots, \lambda)$ and a voltage level from the set of input voltage levels) that minimizes the peak power consumption as well as the average power and energy consumption according to one of the set of constraints such as time constrained scheduling (TCS), and time and resource constrained scheduling (TRCS).

We propose two solutions for the multiple supply-voltages scheduling (MVS) problem targeting peak power consumption as well as other design factors such as average power and energy consumption, and area as shown in Figure 2.1. (1) is an exact solution based on a mixed integer linear programming (MILP) formulation. (2) is a two-phase heuristic that first obtains a guided iterative relaxed LP solution of the MILP formulation followed by a *power-resources saving* procedure, which is a revisit of the output schedule from the first phase in which it tries to minimize the power and/or the operating resources more through scheduling operations in a lower voltage level if possible and/or through moving the operations within their new time-frames if possible without violating the peak power obtained from the first phase.

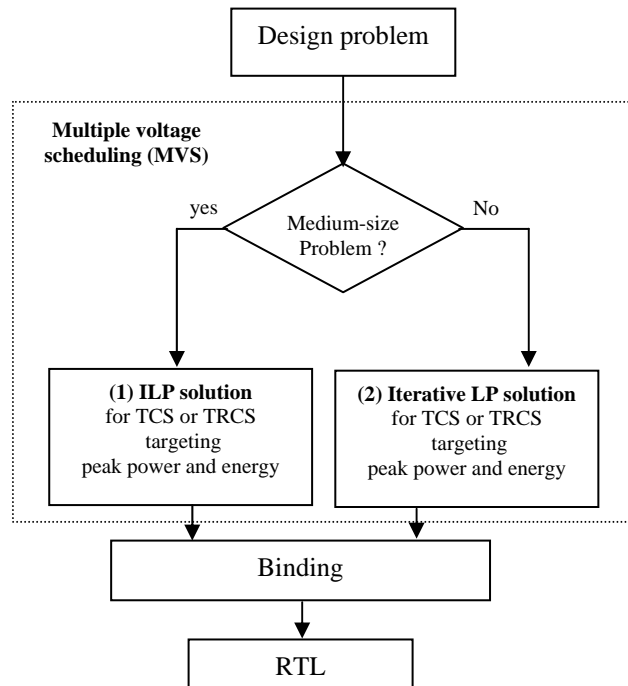


Figure 2.1: Flow-chart for the DSE using multiple voltage scheduling.

Our proposed iterative LP solution has not been presented elsewhere to the best of our knowledge. In addition, our presented MILP formulation differs from the one presented in

[Shi00] that deals with a single voltage level, while ours considers multiple voltages. In addition, the variables used in the MILP formulation by [MR03] are 4-dimensional variables, while ours are 3-dimensional, which has a big impact on decreasing the solution run-time.

2.2 Exact Solution

Our proposed optimal algorithm for the multiple supply voltages scheduling (MVS) problem is as shown in Figure 2.2. It assumes that the clock-selection phase is already done and so the input delays for DFG nodes are expressed in number of csteps. First, the as-soon-as-possible (ASAP) and as-late-as-possible (ALAP) schedule (computed using the highest voltage-level) is calculated as a preprocessing step to tighten the time-frames for graph vertices and so the number of variables in the MILP formulation. Then, the MILP formulation is developed to solve one of the two problems, the *TCS problem* using the objective function (2.1) and the set of inequalities (2.2)-(2.5), or the *TRCS problem* using the objective function (2.1) and the set of inequalities (2.2)-(2.6).

- Calculate ASAP and ALAP using the highest voltage-level.
- Construct the MILP formulation as in Equations (2.1)-(2.6).
- Use an ILP solver to solve the model.
- Construct the optimal schedule.

Figure 2.2: Procedure of the multiple supply-voltages scheduling.

2.2.1 TCS Problem

Given the time constraint λ and a set of voltage levels for the operating resources, and a power/delay table that contains the average power consumption and the delay time needed for each resource operating on each voltage level, find a schedule that minimizes peak power and/or energy consumptions.

2.2.2 TRCS Problem

Given the time constraint λ , the number of resources of each type of computational element, a set of voltage levels for the operating resources, and a power/delay table that contains the average power consumption and the delay time needed for each resource operating on each voltage level, find a schedule that minimizes peak power and/or energy consumption.

Define x_{ijv} to be a 0-1 unknown variable that takes value 1 if node i starts execution at cstep j with voltage level v and 0 otherwise. Then, the MILP formulation is as follows.

$$\text{Minimize :} \quad \alpha P_{peak} + \beta (\frac{1}{\lambda}) \sum_i \sum_j \sum_v x_{ijv} d(i,v) \cdot p(i,v) \quad (2.1)$$

$$\sum_v \sum_j x_{ijv} = 1 \quad \forall i \in V \quad (2.2)$$

$$\sum_v \sum_j (j + d(i,v) - 1) x_{ijv} + \sum_v \sum_j j x_{jiv} \leq -1 \quad \forall (i,l) \in E \quad (2.3)$$

$$\sum_i \sum_v \sum_{j_1=j-d(i,v)+1}^j x_{ijv} p(i,v) \leq P_{peak} \quad \forall j \in [1, \lambda] \quad (2.4)$$

$$\sum_v \sum_j (j + d(i,v) + 1) x_{ijv} \leq \lambda \quad \forall \text{node } i \text{ without successors} \quad (2.5)$$

$$\sum_{i \in FU_k} \sum_v \sum_{j_1=j-d(i,v)+1}^j x_{ijv} \leq M_k \quad \forall j \in [1, \lambda], FU_k \quad (2.6)$$

$$x_{ijv} \in \{0,1\} \quad (2.7)$$

Equation (2.1) is a flexible weighted objective function, where the weight factors can be set according to the design requirements. Equation (2.2) forces each node to start at only one cstep, and be scheduled using one and only one voltage level. The precedence relations are satisfied by Equation (2.3). Peak power can be set as a constraint or can be used as a variable to be minimized in the objective function as shown in Equation (2.4). To meet the latency requirement, each node without successors is forced to finish execution by λ through Equation (2.5). Finally, Equation (2.6) is presented to constrain the number of resources used from each type. Our MILP formulation is flexible since the peak power can be treated as an objective to be optimized or can work as a constraint for those applications that have hard limits on peak power consumption. In addition, for design space exploration, power consumption (average and/or peak) can be set as a constraint for area or time minimization.

As a detailed example of our MILP formulation for the TCS problem, consider the DFG in Figure 1.2, which is redrawn in Figure 2.3 for convenience, with the same assumptions of the power consumptions and execution delays of the DFG nodes as in Section 1.4. The complete listing of constraints and the objective function (using $\alpha = 1$ and $\beta = 1$) is shown in Figure 2.4 using the 0-1 variables associated with each node as shown in Table 2.1. The set of constraints c1:c4 are the uniqueness constraints in Equation (2.2), the set of constraints c5:c7 are the precedence constraints as a result of applying inequality (2.3), and constraints c8:c11 are the peak power constraints, where the peak power is treated as a variable P_{peak} that is included in the objective function to be minimized. The only node without successors is node c that is forced

to finish its execution by four csteps through constraint c12. The output from the MILP solver CPLEX [CPX02] is as shown in Figure 2.3-(b), which is an optimal schedule for peak power as well as average power.

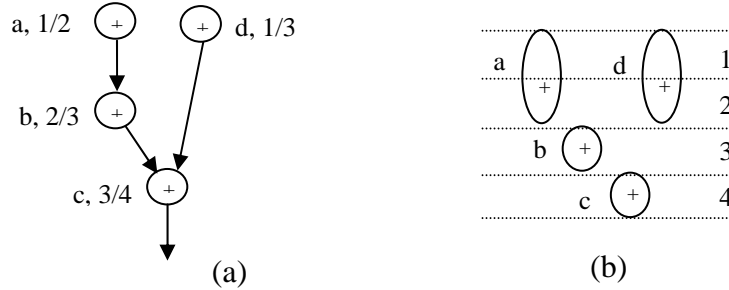


Figure 2.3: Example of the MILP formulation: (a) the DFG annotated with ASAP/ALAP values for lambda = 4, (b) simultaneous peak and average power minimization.

Table 2.1: DFG nodes and their associated 0-1 variables for Figure 2.3-(a)

node	indexed variables	ILP solver variables
a	x_{a11} x_{a21} x_{a12}	$x1$ $x2$ $x3$
b	x_{b21} x_{b31} x_{b22}	$x4$ $x5$ $x6$
c	x_{c31} x_{c41} x_{c32}	$x7$ $x8$ $x9$
d	x_{d11} x_{d21} x_{d31} x_{d12} x_{d22}	$x10$ $x11$ $x12$ $x13$ $x14$

Minimize

$$\text{obj: } 5 x1 + 5 x2 + 4 x3 + 5 x4 + 5 x5 + 4 x6 + 5 x7 + 5 x8 + 4 x9 + 5 x10 + 5 x11 + 5 x12 + 4 x13 + 4 x14 + P_{\text{peak}}$$

Subject To

$$c1: x1 + x2 + x3 = 1$$

$$c2: x4 + x5 + x6 = 1$$

$$c3: x7 + x8 + x9 = 1$$

$$c4: x10 + x11 + x12 + x13 + x14 = 1$$

$$c5: x1 + 2 x2 + 2 x3 - 2 x4 - 3 x5 - 2 x6 \leq -1$$

$$c6: 2 x4 + 3 x5 + 3 x6 - 3 x7 - 4 x8 - 3 x9 \leq -1$$

$$c7: -3 x7 - 4 x8 - 3 x9 + x10 + 2 x11 + 3 x12 + 2 x13 + 3 x14 \leq -1$$

$$c8: 20 x1 + 8 x3 + 20 x10 + 8 x13 - P_{\text{peak}} \leq 0$$

$$c9: 20 x2 + 8 x3 + 20 x4 + 8 x6 + 20 x11 + 8 x13 + 8 x14 - P_{\text{peak}} \leq 0$$

$$c10: 20 x5 + 8 x6 + 20 x7 + 8 x9 + 20 x12 + 8 x14 - P_{\text{peak}} \leq 0$$

$$c11: 20 x8 + 8 x9 - P_{\text{peak}} \leq 0$$

$$c12: 3 x7 + 4 x8 + 4 x9 \leq 4$$

Figure 2.4: Complete listing of constraints and the objective for the DFG in Figure 2.3-(a).

2.3 Iterative LP Relaxation

When the design problem is large, the solution run-time for the MILP becomes a problem because of the exponential nature of the ILP solution algorithms. LP relaxation might be a solution of this problem in some cases, but the quality of the final solution depends on the method of relaxation. In addition, because the LP relaxation of the IP problem in general is not integral, there should be some way to guide the relaxation to get a solution quality close to the optimal solution. The integrality constraint in Equation (2.7) can be relaxed by Equation (2.8). The fractional values that result from running the LP solution once by itself do not reflect meaningful information. Just rounding off the fractional value associated with the 0-1 variable in the final solution is not a good idea because first, the correctness is not guaranteed (for example, precedence relations among the nodes might be violated), second, even if the precedence relations are met, the final solution is far from the optimal one.

$$0 \leq x_{ijv} \leq 1 \quad (2.8)$$

We devise a guided way of relaxation called “*iterative LP relaxation*” as shown in Figure 2.5. The idea is to develop the LP solution iteratively in several stages. In each stage, the variables (especially the 0-1 variables) that are relatively “large” (variables with largest fractional values, which if they are set to 1’s, they lead to optimal or near-optimal solution) to contribute to the optimal solution are selected. If their resultant values are integral, they are set to these values and fixed during the successive iterations. If their resultant values are fractional, they are tested against a threshold value, and any 0-1 variable passing the test, it is set to one and fixed during the successive iterations. At the same time the rest of the 0-1 variables associated with the same node are set to zeros. The solution iterates until all the 0-1 variables pass the threshold test. The threshold value is set dynamically as the maximum value of the 0-1 variables from the resultant solution of the current LP iteration. This is to elect the most mature 0-1 variables to contribute to the final solution.

In the case of time and resource constraint scheduling (TRCS), the LP output solutions may become infeasible during an iteration because the accumulated candidate 0-1 variables that have been set to one in this iteration might violate the resource constraints. Thus, at any iteration during the iterative relaxation, when the resultant LP solution is infeasible, the number of resources is increased by one and the model is resolved again. That increase in the number of resources will be restored again using the power- resources -saving phase of the heuristic.

1. Calculate ASAP and ALAP times using the highest voltage-level.
2. Construct the MILP formulation as in Equations (2.1)-(2.6).
3. Relax the integrality of the 0-1 variables by substituting (2.7) by (2.8).
4. Iterative procedures:
 - 4.1 Use an LP solver to solve the model.
 - 4.2 In case of resource constraint, if the solution is infeasible, increase the number of resources by one and solve again.
 - 4.3 Set the threshold value to be the maximum of the 0-1 variables in the LP solution from step 4.1.
 - 4.4 If any 0-1 variable passes the threshold do:
 - 4.4.1 set its value to 1 and fix it during the successive iterations.
 - 4.4.2 Set all the 0-1 variables associated with the same node to 0.
 - 4.5 Update ASAP and ALAP values.
 - 4.6 If NOT all 0-1 variables are set (to either 0 or 1) GoTo 4.1.
5. Construct the schedule.

Figure 2.5: Iterative LP procedures.

Consider again the DFG example in Figure 2.3 with the associated 0-1 variables for each DFG node for time constraint 4 in Table 2.1. The iterative LP procedure takes 3 iterations to complete the solution. The outcome of each iteration is as shown in Table 2.2. After each iteration, the threshold value is set to the maximum value among the 0-1 variables written in bold face in Table 2.2. After the first iteration, x8 variable is set to 1 and the rest of associated variables of node c, x7 and x9, are set to zeros forcing node c to be scheduled at cstep 3 with the higher voltage level. Node d is scheduled at cstep 1 with the lower voltage level after its associated variable x13 passes the threshold value in the second iteration and is set to 1, while nodes a and b are scheduled in the third iteration. The final LP solution results in the same schedule as that obtained from the MILP solution shown in Figure 2.3-(b).

Although, the worst-case number of iterations is the number of nodes in the DFG, the actual number of iterations is very small as will be clear in the experimental results for most benchmarks. This is because at each iteration many variables pass the threshold and the time-frames become tighter forcing many variables to settle.

2.4 Power- Resources Saving

The goal of the second phase of the algorithm, power-resources saving procedure, is to gain additional power and/or resources saving through exploiting any available flexibility for an operation. It tries to schedule the DFG operation with a lower voltage level (more power saving)

and/or to move it up and down within the available room without violating the peak power obtained from the first phase, to get more peak power saving and/or resources saving. The algorithm for power-resources saving is shown in Figure 2.6. The inputs to the algorithm are the resultant schedule attributes from the first phase (the iterative LP) where *scheduleStep* and *vLevel* are the cstep and the voltage-level stamped to each operation, respectively, and *peak_power* is the resultant peak power consumption from the first phase.

Table 2.2: Gradually iterative LP solution for the DFG in Figure 2.3-(a)

node	var	itr#1	itr#2	itr#3	final sol
a	x1	0.45339	0.59979	0.47056	0
	x2	0	0	0	0
	x3	0.54661	0.40021	0.52944	1
b	x4	0.32353	0	0	0
	x5	0.54661	0.40021	0.52944	1
	x6	0.12986	0.59979	0.47056	0
c	x7	0	0	0	0
	x8	0.67647	1	1	1
	x9	0.32353	0	0	0
d	x10	0	0	0	0
	x11	0	0	0	0
	x12	0	0	0	0
	x13	0.63207	0.67122	1	1
	x14	0.36793	0.32878	0	0
peak power	Ppeak	17.5	23	23	20

2.4.1 Time Complexity of Power-Resources Saving

The power-resources saving algorithm has a worst-case time complexity in the order of $O(n^2)$, where n is the number of operations in the DFG. Following is a derivation for this time complexity.

1. The core of the algorithm inside the first “for-loop” is executed exactly n times (once for each operation) since only unmarked operations are subject to the computations inside that “for-loop”. These n iterations are the summation of the iterations through both “while-loop” and “for-loop”.
2. At each iteration, ASAP and ALAP routines are executed to update the time-frames of the DFG operations. This is done with $O(n)$ time complexity.

3. The potential reduction in peak power and /or in resources is examined for each potential voltage-level and for each cstep within the time-frame of unmarked operation at hand. This requires $O(\lambda V)$ calculations where λ is the total time constraint and V is the number of input voltage-levels. This is because the time-frame for an operation is at most λ and the number of potential voltage-levels is at most V (a voltage-level is excluded for an operation when there is no room for that operation to be scheduled using that voltage-level).
4. In each iteration, the complexity is determined by the maximum of $O(n)$ and $O(\lambda V)$. In practice, the usual number of operating voltage-levels in a circuit is two or three and the time-fame of an operation is much smaller than λ , therefore, the complexity of steps 2 and 3 is $O(n)$.

Power-resources saving(*scheduleStep*, *vLevel*, *peak_power*)
marked = 0 for all operations.
count = 0
while(*count* < *numOperations*) do
 for(*op* = 1: *numOperations*)
 1. if (*marked*(*op*)=1 or *indegree*(*op*) > 0)
 skip the rest of loop body.
 2. update the time frames
 3. compute the room of *op* using *scheduleStep*
 of its predecessors and its successors.
 4. for(*v* = *numVlevels*:1 step -1)
 4.1 if (there is a room to schedule *op* with *v* without
 violating the peak power and the input resource
 constraints if any)
 4.1.1 schedule *op* at *cstep* within its time frame to get
 smaller power and/or resources and set:
 4.1.2 *marked*(*op*) = 1.
 4.1.3 *vLevel*(*op*) = *v*.
 4.1.4 *scheduleStep*(*op*) = *cstep*.
 4.1.5 *count* = *count* + 1.

Figure 2.6: Power-resources saving algorithm.

Figure 2.7 is an example showing the effect of the power-resources saving procedure to minimize peak power consumption when there are some flexibility. Figure 2.7-(a) is the DFG of the HAL benchmark where operations 1, 2, 3, 6, 7 and 8 are multiplication operations and

operations 4, 5, 9, 10 and 11 are addition operations. The input module library is shown in Table 2.3. The resultant schedule from the iterative LP approach is shown in Figure 2.7-(b); while the modified schedule after the power-resources saving procedure is shown in Figure 2.7-(c). As it is clear from the two schedules, operation 8 is pushed from the high peak power csteps 1 and 2, to a lower peak power csteps 3 through 6. This results in a reduction of peak power consumption from 123 μ watt to 110 μ watt, which happens to be optimal in this case.

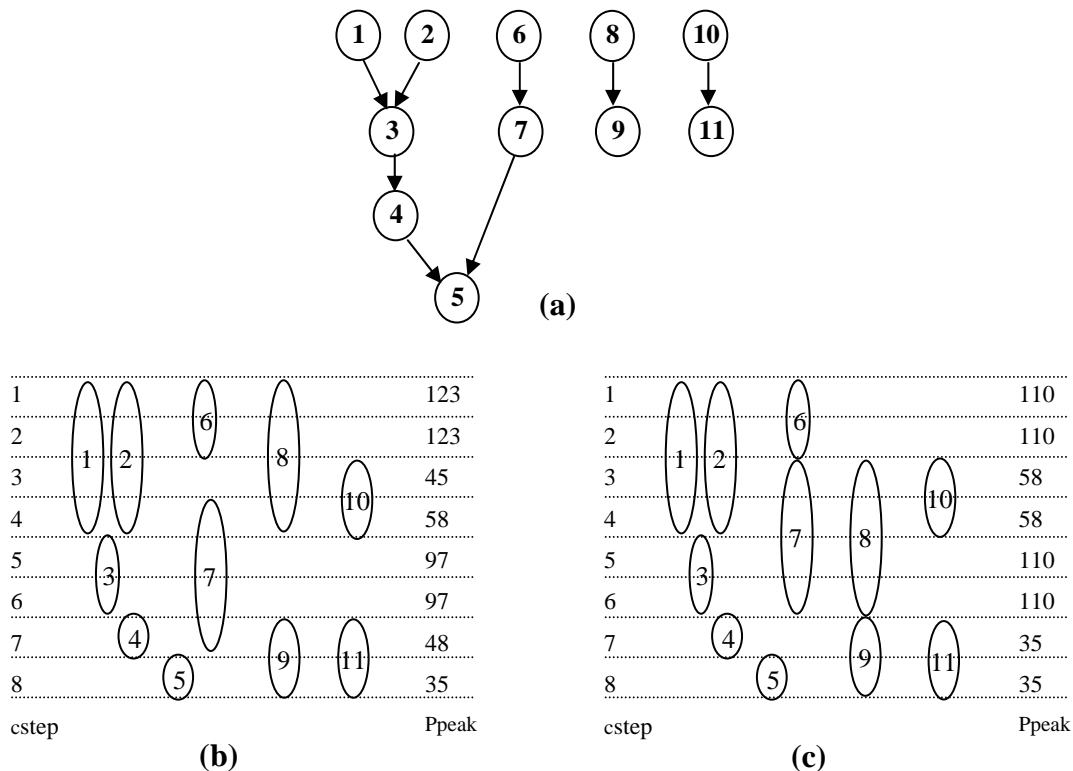


Figure 2.7: An example demonstrating the effect of power-resources saving procedure: (a) the DFG, (b) resultant iterative LP schedule, (c) schedule after power- resources saving.

2.5 Experimental Results

The two proposed solutions, the exact and the two-phase heuristic, are tested on standard benchmarks like HAL, ARF, and EWF using the module library shown in Table 2.3. In the tabulated results, “avg” means average power, “peak” means peak power, “[*, +]” means the number of [multipliers and adders] resources used, and “#itr” is the number of LP iterations until the iterative LP solution is finalized. Experiments are run for each benchmark with time constraint varying from the critical path length to twice the critical path length. The exact ILP solution and the results of the two-phase heuristic after each phase of the algorithm are tabulated

in the set of columns under the heading “ILP Sol”, “Iterative LP Sol”, and “After Power Saving”, respectively. Results of the tested benchmarks under different time constraints for the TCS problem are shown in Tables 2.4 through 2.6; while the results for the TRCS problem are shown in Tables 2.7 through 2.12 under different sets of resource constraints. Results of the iterative LP approach followed by the power-resource saving procedure are compared to the optimal solution (ILP solution) as tabulated. These results show that, in most cases the results of our heuristic well match those obtained from the optimal solution; and for those results that do not exactly match the optimal solution, the error is very small. The results also show the efficiency in run time of the iterative LP solution compared to the ILP. For example, the run time for the ARF benchmark under time constraint 19 in Table 2.5 takes a fraction of second for the iterative LP solution compared to 132 seconds for the exact solution; while the two solutions are the same. In case of the TRCS problem, resource constraints might be violated in the LP solution under certain time constraints as shown in Tables 2.8, 2.9, and 2.12. However, the resource constraints are satisfied in the final solution using the power-resources saving procedure as shown in the results.

In addition to the structure of the DFG, the quality of the iterative LP solution is affected by the distance between the time constraint and the length of the critical path, which is reflected in the length of time-frame of each DFG node and so the number of 0-1 associated variables. When the time constraint is very close to the critical path length, the number of variables associated with each node is very small limiting the number of candidate variables that contribute to the objective function minimization. On the other hand, when the time constraint is far from the critical path length, the decision taken at a certain point to fix the schedule of some nodes does not have much impact on those nodes whose time-frames are limited. This is because there is a big chance for each node even after its time-frame is restricted to have a variable mature enough to contribute to the objective function minimization. As the results show, the iterative LP solution is very close and matches in most cases the exact ILP solution at both ends of the results tables, when the time constraint is very close to or very far from the critical path length. The MILP solution run time is not only affected by the structure of the DFG and the number of nodes in it, but it is also affected by how far the time constraint is from the critical path length as shown in most benchmarks especially the EWF benchmark in Tables 2.6, 2.11, and 2.12.

Table 2.3: Modules library for MVS

Module	5.0 V		3.3 V	
	d	power	d	power
MULT16	2	84	4	13
ADD16	1	26	2	6
SUB16	1	26	2	6

Table 2.4: HAL under TCS

L	ILP Sol				Iterative LP Sol					After Power Saving		
	avg	peak	[*,+]	time	avg	peak	[*,+]	#itr	it_t	avg	peak	[*,+]
6	162.33	265	4,3	0.01	162.33	265	4,3	3	0.03	162.33	265	4,3
7	122.57	181	3,3	0.02	122.57	181	3,3	3	0.03	122.57	181	3,2
8	78.25	110	4,3	0.03	78.27	123	4,3	4	0.04	78.27	110	4,3
9	55.4	97	4,2	0.12	56.67	97	4,3	5	0.05	55.4	97	4,3
10	39.4	45	3,3	0.30	39.4	45	3,3	5	0.05	39.4	45	3,3
11	34.82	39	3,3	0.06	34.82	39	3,3	7	0.07	34.82	39	3,3
12	31	39	3,3	0.04	31	39	3,2	8	0.08	31	39	3,3

Table 2.5: ARF under TCS

L	ILP Sol				Iterative LP Sol					After Power Saving		
	avg	peak	[*,+]	time	avg	peak	[*,+]	#itr	it_t	avg	peak	[*,+]
11	225.27	362	6,4	0.03	225.27	388	8,4	2	0.02	225.27	362	6,3
12	215.25	349	5,4	0.22	204.67	388	8,4	6	0.12	204.67	362	6,4
13	154.92	336	6,4	0.65	188.92	388	8,4	7	0.28	187.23	362	6,4
14	142.28	336	8,3	1.86	142.28	348	8,4	7	0.42	142.28	336	8,4
15	103.33	194	8,4	1.74	103.33	336	8,4	6	0.42	103.33	336	6,2
16	95.5	194	8,3	7.49	95.5	336	8,4	5	0.4	95.5	336	6,3
19	54.84	64	4,2	132	54.84	64	4,4	6	0.66	54.84	64	4,2
22	45.36	52	4,3	22.3	44.36	64	4,4	5	0.75	44.36	64	4,3

Table 2.6: EWF under TCS

L	ILP Sol				Iterative LP Sol					After Power Saving		
	avg	peak	[*,+]	time	avg	peak	[*,+]	#itr	it_t	avg	peak	[*,+]
17	111.64	252	3,5	0.02	111.64	258	3,5	5	0.05	111.64	258	3,5
18	105.4	168	2,5	0.06	103	252	3,5	7	0.14	103	252	3,5
20	71.15	110	3,5	1.77	71.7	168	4,5	10	0.7	70.6	168	4,4
21	56.19	97	4,5	3.69	60	107	4,5	10	1.1	60	107	4,5
28	30.7	37	2,5	677	29.14	39	3,5	22	9.68	29.14	39	3,5
34	22.7	26	2,4	3340	22.7	39	2,4	24	16.6	22.7	32	2,4

Table 2.7: HAL [3*, 3+] under TRCS

L	ILP Sol			Iterative LP Sol					After Power Saving		
	avg	peak	time	avg	peak	[*,+]	#itr, #f	it_t	avg	peak	[*,+]
6	183.5	252	0.02	183.5	275	3,3	4,0	0.04	181.67	258	3,3
7	122.57	181	0.02	122.57	181	3,3	3,0	0.03	122.57	181	3,2
8	92.75	181	0.33	92.75	181	3,3	8,0	0.08	92.75	181	3,3
9	56.67	110	0.18	56.67	110	3,3	5,0	0.05	56.67	110	3,2
10	39.4	45	0.30	39.4	45	3,3	4,0	0.08	39.4	45	3,3
11	34.82	39	0.06	34.82	39	3,3	8,0	0.08	34.82	39	3,3
12	31	39	0.05	31	39	3,2	8,0	0.24	31	39	3,3

Table 2.8: HAL [2*, 2+] under TRCS

L	ILP Sol			Iterative LP Sol					After Power Saving		
	avg	peak	time	avg	peak	[*,+]	#itr, #f	it_t	avg	peak	[*,+]
7	155.71	174	0.02	155.71	174	2,2	3,0	0.03	155.71	174	2,2
8	139	168	0.19	137.62	214	2,2	8,0	0.08	133.5	174	2,2
9	95.33	103	0.10	56.67	116	3,2	5,0	0.1	69.55	103	3,2
10	83.6	103	1.34	64.8	113	3,2	4,0	0.08	61.5	103	3,2
11	56.9	97	0.73	56.91	97	2,2	8,0	0.24	56.91	97	2,2
12	50.33	97	1.7	40.67	97	3,2	8,0	0.16	50.33	97	2,2

Table 2.9: ARF [6*, 3+] under TRCS

L	ILP Sol			Iterative LP Sol					After Power Saving		
	avg	peak	time	avg	peak	[*,+]	#itr, #f	it_t	avg	peak	[*,+]
11	225.27	362	0.04	225.27	388	8,4	2,2	0.04	225.27	362	6,3
12	215.25	349	4.2	204.67	388	8,4	6,1	0.12	204.67	362	6,3
13	154.92	336	1.58	190.61	336	4,4	5,0	0.45	188.92	336	4,3
14	142.28	336	2.98	142.28	348	8,4	7,1	0.49	158.85	342	6,3
15	103.33	194	2.12	103.33	336	8,2	6,2	0.72	118.8	336	6,3
16	95.5	194	11.00	95.5	336	8,4	5,0	0.55	95.5	336	6,3
19	54.84	64	477.2	54.84	64	4,4	8,0	0.88	54.84	64	4,2
22	45.36	52	81.73	44.36	64	4,2	6,0	1.2	44.36	64	4,3

Table 2.10: ARF [4*, 2+] under TRCS

L	ILP Sol			Iterative LP Sol					After Power Saving		
	avg	peak	time	avg	peak	[*,+]	#itr, #f	it_t	avg	peak	[*,+]
11	267.45	382	0.02	267.45	382	4,2	2,0	0.02	267.45	382	4,2
12	246	336	0.31	243.33	382	4,2	7,0	0.21	243.33	348	4,2
13	188.92	336	3.24	188.92	348	4,2	5,0	0.3	187.23	342	4,2
14	158.85	336	3.14	178.57	336	4,2	9,0	1.08	174.64	336	4,2
15	132.8	194	44.81	165.2	336	4,2	8,0	0.96	162.26	336	4,2
16	118.62	181	23.63	124.5	348	4,2	6,0	1.02	124.5	348	4,2
19	54.84	64	2.16	54.84	64	4,4	7,1	1.68	54.84	64	4,2
22	45.36	52	26.93	44.36	64	4,2	6,0	1.68	44.36	64	4,3

Table 2.11: EWF [3*, 5+] under TRCS

L	ILP Sol			Iterative LP Sol					After Power Saving		
	avg	peak	time	avg	peak	[*,+]	#itr, #f	it_t	avg	peak	[*,+]
17	111.64	252	0.02	111.64	252	3,5	4,0	0.04	111.64	252	3,5
18	105.44	168	0.09	103	258	3,5	6,0	0.18	103	252	3,5
20	71.15	110	2.69	86.35	187	3,5	12,0	0.84	84.7	187	3,5
21	61.19	107	9.79	60.67	126	3,5	11,0	1.32	60.67	126	3,5
28	30.71	37	1704	29.14	39	3,5	21,0	8.61	29.14	39	3,5
34	22.7	26	12350	22.7	31	2,5	24,0	14.4	22.7	29	2,4

Table 2.12: EWF [2*, 3+] under TRCS

L	ILP Sol			Iterative LP Sol					After Power Saving		
	avg	peak	time	avg	peak	[*,+]	#itr, #f	it_t	avg	peak	[*,+]
18	104.83	174	0.05	104.83	258	3 3	8,0	0.16	103.6	186	2,3
20	80.55	120	5.57	80.55	126	2,3	17,0	1.7	80.55	120	2,3
21	71.19	107	18.67	77.24	136	2,3	19,0	2.47	75.14	119	2,3
28	29.92	38	662	29.92	38	2,3	19,0	10.64	29.92	38	2,3
34	22.7	26	3860	22.7	32	2,3	23,0	18.17	22.7	32	2,3

2.6 Chapter Summary

In this chapter, we have presented an MILP formulation for the scheduling problem using multiple supply-voltages in order to optimize peak power as well as average power and energy consumptions. Our exact solution for optimal peak and/ or average power scheduling is considered under two sets of constraints, time constraint alone (TCS), and time and resource constraints (TRCS). Then, we have devised a two-phase heuristic to solve the multiple supply-voltages scheduling for peak and average power minimization using the same sets of constraints. First, we developed a guided LP relaxation in which the MILP formulation is iteratively relaxed to obtain a minimal peak and/or average power minimization. Then in the second phase, the power-resources saving procedure is developed to restore the violation in resource constraints (if any) in case of TRCS or to achieve minimal resource usage in case of TCS; and to restructure the output LP schedule in order to obtain more power saving. The results for peak and average power of our two-phase heuristic well match those obtained by the optimal solution as have been validated by extensive experiments on several benchmarks.

CHAPTER 3

MODIFIED FORCE-DIRECTED SCHEDULING FOR PEAK AND AVERAGE POWER OPTIMIZATION USING MULTIPLE SUPPLY-VOLTAGES

The scheduling problem in HLS is a well-known NP-complete problem and adding another dimension for supply-voltage selection in the MVS increases the problem complexity. A fast and effective heuristic is needed to search for the solution especially when the design space is explored to trade off one objective for another such as power and execution time. The algorithm in this chapter is a potential solution in this direction; and it considers the same problem addressed in the previous chapter using different technique to solve for MVS. It is based on the idea of distributing the utilization of certain resources to minimize certain cost as introduced by Paulin and Knight [PK89]. The resource to be distributed in the developed algorithm here is the power consumption utilization to reduce the peak power consumption.

3.1 Problem Definition

The input to the problem includes a DFG representation of the design problem, a set of voltage levels for the operating resources, a power/delay table that contains the average power consumption and the delay time needed for each resource operating on each voltage level and the time constraint, λ . The goal is to find a schedule (in which each operation is stamped to a control step, $cstep \in (1, 2, \dots, \lambda)$ and a voltage level from the set of input voltage levels) that minimizes the peak power consumption as well as the average power and energy consumption.

Our solution to the multiple supply voltages scheduling (MVS) problem is a two-phase algorithm. The first phase is a modified power-force-directed scheduling (MPFDS) heuristic based on the force-directed scheduling heuristic introduced by Paulin and Knight [PK89] targeting power (peak and average) and energy consumption minimization. The second phase is the *power-resources saving* procedure that revisits the output schedule from the first phase in

which it tries to further reduce the power and/or the number of operating resources by scheduling DFG operations in lower voltage levels if possible and/or by moving the operations within their new timeframes where possible without violating the peak power obtained from the first phase.

3.2 Basic Force-Directed Scheduling

Force-directed scheduling algorithm FDS introduced by Paulin and Knight [PK89] is a heuristic for time-constrained scheduling, in which, given a DFG representation of the scheduling problem and the maximum time allowed to finish the computations, λ , the goal is to find the minimum number of resources (or area) required. The basic idea of the FDS heuristic is attempting to balance the concurrency of operations in each time step and achieving high utilization of the resources by distributing the operations approximately evenly over the allowed execution time, λ . This is achieved through taking into account the global effect of attempting to schedule an operation at a certain candidate cstep.

All operations are assumed to have unit delays in the FDS. The time-frame of an operation is the set of csteps that start at its earliest time, ASAP, and end at its latest time, ALAP; and mobility of a node i is the length of its time-frame and equals the difference between its ALAP and ASAP time steps plus one. The probability $pr(i)$ of an operation i is the reciprocal of its mobility and it is uniform over its time-frame.

The FDS heuristic starts by calculating the time-frame for each operation, which is the set of candidate csteps at which the operation can be scheduled. The concurrency of similar operations is captured by the distribution graph, which acts as the spring constant in *Hooke's law* analogy ($F = Kx$, K represents the spring constant, x the displacement, and F is the force exerted to cause the displacement). The distribution graph, DG, is calculated in each cstep j and it is the sum of the probabilities of operations at that cstep as follows.

$$DG(j) = \sum_i prob(i, j) , \quad (3.1)$$

where $prob(i, j)$ is the probability of an operation i at cstep j which equals $pr(i)$ at its timeframe and zero elsewhere.

The FDS computes a set of forces for each candidate cstep of each operation. These forces are self force, predecessor force, and successor force. Self force, selfForce, accounts for the

effect of scheduling an operation in a tentative cstep from its time-frame and is given by Equation (3.2). Predecessor and successor forces, $psForce$, account for the effect of tentative assignment of an operation on its predecessors and successors, respectively, because of the change in their time-frames as shown in Equation (3.3).

$$selfForce(i, j) = DG(j) - pr(i) \sum_{j_1=ASAP(i)}^{ALAP(i)} DG(j_1), \quad (3.2)$$

$$\text{and } psForce(l, j) = newpr(l) \sum_{j_2=newASAP(l)}^{newALAP(l)} DG(j_2) - pr(l) \sum_{j_1=ASAP(l)}^{ALAP(l)} DG(j_1), \quad (3.3)$$

where $newASAP$ and $newALAP$ are the restricted ASAP and ALAP, respectively, for the predecessor or successor, l , of the operation being considered, and $newPr$ is the probability of l in the restricted time-frame and is computed in a similar way as the original probability, pr . These three forces are added together to form the total force, $totalForce$.

$$totalForce(i, j) = selfForce(i, j) + \sum_{l \in ps(i)} psForce(l, j) \quad (3.4)$$

In each iteration, the total force is calculated for each operation in each cstep of its time-frame, and then the operation with the smallest $totalForce$ is picked up and stamped to that respective cstep. Then the time-frame for each operation is updated and the process is repeated until all operations are scheduled.

3.3 Modified Power-Force-Directed Scheduling (MPFDS)

The goal of our modified power-force-directed scheduling algorithm (MPFDS) is to minimize power (peak and average) consumption by assigning to each operation the smallest voltage level possible from the input voltage levels without violating the time constraint and at the same time to distributes the DFG operations over the total allowed time in such a way as to balance the power consumptions to achieve minimum peak power. This is achieved by taking into account the global effect of power consumptions in the entire DFG when attempting to schedule an operation in a certain tentative cstep and at a certain voltage level. Our MPFDS algorithm is an extension of but is different from the original form of force-directed scheduling introduced by Paulin and Knight [PK89], in which the basic FDS does not deal with multicycle

operations or variable cost for the same operation since the delay and power consumption are different for each voltage level. Our MPFDS algorithm considers multicycles as well as the variable power and delay of each operation when it is tentatively scheduled with different voltage levels.

A distribution graph called *the power distribution graph*, pDG, is constructed to represent the probabilistic power consumption at each cstep taking into consideration the different voltage levels that an operation can be assigned. The power distribution graph, pDG, at each cstep j is given by:

$$pDG(j) = \sum_i \sum_v prob(i, j, v) * p(i, v), \quad (3.5)$$

where $p(i, v)$ is the power consumed by the resource executing operation i when operating at voltage level v , and $prob(i, j, v)$ is the probability of an operation i to be scheduled at cstep j with voltage level v . Because we deal with multicycle operations, we refer to the last cycle of the operation when we talk about scheduling the operation at a certain cstep j and at the same time all other cycles of the operation are considered at csteps $j-1, j-2, \dots, j-d(i,v)+1$. Thus the probability, $prob(i, j, v)$, is computed in such a way as to capture the possibility that an operation i is active in csteps $j, j-1, j-2, \dots, j-d(i,v)+1$ and the possibility that there is room for operation i to be scheduled with voltage level v . It is computed as the reciprocal of its mobility multiplied by the number of voltage levels with which it can be scheduled. For example, consider an operation i with a delay of 2 cycles if it is executed with voltage level v_1 and 4 cycles if it is executed with voltage level v_2 and its ASAP/ALAP times are 2/5 (ASAP and ALAP are computed using the highest voltage level v_1). Therefore, its mobility is 4 and the last cycle of operation i can be located at csteps 3, 4, 5, or 6 if it scheduled with voltage level v_1 and at csteps 5 or 6 if it scheduled with voltage level v_2 as shown in Figure 2-(a). In this case, the probability of operation i equals $1/(4*2) = 1/8$ at csteps 2, 3, 4, 5, and 6 for both v_1 and v_2 and zero elsewhere. Consider operation i again but with ASAP/ALAP times of 2/3. Therefore, its mobility is 2 and the last cycle of operation i can be located at csteps 3 or 4 if it scheduled with voltage level v_1 but there is no room to be scheduled with voltage level v_2 as shown in Figure 3.1-(b). Thus, the probability is $1/(2*1)$ at csteps 2, 3, and 4 for voltage level v_1 and zero for v_2 .

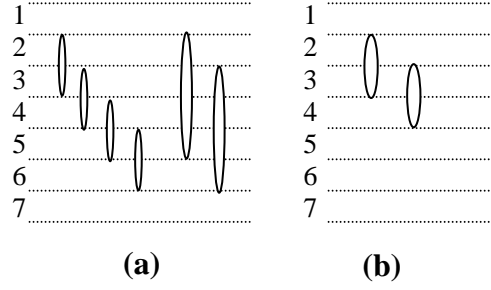


Figure 3.1: Potential schedule of sample operation: **(a)** ASAP/ALAP is 2/5, **(b)** ASAP/ALAP is 2/3.

Our MPFDS algorithm works as shown in Figure 3.2. First, the ASAP and ALAP times for each operation are computed using the delay of the corresponding functional unit when it operates at the highest voltage level followed by computing the probability of each operation using the way discussed previously. Then a power distribution graph is constructed using Equation (3.5). The main loop of the algorithm is the one used to compute the resultant forces when an operation is tentatively scheduled at a cstep within its time-frame and at a possible voltage level. These forces are calculated in such a way as to balance the power distribution over all csteps and at the same time schedule operations with the smallest power possible taking into consideration the global effect when doing so. The total force is the sum of two forces, self force accounting for the effect of power consumption by an operation i when it is tentatively scheduled at cstep j and voltage level v ; and the predecessor/successor force, $psforce$, accounting for the power consumption of predecessors/successors of operation i due to the change of their time-frames. Equations (3.6) and (3.7) show how these forces are calculated and how the effect of power consumption is considered. Finally, the operation with the smallest total force is picked and stamped to the corresponding cstep and the corresponding voltage level. Then, the time-frame for each operation is updated and the process is repeated until all operations are scheduled.

$$\begin{aligned}
 selfForce(i, j, v) = & \sum_{j_2=j-delay(i,v)+1}^j pDG(j_2) * p(i, v) \\
 & - \sum_{j_1} \left(pDG(j_1) \sum_{vv} prob(i, j_1, vv) * p(i, vv) \right)
 \end{aligned} \tag{3.6}$$

$$\text{and } psForce(i, j, v) = \sum_{k \in ps(i)} \left(\begin{aligned} & \sum_{j_4} \left(pDG(j_4) \sum_{vv} newprob(k, j_4, vv) * p(k, vv) \right) \\ & - \sum_{j_3} \left(pDG(j_3) \sum_{vv} prob(k, j_3, vv) * p(k, vv) \right) \end{aligned} \right) \tag{3.7}$$

where $j_1 \in [\text{ASAP}(i) + d(i, v_1) - 1, \text{ALAP}(i)]$, $j_3 \in [\text{ASAP}(k) + d(k, v_1) - 1, \text{ALAP}(k)]$, and $j_4 \in [\text{newASAP}(k) + d(k, v_1) - 1, \text{newALAP}(k)]$; and newASAP , newALAP , and newProb are the ASAP, ALAP, and probability of predecessors/successors of operation i , respectively, due to the change in their time-frames.

Repeat until (all operations are scheduled)

1. Calculate ASAP and ALAP times.
2. Update power distribution graph (pDG) using Equation (3.5).
3. For each operation that is not scheduled yet, calculate self force and predecessor/successor forces for each voltage-level v and at each tentative cstep using Equations (3.6) and (3.7) respectively.
4. Add the computed self forces and predecessor/successor forces together to form the total forces.
5. Schedule operation with the lowest total force at the corresponding cstep using the associated voltage-level.

End Repeat

Figure 3.2: MPFDS algorithm.

3.3.1 Time Complexity of MPFDS

The worst-case time complexity of our modified power force directed scheduling algorithm (MPFDS) is in the order of $O(\lambda V^2 n^3)$, where λ is the total time constraint, V is the number of input voltage-levels, and n is the number of operations in the DFG. In practice, because the usual number of operating voltage-levels in a circuit is two or three, this time complexity is reduced to $O(\lambda n^3)$ same as the basic FDS algorithm. Here is the derivation for this time complexity.

1. There is at least one operation scheduled in each iteration. Since scheduling an operation affects mobility of other operations forcing their time-frames to be ones (same ASAP and ALAP values for each operation), they are scheduled in the same iteration too. Thus, there are at most n iterations.
2. At each iteration there are at most n unscheduled operations that must be considered for force calculations.
3. Forces for each of these unscheduled operations are calculated for each potential voltage-level and for each cstep within its time-frame, which requires $O(\lambda V n)$ calculations.

Because the time-frame for an operation is at most λ and the number of potential voltage-levels is at most V (a voltage-level is excluded for an operation when there is no room for that operation to be scheduled using that voltage-level).

4. For each potential voltage-level and for each tentative cstep of an operation to be scheduled at, there are at most $n-1$ predecessors/successors affected. This requires $O(Vn)$ calculations because their forces need to be calculated for each voltage-level.

3.4 Power-Resources Saving

The power-resources saving algorithm works as a post-processing step for the output of the MPFDS heuristic to gain more power and/or resources saving. This is achieved through exploiting any room for an operation within its time-frame and trying to schedule it with a lower voltage level (more power saving) and/or moving it up and down within the available room without violating the peak power obtained from the first phase. The algorithm for power-resource saving is the same as the one introduced in Chapter 2, Figure 2.6.

The efficiency of the power-resources saving procedure to further reduce the peak power consumption exploiting the flexibility in the time-frames of the DFG operations is demonstrated using the example in Figure 3.3. The two schedule in Figure 3.3-(a) and Figure 3.3-(b) are for HAL benchmark stated earlier in Section 2.4, Figure 2.7-(a) using the same module library from Table 2.3. Figure 3.3-(a) is the resultant schedule form the MPFDS technique, while Figure 3.3-(b) is the modified schedule after the power-resources saving.

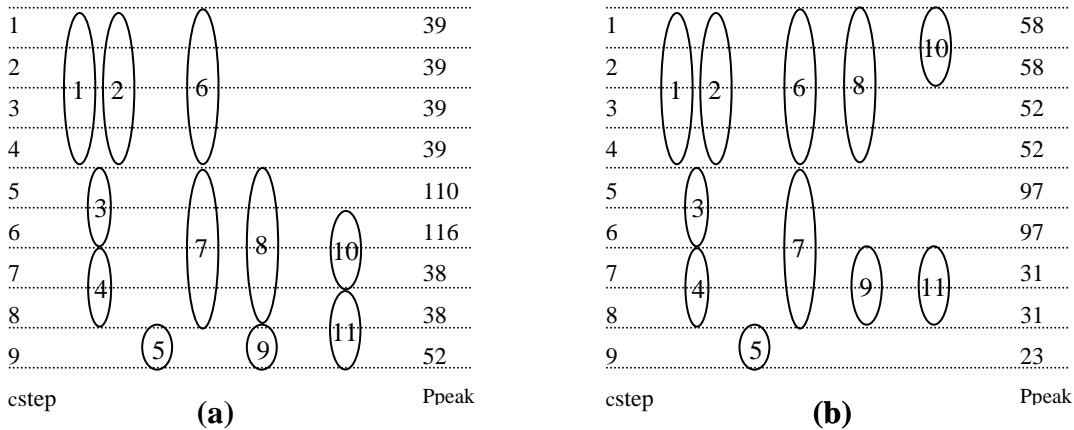


Figure 3.3: Effect of power-resources saving procedure: (a) resultant schedule of the MPFDS heuristic, (b) schedule after power-resources saving procedure.

In Figure 3.3-(a), operations 8 and 10 can start earlier at cstep 1 where the peak power consumption is low making them to be candidates to be pushed back away from csteps 5 and 6 where the peak power consumption is high. As a result, a room is emerged for operation 9 to be scheduled at a lower voltage level. This is completely achieved by the power-resources saving procedure as shown in Figure 3.3-(b). The peak power consumption is reduced from 116 μ watt to 97 μ watt, and this coincides with the optimal solution in this case.

3.5 Experimental Results

Our presented algorithm is tested on standard benchmarks like HAL, ARF, and EWF using the same module library in Table 2.3. Table 3.1 shows peak power, average power, and the number of resources for each benchmark with time constraint varying from the critical path length to twice the critical path length after each phase of the algorithm. Results are compared to the optimal solution (ILP solution) as tabulated in Table 3.1 showing that in most cases the results of our heuristic well match those obtained from the optimal solution especially for the time constraints located between the critical path length and around 1.5 times the critical path length. In some cases, the number of resources resulting from the optimal solution is more than that obtained from the MPFDS heuristic because the ILP solution is a time-constrained scheduling and does not take into consideration resource minimization, while phase II of our algorithm exploits any opportunity to get a smaller number of resources. Table 3.1 also shows the benefits of post processing the output of the MPFDS heuristic with the power-resources saving procedure in both power and resource minimization. In many cases, the power-resources saving algorithm brings the output of MPFDS back to match the optimal solution or to be within a small error for average and/or peak power and it reduces the number of resources required even when there is no room for more power reduction. Results for the power-resources saving algorithm in Table 3.1 are obtained with no constraint on the resources obtained from the first phase. In some cases, the resultant resources after the power-area saving might get higher than that obtained from the MPFDS phase to get more power reduction as in the HAL benchmark under time constraint 9, the ARF benchmark under time constraint 16, and the EWF benchmark under time constraint 18 in Table 3.1. The power-resources saving algorithm can eliminate this problem by keeping the resources obtained from the MPFDS phase inviolated in the power-resources saving post processing. This is achieved in the cost of less power savings in some cases as shown in Table 3.2 for the cases mentioned above.

Table 3.1: Peak and average power results for ILP solution and the two-phase heuristic

problem	L	ILP			MPFDS			Power area saving		
		avg	peak	[*,+]	avg	peak	[*,+]	avg	peak	[*,+]
HAL	6	162.33	265	4,3	166	265	4,3	162.33	265	4,3
	7	122.57	181	3,3	125.71	187	3,3	124.14	181	3,3
	8	78.25	110	4,3	94.12	181	3,3	92.75	181	3,3
	9	55.44	97	4,2	56.67	116	3,3	55.44	97	4,3
	10	39.4	45	3,3	40.5	49	3,2	40.5	46	3,2
	11	34.82	39	3,3	34.82	45	3,3	34.82	45	3,3
	12	31	39	3,3	31.92	39	3,3	31	39	3,3
ARF	11	225.27	362	6,3	227.27	375	7,4	225.27	362	6,3
	15	103.33	194	6,4	103.33	207	7,4	103.33	194	6,4
	16	95.5	194	8,3	96.87	200	8,2	96.19	194	8,3
	18	59.11	65	5,4	72	194	6,3	71.4	194	6,3
	22	45.36	52	4,2	57.91	194	5,4	56.91	194	4,3
EWF	17	111.65	252	3,5	106.12	265	4,5	104.82	265	4,5
	18	99	181	3,5	93.78	194	4,4	91.33	194	4,5
	21	62.24	110	3,5	66.7	116	3,5	65.67	116	3,5
	28	30.7	37	2,5	35.25	97	2,3	34.86	97	2,3
	34	22.7	26	2,4	29	84	2,3	28.38	84	2,3

Table 3.2: Comparison between power-resources saving with and without resource constraints

problem	L	MPFDS			Phase II with no resource constraints			Phase II with resource constraints		
		avg	peak	[*,+]	avg	peak	[*,+]	avg	peak	[*,+]
HAL	6	166	265	4,3	162.33	265	4,3	162.33	265	4,3
	7	125.71	187	3,3	124.14	181	3,3	124.14	181	3,3
	8	94.12	181	3,3	92.75	181	3,3	92.75	181	3,3
	9	56.67	116	3,3	55.44	97	4,3	56.67	110	3,2
	10	40.5	49	3,2	40.5	46	3,2	40.5	46	3,2
	11	34.82	45	3,3	34.82	45	3,3	34.82	45	3,3
	12	31.92	39	3,3	31	39	3,3	31	39	3,3
ARF	11	227.27	375	7,4	225.27	362	6,3	225.27	362	6,3
	15	103.33	207	7,4	103.33	194	6,4	103.33	194	6,4
	16	96.87	200	8,2	96.19	194	8,3	96.19	200	8,2
	18	72	194	6,3	71.4	194	6,3	71.4	194	6,3
	22	57.91	194	5,4	56.91	194	4,3	56.91	194	4,3
EWF	17	106.12	265	4,5	104.82	265	4,5	104.82	265	4,5
	18	93.78	194	4,4	91.33	194	4,5	91.94	194	4,4
	21	66.7	116	3,5	65.67	116	3,5	65.67	116	3,5
	28	35.25	97	2,3	34.86	97	2,3	34.86	97	2,3
	34	29	84	2,3	28.38	84	2,3	28.38	84	2,3

3.6 Chapter Summary

In this chapter, we have again considered the problem of peak and average power minimization in the scheduling in HLS with multiple supply voltages under time constraint. We have proposed a two-phase heuristic to get a near-optimal solution in a small amount of time. The first phase is a modified power-force-directed scheduling (MPFDS) heuristic based on the known basic force-directed scheduling. The MPFDS tries to minimize power (peak and average) consumption through assigning each operation the smallest voltage level possible from the given voltage levels without violating the time constraint and at the same time distributing the DFG operations over the entire allowed time to balance the power consumptions and to achieve minimum peak power. The second phase is a preprocessing procedure (power-resources saving) that is a revisit of the output schedule from the first phase to exploit the available room to get more power and/or operating resource minimization. Results show that our proposed heuristic is capable of achieving near-optimal results with polynomial time complexity.

CHAPTER 4

MODULE SELECTION AND SCHEDULING UNIFICATION FOR PEAK AND AVERAGE POWER OPTIMIZATION

The techniques developed for low-power design so far consider the scheduling task in HLS, in which the type of functional unit for each DFG operation realization is fixed. Thus the execution delay (for a given voltage level) and the area requirement for each DFG operation is known apriori. For example, an addition operation may be mapped to a carry-look-ahead adder for its realization. In this chapter we address the module selection task for low-power design, which is the inter-dependent on the scheduling task.

Module selection is the problem of allocating to each computational element in the DFG a module type from a given module library that matches its functionality and satisfies the design requirements. Peak power minimization plays a substantial role in low-power design because it has a direct effect on battery lifetime as well as on the reliability of the integrated circuit. Therefore, peak power should be included in the module selection process as well as the average power.

A design problem is represented by a data-flow graph, $G=(V, E, T)$ in which each vertex $v \in V$ represents a computational operation, edges E represent the data flow between these computational operations, and T is the set of operation types, which define the set of functional units on which an operation can be realized. Each operation is implemented through a logic abstraction called a *functional unit* while the library component that is used to realize the functional unit is called a *module*. For example, addition can be implemented by an adder, where ripple-carry adder and carry-look-ahead adder are two module types that can realize the adder.

4.1 Problem Definition

Given a DFG representation of the design problem, $G(V, E, T)$, the time constraint, λ , an input module library in which each module is characterized by the execution time, the average

power consumption and its size as the number of area units, the task is to find a schedule (in which each operation is stamped to a control step, $cstep \in (1, 2, \dots, \lambda)$ and a module from the set of input modules) that minimizes the peak power consumption as well as the average power and energy consumption according to one of the sets of constraints such as time-constrained module-selection and scheduling (TCMSS), and time and area constrained module-selection and scheduling (TACMSS).

Because of the inter-dependent nature of the scheduling and module selection processes, optimizing the design for each task alone achieves only local optimality, while considering both tasks concurrently is a big step toward the global optimality. We propose two solutions for the unified scheduling and module selection problem targeting peak power consumption as well as other design factors such as average power and energy consumption, and area as shown in Figure 4.1. In Figure 4.1, (1) is an exact solution based on a mixed integer linear programming (MILP) formulation, and (2) is a two-phase heuristic that first obtains a guided iterative relaxed LP solution of the MILP formulation followed by a *power-area saving* procedure which revisits of the output schedule from the first phase in which it tries to minimize the power and/or the area further by scheduling operations in a less power-expensive module if possible and/or moving the operations within their new time-frames if possible without violating the peak power obtained from the first phase.

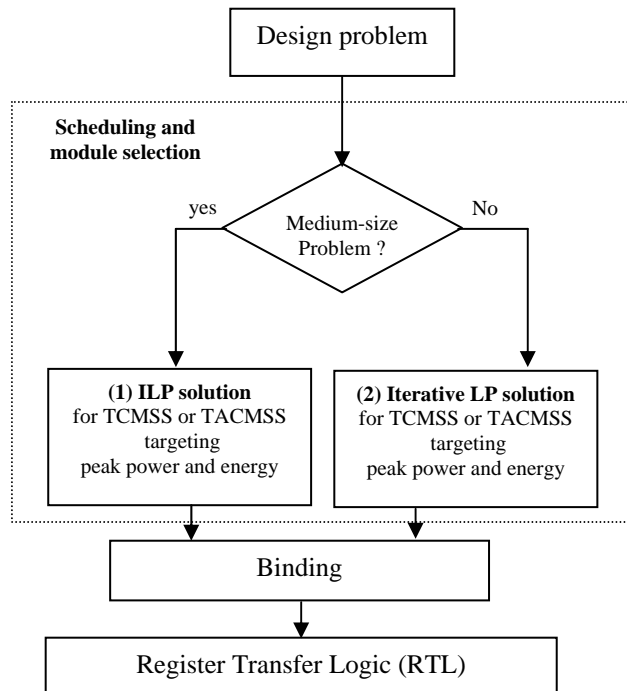


Figure 4.1: Flow-chart for the scheduling and module selection process.

Our MILP formulation addresses peak power consumption, which is not considered in the previous work (to the best of our knowledge), and considers the scheduling and Module selection tasks for power minimization simultaneously. In addition, we present two solutions, an optimal solution for medium size problems and a heuristic approach for large problems in order to suite a large variation of design problems. Our formulation adopts an unrestricted library in which a computational element can be mapped to many module types and some module types can realize many operation types. For example, an addition operation can be mapped to an ALU, a ripple-carry adder, or carry-look-ahead adder; and at the same time, an ALU can implement addition operations as well as subtraction and comparison operations. For efficient design space exploration, the input module library is assumed to include modules with varying values of delay, area, and power consumption.

4.2 Exact Solution

Our proposed optimal algorithm for the unified module selection and scheduling for power minimization problem (UMSSP) is shown in Figure 4.2. It assumes that the clock selection has already been done and so the input delays for DFG nodes are represented in terms of number of csteps. First, the as-soon-as-possible (ASAP) and as-late-as-possible (ALAP) schedules (computed using the highest-speed modules) are calculated as a preprocessing step to tighten the time-frame for DFG vertices and so the number of variables in the MILP formulation. Then, the MILP formulation is developed to solve one of the two problems: (i) the TCMSS problem using the objective function (4.1) and the set of inequalities (4.2)-(4.4), or (ii) the TACMSS problem using the objective function (4.1) and the set of inequalities (4.2)-(4.6). The development and the explanation of the sets of constraints for these problems will be clear in the text later.

- *Calculate ASAP and ALAP using the highest-speed modules.*
- *Construct the MILP formulation as in Equations (4.1)-(4.6).*
- *Use an ILP solver to solve the model.*
- *Construct the optimal schedule.*

Figure 4.2: Unified module selection and scheduling for power minimization.

4.2.1 TCMSS Problem

Given the time constraint λ and an input module library that includes matching module types with a variety of delays, areas, and power consumptions, allocate to each operation a module

type and a schedule (starting control step) that minimizes peak power and/or energy consumptions within the given time constraints.

4.2.2 TACMSS Problem

The input to this problem are the time constraint λ , the area constraint, and an input module library that includes matching module types with a variety of delays, areas, and power consumptions. The goal is to allocate a module type and a schedule (starting control step) for each operation that minimizes peak power and/or energy consumptions so that the given constraints are satisfied.

Define x_{ijm} to be a 0-1 unknown variable that takes value 1 if node i starts execution at cstep j using module type m and 0 otherwise. Then, the MILP formulation is as follows.

- **Objective Function**

The objective is a weighted sum of both peak power and average power consumptions. The flexibility in the choice of weight factors can be used to achieve certain design requirements. Thus, objective function is expressed as:

$$\text{Minimize: } \alpha P_{peak} + \beta (\frac{1}{\lambda}) \sum_i \sum_j \sum_{m \in T(i)} x_{ijm} d(i, m) p(i, m) \quad (4.1)$$

where P_{peak} will be used as a variable in the constraints set. Note that by treating $m = 0$ to indicate software, the hardware-software codesign problem can be addressed easily.

- **Uniqueness Constraints**

Each node has to start at only one cstep within its time-frame and be scheduled using one and only one matching module type. Inequality (4.2) is used to model this constraint.

$$\sum_{m \in T(i)} \sum_j x_{ijm} = 1 \quad \forall i \in V, j \in R(i) \quad (4.2)$$

- **Precedence Constraints**

The fact that a node can start only after all its predecessors are finished, is modeled as precedence relations, one for each edge as in inequality (4.3).

$$\sum_{m \in T(i)} \sum_j (j + d(i, m) - 1) x_{ijm} + \sum_{m \in T(l)} \sum_j j x_{ljm} \leq -1 \quad \forall (i, l) \in E \quad (4.3)$$

- **Peak power Constraints**

The maximum power consumption (Peak power) is modeled by using a single variable that constrains the power consumption of all active nodes at each cstep. It can be set as a

constraint or can be used as a variable to be minimized in the objective function as shown in inequality (4.4).

$$\sum_i \sum_{m \in T(i)} \sum_{j_1=j-d(i,m)+1}^j x_{ij_1m} p(i, m) \leq P_{peak} \quad \forall j \in [1, \lambda] \quad (4.4)$$

▪ Area Constraints

A single variable is used to model the number of instances used from each module type through constraining the number of active nodes with matching module type at each cstep as in inequality (4.5). Then, the total area constraint is posed through forcing the summation of the number of instances used from each module type weighted by their size to be less than that area constraint as shown in inequality (4.6).

$$\sum_{\substack{i \\ m \in T(i)}} \sum_{j_1=j-d(i,m)+1}^j x_{ij_1m} \leq r_m \quad \forall j \in [1, \lambda] \text{ and module type } m \quad (4.5)$$

$$\sum_{m=1}^M r_m \text{ area}(m) \leq A \quad (4.6)$$

4.3 Iterative LP Relaxation

The solution of the MILP formulation above suffers from large execution-times as the design problem becomes large because of the exponential nature of the ILP solution algorithms. With an acceptable quality of the final solution, LP relaxation can be a solution for this problem. LP relaxation of the IP problem in general does not lead to integral solution, thus, there should be some way to guide the relaxation to get a solution quality close to the optimal solution. The integrality constraint of the 0/1 variables can be relaxed by Equation (4.7).

$$0 \leq x_{ijm} \leq 1 \quad (4.7)$$

The fractional values that result from running the LP solution once by itself do not reflect meaningful information. Just rounding off the fractional value associated with the 0-1 variable is not a good idea because: (i) the correctness is not guaranteed (for example, precedence relations among the nodes might be violated), and (ii) even if the precedence relations are met, the final solution is far from the optimal one.

We devise a guided way of relaxation called *iterative LP relaxation* as shown in Figure 4.3, which is similar to the one presented in Section 2.3. The idea is to develop the LP solution

iteratively in several stages. In each stage, the variables (especially the 0-1 variables) that are close enough to 1 to contribute to the optimal solution are selected. If their resultant values are integral, they are set to these values and fixed during the successive iterations. If their resultant values are fractional, they are tested against a threshold value, and any 0-1 variable that passes the test (i.e., higher than the threshold) it is set to 1 and fixed during the successive iterations. At the same time the rest of the 0-1 variables associated with the same node are set to zeros. The solution iterates until all the 0-1 variables pass the threshold test. The threshold value is set dynamically as the maximum value of the 0-1 variables from the resultant solution of the current LP iteration. This is done to choose the most mature 0-1 variables (variables with largest fractional values, which if they are set to 1's, they lead to optimal or near-optimal solution) to contribute to the final solution.

During the constructive process of the LP solution in the case of time and area constraints, the accumulated candidate 0-1 variables that have been set to one might force the LP solution to become infeasible in some iteration and hence violate the area constraint. Thus at any iteration during the iterative relaxation, if the resultant LP solution is infeasible, the area constraint is relaxed and increased by a small amount and the model is solved again. The power-area-saving phase of the heuristic takes care of the area constraints and tries to restore it back to be within the input constraints.

1. Calculate ASAP and ALAP using the highest-speed modules.
2. Construct the MILP formulation as in equations (4.1)-(4.6).
3. Relax the integrality of the 0-1 variables using (4.7).
4. Iterative procedures:
 - 4.1 Use an LP solver to solve the model.
 - 4.2 In case of an area constraint, if the solution is infeasible, increase the area by a small amount and solve again.
 - 4.3 Set the threshold value to be the maximum of the 0-1 variables resultant values.
 - 4.4 If any 0-1 variable passes the threshold do:
 - 4.4.1 Set its value to 1 and fix it during the successive iterations.
 - 4.4.2 Set all the 0-1 variables associated with the same node to 0.
 - 4.5 Update ASAP and ALAP values.
 - 4.6 If NOT all 0-1 variables are set (to either 0 or 1) GoTo 4.1.
5. Construct the schedule.

Figure 4.3: Iterative LP procedures.

Although, the worst case number of iterations is the number of nodes in the DFG, the actual number of iterations is observed to be very small from our experimental results for most

benchmarks. This is because, at each iteration many variables pass the threshold, and in addition, the time-frames become tighter forcing many variables to settle.

Consider the DFG example in Figure 4.4 using the module library in Table 4.3, the associated 0-1 variables for each DFG node for time constraint 4 are shown in Table 4.1. The iterative LP procedure for the LP formulation in Figure 4.5 (Equations (4.1) through (4.4)) takes 3 iterations to complete the solution. The outcome of each iteration is as shown in Table 4.2. After each iteration, the threshold value is set to the maximum value among the 0-1 variables that are written in bold face in Table 4.2. After the first iteration, x8 variable is already one and the rest of the associated variables of node c, x7 and x9, are zeros forcing node c to be scheduled at cstep 4 using the carry-look-ahead adder. Node b is scheduled at cstep3 with the carry-look-ahead adder after its associated variable x5 passes the threshold value in the second iteration and is set to one. In the third iteration, x1 associated with node a and x11 associated with node d pass the threshold value and they are set to ones forcing nodes a and d to be scheduled at csteps 1 and 2 respectively, using the carry-look-ahead adder. The final LP solution results in the optimal peak power schedule of 10.5 μ watts and 6.6 units of area (one carry-look-ahead adder) as shown in Figure 4.4-(b).

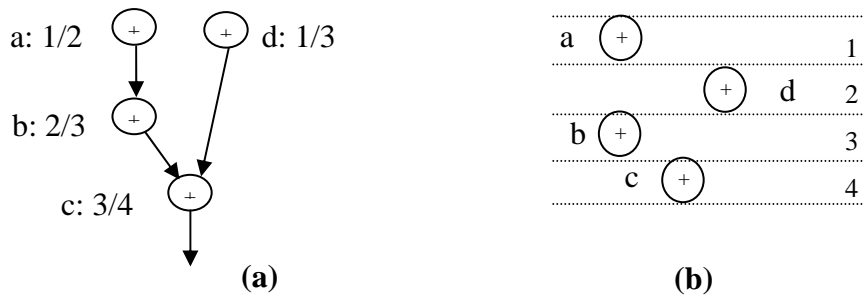


Figure 4.4: A DFG example: (a) the DFG annotated with ASAP/ALAP values for $\lambda = 4$, (b) resultant LP schedule with peak power = 10.5 μ watts and 6.6 units area.

Table 4.1: DFG nodes and their associated 0-1 variables for Figure 4.4-(a)

node	indexed variables	ILP solver variables
a	x_{a11} x_{a21} x_{a12}	x1 x2 x3
b	x_{b21} x_{b31} x_{b22}	x4 x5 x6
c	x_{c31} x_{c41} x_{c32}	x7 x8 x9
d	x_{d11} x_{d21} x_{d31} x_{d12} x_{d22}	x10 x11 x12 x13 x14

Minimize:
obj: 2.625 x1 + 2.625 x2 + 2.7 x3 + 2.625 x4 + 2.625 x5 + 2.7 x6 + 2.625 x7
+ 2.625 x8 + 2.7 x9 + 2.625 x10 + 2.625 x11 + 2.625 x12 + 2.7 x13 + 2.7 x14
+ Ppeak

Subject To:

c1: x1 + x2 + x3 = 1
c2: x4 + x5 + x6 = 1
c3: x7 + x8 + x9 = 1
c4: x10 + x11 + x12 + x13 + x14 = 1

c5: x1 + 2 x2 + 2 x3 - 2 x4 - 3 x5 - 2 x6 <= -1
c6: 2 x4 + 3 x5 + 3 x6 - 3 x7 - 4 x8 - 3 x9 <= -1
c7: - 3 x7 - 4 x8 - 3 x9 + x10 + 2 x11 + 3 x12 + 2 x13 + 3 x14 <= -1

c8: 10.5 x1 + 5.4 x3 + 10.5 x10 + 5.4 x13 - Ppeak <= 0
c9: 10.5 x2 + 5.4 x3 + 10.5 x4 + 5.4 x6 + 10.5 x11 + 5.4 x13 + 5.4
x14 - Ppeak <= 0
c10: 10.5 x5 + 5.4 x6 + 10.5 x7 + 5.4 x9 + 10.5 x12 + 5.4 x14 - Ppeak <= 0
c11: 10.5 x8 + 5.4 x9 - Ppeak <= 0

Figure 4.5: Complete listing of constraints and the objective for the DFG in Figure 4.4-(a).

Table 4.2: Gradually iterative LP solution for the DFG in Figure 4.4-(a)

node	var	itr#1	itr#2	itr#3	final sol
a	x1	0.56276	0.61460	0.52133	1
	x2	0.43724	0.38540	0.47867	0
	x3	0	0	0	0
b	x4	0.23329	0.16554	0	0
	x5	0.76671	0.83446	1	1
	x6	0	0	0	0
c	x7	0	0	0	0
	x8	1	1	1	1
	x9	0	0	0	0
d	x10	0.43724	0.38540	0.47867	0
	x11	0.32946	0.44906	0.52133	1
	x12	0.23329	0.16554	0	0
	x13	0	0	0	0
	x14	0	0	0	0
peak power	Ppeak	10.5	10.5	10.5	10.5

4.4 Power-Area Saving

The goal of the second phase of the algorithm is to gain more power and/or area saving through exploiting any available flexibility for each DFG operation. This phase tries to schedule an operation with a less area-intensive module (more power saving) and/or by moving an operation up and down within the available room without violating the peak power obtained

from the first phase, to get more peak power saving and/or area saving. The time complexity of the power-area saving algorithm is $O(n^2)$ same as that of the power-resources saving algorithm in chapter 2, and the algorithm is shown in Figure 4.6. The input to this algorithm is the resultant schedule attributes from the first phase (MPFDS) where, *scheduleStep* and *Op-module* are the cstep and the library module stamped to each operation, respectively; and peak-power is resultant peak power consumption from the first phase.

```

Power-area saving( scheduleStep, Op-module, peak_power )
marked = 0 for all operations.
count = 0
while(count < numOperations) do
  for(op = 1: numOperations)
    5. if ( marked(op) = 1 or indegree(op) > 0)
      skip the rest of loop body.
    6. update the time frames
    7. compute the room of op using scheduleStep
      of its predecessors and its successors.
    8. for(md =least power-consuming matching module : most power-
      consuming matching module )
      4.2 if (there is a room to schedule op with md without violating
      the peak power and the input area constraints if any)
        4.1.1 schedule op at cstep within its time frame to get
          smaller power and/or area and set:
        4.1.2 marked(op) = 1.
        4.1.3 Op-module(op) = md.
        4.1.4 scheduleStep(op) = cstep.
        4.1.5 count = count + 1.

```

Figure 4.6: Power-area saving algorithm.

As an example, consider the DFG for the HAL benchmark stated earlier in Section 2.4, Figure 2.7-(a). Assume that the input module library is as shown in Table 4.3; Figure 4.7-(a) is the resultant schedule from the iterative LP approach, while Figure 4.7-(b) is the modified schedule after the power-area saving procedure. The power-area saving heuristic exploits the available room in the time-frame of operation 8 to move it from the high peak power region (csteps 1 and 2) to the low peak power region (csteps 3 through 6). At the same time, operations 9, 10, and 11 are re-allocated to a lower power and a lower area module (ripple-carry adder). This results in a reduction in peak power consumption from 235.1 μ watt to 209.8 μ watt. In addition, the configuration of the allocated resources has changed from two ‘‘Carry Look-ahead’’

adder modules in Figure 4.7-(a) to one “Carry Look-ahead” adder module and two “Ripple-Carry” adder modules (area has been reduced from 110.6 units to 106.9 units). This example demonstrates the efficiency of the power-area saving procedure to further reduce the peak power consumption as well as the total area usage.

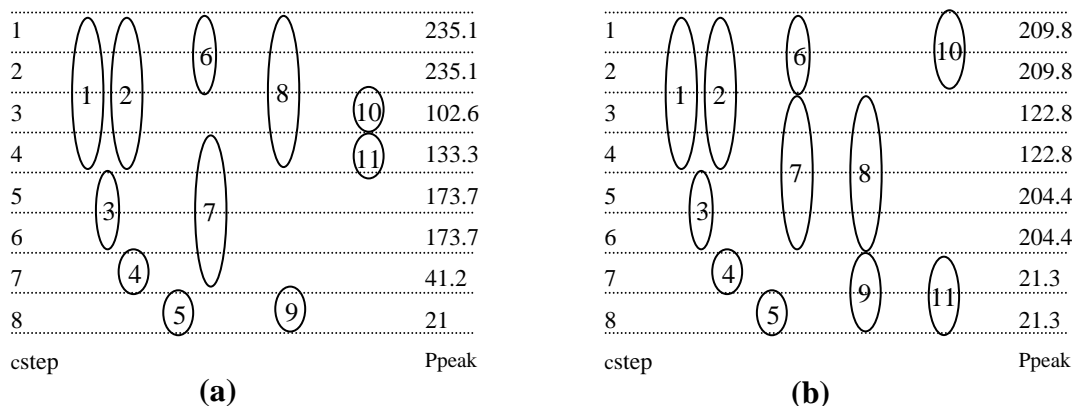


Figure 4.7: Power-area saving procedure example: (a) resultant iterative LP schedule, (b) schedule after power-area saving procedure.

4.5 Experimental Results

Our proposed MILP formulation for the optimal solution as well as the two-phase heuristic, are tested on standard benchmarks like HAL, ARF, EWF, and FDCT using the module library shown in Table 4.3 from [Shn97]. In the tabulated results, notations are used to describe certain quantities such as “L” for the input time constraint in number of csteps, “avg” for average power, “peak” for peak power, “area” for the number of area units used, and “#itr” is the number of LP iterations until the iterative LP solution is finalized, while “time” is the CPU time in seconds needed by the “CPLEX” solver [CPX02] to get the final results. Experiments are run for each benchmark with time constraint varying from the critical path length to twice the critical path length in order to cover a long range of time constraints during design space exploration. Optimal results from the MILP solution are tabulated under the heading “ILP Sol”; while the results from the two-phase heuristic after each phase of the algorithm are tabulated under the headings “Iterative LP Sol” and “After Power Saving” respectively. Tables 4.4 through 4.7 show the results of the tested benchmarks for the TCMSS problem. The results of the TACMSS problem are shown in Tables 4.8 through 4.15 under different sets of area constraints. The results of the iterative LP approach followed by the power-area-saving procedures are compared to the

optimal solution (ILP solution) as tabulated. The results show that in most cases, the results of our heuristic match well those obtained from the optimal solution; and for those results that do not exactly match the optimal solution they are very close. The allowed violation in area constraint during the construction of the relaxed LP solution is restored in most cases using the power-area saving procedure as shown in the results. However, for those cases when the power-area saving procedure failed to bring back the area to satisfy the input constraint, the area penalty is very small. The results also show the efficiency in run time of the iterative LP solution compared to the ILP. For example, the run time for the ARF benchmark under area constraint 260 and time constraint 22 in Table 4.10 takes a fraction of a second for the iterative LP solution compared to 246 seconds of the CPU time for the exact solution; while the two solutions are the same.

Table 4.3: Module library for module selection

Module	del	power	area
Array Multiplier	2	143	32.1
Booth Multiplier	4	30.7	16.4
Carry Look-ahead Adder	1	10.5	6.6
Ripple-Carry Adder	2	5.4	1.3

Table 4.4: HAL under TCMSS

L	ILP Sol				Iterative LP Sol					After Power Saving		
	avg	peak	area	time	avg	peak	area	#itr	time	avg	peak	area
6	294.7	429	116.1	0.01	294.7	439.5	109.5	6	0.06	294.9	434.4	105.5
8	139.4	204.4	110.9	0.03	139.4	235.1	110.9	7	0.07	139.6	209.8	106.9
9	124	148.4	118.8	0.11	105.8	173.7	110.9	7	0.07	105.9	173.7	106.9
10	78.93	92.1	69	0.04	78.93	92.1	62.4	7	0.07	78.99	92.1	58.4
11	71.75	92.1	69	0.16	71.75	92.1	69	7	0.14	71.84	92.1	66.3
12	65.77	92.1	69	0.21	65.77	102.6	62.4	8	0.16	65.87	97.5	59.7

Table 4.5: ARF under TCMSS

L	ILP Sol				Iterative LP Sol					After Power Saving		
	avg	peak	area	time	avg	peak	area	#itr	time	avg	peak	area
11	397.8	602.7	171.2	0.07	427.5	593	154.8	2	0.02	427.5	593	142.9
12	351	602.7	174.4	0.04	337.4	694.8	220.4	10	0.2	337.5	638.8	183.6
13	261.3	572	253.2	0.54	311.5	694.8	220.4	8	0.32	311.6	577.4	209.8
14	242.7	572	286	1.42	242.6	572	286	7	0.35	242.7	572	275.4
15	204.7	316.7	205.4	1.53	226.4	572	286	6	0.3	182.9	572	275.4
16	212.3	291.4	223.1	8.45	171.5	572	286	8	0.64	171.5	572	243.9
19	110	133.3	92	36.2	110	143.8	92	6	0.3	110	143.8	81.4
22	95	122.8	92	88.2	95	122.8	92	7	0.84	95	122.8	84

Table 4.6: EWF under TCMSS

L	ILP Sol				Iterative LP Sol					After Power Saving		
	avg	peak	area	time	avg	peak	area	#itr	time	avg	peak	area
17	150.6	429	122.7	0.02	150.6	429	122.7	5	0.1	150.7	429	125.3
18	142.3	286	97.2	0.04	142.3	429	122.7	7	0.14	124.2	347.4	128.6
20	111.7	173.7	81.5	1.81	103.6	204.4	91.3	16	0.8	103.7	204.4	95.2
21	90.9	153.5	130.7	1.76	90.87	286	156.2	11	0.66	75.4	286	158.8
28	44.8	61.4	59.2	25.8	44.84	92.1	82.2	21	2.73	44.9	92.1	74.2
34	36.9	61.4	65.8	11e3	36.9	71.9	59.2	25	4.75	37.1	66.8	48.6

Table 4.7: FDCT under TCMSS

L	ILP Sol				Iterative LP Sol					After Power Saving		
	avg	peak	area	time	avg	peak	area	#itr	time	avg	peak	area
8	620.2	1174	312.8	0.02	620.2	1174	326	6	0.06	620.3	1174	324.6
9	460.6	1144	394.8	0.04	460.6	1318	426.9	9	0.18	460.9	1303	432.1
10	300.3	368.4	313.8	0.60	300.3	399.1	330.2	10	0.4	300.5	399.1	324.8
12	195.8	276.3	200.4	0.86	195.8	276.3	193.8	9	1.08	195.9	276.3	192.4
14	179.5	245.6	216.1	66.5	167.9	276.3	200.4	11	1.12	168.1	276.3	195
15	167.6	214.9	199.7	415	156.7	307	216.8	12	1.5	156.9	245.6	176

Table 4.8: HAL 120 under TACMSS

L	ILP Sol			Iterative LP Sol					After Power Saving		
	avg	peak	time	avg	peak	area	#itr	time	avg	peak	area
6	294.75	429	0.02	294.75	439.5	109.5	6	0.06	294.9	434.4	105.5
7	206.01	316.7	0.03	229.33	316.7	93.8	7	0.07	206.1	316.7	111.5
8	139.46	204.4	0.04	139.46	235.1	104.3	7	0.14	139.54	235.1	106.9
9	124	148.4	0.14	105.83	173.7	104.3	7	0.07	105.93	173.7	106.9
10	78.93	92.1	0.05	78.93	92.1	69	7	0.07	78.99	92.1	58.4
11	71.755	92.1	0.06	71.755	92.1	69	6	0.18	71.836	92.1	66.3
12	65.775	92.1	0.07	65.775	92.1	62.4	9	0.27	65.875	92.1	59.7

Table 4.9: HAL 100 under TACMSS

L	ILP Sol			Iterative LP Sol					After Power Saving		
	avg	peak	time	avg	peak	area	#itr	time	avg	peak	area
7	252.73	291.4	0.10	229.33	316.7	93.8	7	0.14	229.41	316.7	89.8
8	159.86	235.1	0.20	139.46	204.4	110.9	7	0.14	139.57	209.8	106.9
9	124	179.1	0.36	105.83	184.2	104.3	7	0.14	105.93	173.7	106.9
10	78.93	92.1	0.05	78.93	92.1	62.4	6	0.12	78.99	92.1	58.4
11	71.755	92.1	0.07	71.755	92.1	69	7	0.21	71.836	92.1	66.3
12	65.775	92.1	0.31	65.775	92.1	62.4	9	0.27	65.875	92.1	59.7

Table 4.10: ARF 260 under TACMSS

L	ILP Sol			Iterative LP Sol					After Power Saving		
	avg	peak	time	avg	peak	area	#itr	time	avg	peak	area
11	397.78	602.7	0.07	427.45	593	141.6	2	0.04	427.51	593	142.9
12	351.03	602.7	0.06	337.43	694.8	220.4	10	0.2	337.51	633.4	178.3
13	261.26	572	0.65	311.48	694.8	207.2	7	0.35	311.62	633.4	194.7
14	242.6	572	1.4	242.6	572	286	8	0.4	242.69	572	275.4
15	204.67	316.7	2.03	226.43	572	272.8	6	0.36	193.83	572	259
16	212.31	291.4	14.5	171.48	572	286	8	0.72	171.55	572	242.6
19	110.04	133.3	28.6	110.04	143.8	92	6	0.42	110.11	143.8	81.4
22	95.036	122.8	246	95.036	122.8	92	6	0.72	95.091	122.8	84

Table 4.11: ARF 170 under TACMSS

L	ILP Sol			Iterative LP Sol					After Power Saving		
	avg	peak	time	avg	peak	area	#itr	time	avg	peak	area
11	397.78	602.7	0.03	427.45	593	141.6	2	0.04	427.51	593	142.9
12	391.83	572	0.32	391.88	593	144.2	5	0.15	391.93	582.8	144.2
13	336.58	572	3.33	311.48	694.8	207.2	7	0.42	311.62	633.4	177
14	289.27	572	10.3	242.6	694.8	207.2	6	0.36	242.69	602.7	242.6
15	204.67	316.7	4.87	269.95	694.8	207.2	6	0.42	204.71	633.4	209.8
16	191.88	316.7	25.9	212.28	593	220.4	8	0.72	191.95	572	226.2
19	110.04	133.3	21.8	110.04	143.8	92	6	0.48	110.11	143.8	81.4
22	95.036	122.8	263	95.036	122.8	92	7	0.84	95.091	122.8	84

Table 4.12: EWF 120 under TACMSS

L	ILP Sol			Iterative LP Sol					After Power Saving		
	avg	peak	time	avg	peak	area	#itr	time	avg	peak	area
17	150.65	439.5	0.02	150.65	439.5	122.7	7	0.14	150.74	434.4	125.3
18	142.28	286	0.06	142.28	439.5	122.7	7	0.14	124.24	347.4	128.6
20	111.73	173.7	2.29	103.57	214.9	84.7	16	0.96	103.67	209.8	88.6
21	90.867	153.5	3.12	90.867	286	156.2	11	0.88	75.41	286	153.5
28	44.836	61.4	378	44.836	92.1	82.2	23	4.83	44.964	66.8	57.8

Table 4.13: EWF 100 under TACMSS

L	ILP Sol			Iterative LP Sol					After Power Saving		
	avg	peak	time	avg	peak	area	#itr	time	avg	peak	area
18	142.28	286	0.05	142.28	429	122.7	7	0.21	124.24	347.4	128.6
20	111.73	173.7	3.22	103.57	214.9	91.3	16	1.44	103.67	209.8	88.6
21	90.867	153.5	10	83.095	245.6	101.1	12	1.32	75.395	173.7	120.1
28	44.836	61.4	23.5	44.836	92.1	82.2	25	5	44.964	66.8	57.8

Table 4.14: FDCT 315 under TACMSS

L	ILP Sol			Iterative LP Sol					After Power Saving		
	avg	peak	time	avg	peak	area	#itr	time	avg	peak	area
8	620.16	1174.7	0.03	620.16	1174.7	312.8	6	0.12	620.24	1174.7	314.1
9	551.26	1144	2.34	569.59	1287	320.5	10	0.6	569.79	1287	344.2
10	300.29	368.4	0.36	300.29	399.1	323.6	8	0.48	300.53	399.1	323.5
12	195.84	276.3	0.94	195.84	276.3	200.4	9	0.81	195.97	276.3	192.4
14	179.52	245.6	32.3	167.86	276.3	193.8	11	1.21	168.12	276.3	195
15	167.55	214.9	311	156.67	307	216.8	12	1.2	156.89	245.6	176

Table 4.15: FDCT 280 under TACMSS

L	ILP Sol			Iterative LP Sol					After Power Saving		
	avg	peak	time	avg	peak	area	#itr	time	avg	peak	area
9	569.49	1144	0.20	569.46	1308	317.9	12	0.84	551.76	1190.9	306.1
10	316.61	420.1	15.5	332.93	675.4	332	7	0.7	267.86	429.8	338.6
12	195.84	276.3	0.88	195.84	276.3	193.8	8	0.72	195.97	276.3	192.4
14	179.52	245.6	26.4	167.86	276.3	200.4	12	1.32	168.12	276.3	195
15	167.55	214.9	306	156.67	307	203.6	11	1.32	156.89	245.6	176

4.6 Chapter Summary

In this chapter, we have addressed the problem of power optimization during a combined module-selection and scheduling process. We have presented a unified MILP formulation for the two interwoven tasks in HLS, the module-selection and scheduling tasks, that targets peak power as well as average power optimization. We considered the problem of module-selection and scheduling under two sets of constraints, time constraint alone (TCMSS), and time and area constraints (TACMSS). First, we have presented an MILP formulation for finding an optimal solution for peak and average power consumption. Then, we have devised a two-phase heuristic to solve the same problem. In the first phase of our heuristic, the integrality in the MILP formulation is relaxed and a constructive LP solution is developed to get a near-optimal solution in a reasonable amount of time. In the second phase, we presented a power-area saving procedure to exploit any available room in the resultant LP solution to achieve further saving in peak and average power and at the same time restructure the resultant LP solution to satisfy the area constraint if it has been violated during the construction of the LP solution. The extensive experiments on several benchmarks show that our developed heuristic achieves peak and average power minimization that well match the results obtained by our MILP optimal solution.

CHAPTER 5

SIMULTANEOUS PEAK AND AVERAGE POWER OPTIMIZATION IN SYNCHRONOUS SEQUENTIAL CIRCUITS USING RETIMING AND MULTIPLE SUPPLY-VOLTAGES

Retiming was first introduced by Leiserson and Saxe [LS86] as an optimization technique for synchronous sequential digital circuits. It redistributes the registers in the circuit in such a way as to achieve a certain objective while preserving its original functionality. Since retiming can be used to shorten the critical path delay and to increase the parallelization among the computation nodes, it can be used to increase the number of computational elements that are candidates for voltage scaling.

In this chapter, we address the problem of minimizing dynamic peak and average power consumptions in synchronous sequential circuits by first presenting a mathematical MILP formulation for the exact solution through simultaneously combining retiming and multiple voltages scheduling for power (peak and/ or average) consumptions optimization. And, because the formulation is expensive regarding the number of variables, an efficient two-step heuristic is presented. First, a power-oriented retiming is devised which is polynomial solvable, followed by a MILP formulation for the resultant retimed circuit with the objective of optimizing power (peak and/ or average) consumptions.

5.1 Synchronous Circuit Representation

A synchronous graph model, $G = (V, E, d, w)$, is used as in [LS86] to model the synchronous sequential digital circuit; this is also referred to as sequential data-flow graph, (SDFG). Here V is the set of computational nodes, E is the set of interconnection edges between these nodes, d is a nonnegative number attached to each node representing its execution delay, and w is the set of weights in which each edge (u, v) has a weight $w(u, v)$ that expresses the number of registers

associated with that edge. Note that in the case of multiple voltage levels, the node-delays used in the retiming process are the node-delays at the highest voltage level.

5.2 Basics of Retiming

Retiming as introduced by Leiserson and Saxe [LS86] is an efficient optimization technique for the synchronous sequential digital circuits in which it restructures the digital circuit in order to achieve a certain objective while preserving its original functionality. A retiming r of a node u in the synchronous sequential data-flow graph, SDFG, is defined as the number of registers that is drawn from each outgoing edge of node u and pushed back to each incoming edge of that node u . So, retiming r can be positive, negative or zero. Mathematically, $r: V \rightarrow Z$, and it can be made positive by adding a large enough constant to the retiming r of all nodes. A retimed graph $G_r = (V, E, d, w_r)$, is generated as a result of retiming an SDFG, $G = (V, E, d, w)$, by a retiming r where

$$w_r(u,v) = w(u,v) + r(v) - r(u) \quad \forall (u,v) \in E \quad (5.1)$$

Since w_r is the register count in an edge, retiming r is a valid retiming if w_r is greater than or equal to zero. A similar expression results for a path P that originates at node u and terminates at node v , $P(u,v)$:

$$w_r(P(u,v)) = w(P(u,v)) + r(v) - r(u) \quad \forall u, v \in V, \quad (5.2)$$

where $w(P(u,v))$ is the sum of the edges weights of all edges that belong to path P . The delay of a path P , $d(P(u,v))$ is defined as the sum of the delays of all nodes that constitute that path P . The critical path of a graph G is the longest zero-weight path, where the zero-weight path P is any path P such that $w(P) = 0$. The clock period of a graph G is the delay of the critical path. Figure 5.1 is an example of an SDFG assuming that nodes **1**, **2**, and **3** are multiplication elements (delay = two units), while nodes **4**, **5**, and **6** are addition elements (delay = one unit) whose critical path is $\langle \mathbf{1}, \mathbf{4}, \mathbf{5}, \mathbf{6} \rangle$ and clock period 5.

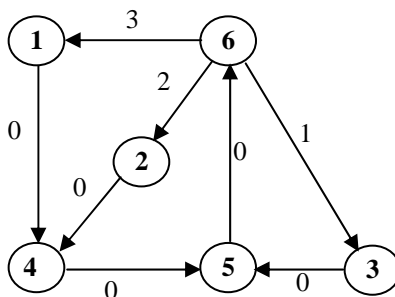


Figure 5.1: An example for SDFG.

Two matrices W and D that used in the retiming process are introduced in [LS86]; they are defined as follows:

$$W(u,v) = \min\{w(P) : P \text{ is a path from } u \text{ to } v.\}$$

$$D(u,v) = \max\{d(P) : P \text{ is a path from } u \text{ to } v \text{ and } w(p) = W(u,v).\}$$

5.3 Unifying Retiming and Power Optimization

Combining retiming with voltage scaling is a useful technique for power optimization since it can restructure the critical path in such a way as to move the power-expensive nodes out of the critical path and increase the number of nodes that are potential candidates for scheduling at lower supply-voltages.

While combining retiming and average power minimization can be done as in [CC03b, CW04], optimizing for peak power is harder to achieve by a simple formulation as in [CC03b, CW04]. This is because expressing the peak power consumption requires accounting for the contribution of the power consumption of a computation node towards power at every control step (or cstep) in which that computation node is active. In our MILP formulation, we introduce 0-1 variables to capture this activity information. Our MILP formulation is flexible in that peak power can be treated as an objective to be optimized for or can be used as a constraint for those applications with hard limits on peak power. The mathematical foundation for the MILP formulation is based on the following lemma and theorem.

Lemma 5.1:

Let $G = (V, E, d, w)$ be a synchronous sequential digital circuit, and let λ be a positive real number. There exists a legal retiming r of G such that the clock period of the retimed graph G_r is less than or equal to λ , if and only if there is an assignment of a real value $h(v)$ and an integer value $r(v)$ to each node $v \in V$ such that the following conditions are met:

1. $h(v) \geq 1 \quad \forall v \in V$
2. $h(v) + d(v) \leq \lambda + 1 \quad \forall v \in V$
3. $r(u) - r(v) \leq w(u,v) \quad \forall (u,v) \in E$
4. $h(u) - h(v) \leq -d(u) \quad \forall (u,v) \in E \text{ such that } r(u) - r(v) = w(u,v)$

Proof: It follows lemma 9 in [LS86] by using $h(v) = s(v) - d(v) + 1$.

Define x_{ijv} to be a 0-1 unknown variable that takes value 1 if node i starts execution at cstep j with voltage level v and 0 otherwise. Then, we can derive the following theorem.

Theorem 5.1:

Let $G = (V, E, d, w)$ be a synchronous sequential digital circuit, and let λ be a positive real number. Then, there exists a legal retiming r of G such that the clock period of the resultant retimed graph G_r is less than or equal to λ , if and only if there exist an assignment of a 0/1 value for each x_{ijv} variable, a real value $h(i)$ and an integer values $r(i)$ to each node $i \in V$ such that the following conditions are satisfied:

$$\begin{aligned}
T1. \quad & \sum_v \sum_j x_{ijv} = 1 && \forall i \in V \\
T2. \quad & \lambda R(i) - \lambda r(i) - \sum_v \sum_j j x_{ijv} = 0 && \forall i \in V \\
T3. \quad & -\lambda R(i) + \lambda r(i) \leq -1 && \forall i \in V \\
T4. \quad & \lambda R(i) - \lambda r(i) + \sum_v \sum_j x_{ijv} d(i, v) \leq \lambda + 1 && \forall i \in V \\
T5. \quad & r(i) - r(l) \leq w(i, l) && \forall (i, l) \in E \\
T6. \quad & \lambda R(i) - \lambda R(l) + \sum_v \sum_j x_{ijv} d(i, v) \leq \lambda w(i, l) && \forall (i, l) \in E
\end{aligned}$$

Proof: By using a similar transformation as in [LS86] in which for each node i , we substitute $h(i) = \lambda R(i) - \lambda r(i)$ in lemma 5.1 to mathematically formulate condition (4) in lemma 5.1; since the quantity $h(i)$ as well as the quantity $\sum_v \sum_j j x_{ijv}$ represent the start scheduling step, we can relate the two quantities by condition T2 above. Thus, the proof follows the proof of lemma 5.1 noting that the delay of a node i , $d(i) = \sum_v \sum_j x_{ijv} d(i, v)$.

Using the introduced 0-1 variables, the power consumption at any cstep j can be written as $P_j = \sum_v \sum_i x_{ijv} p(i, v)$. The peak power consumption therefore is $P_{peak} = \max_j (P_j)$. Hence, the peak power constraint can be written as:

$$\sum_v \sum_i x_{ijv} * p(i, v) \leq P_{peak} \quad \forall \text{cstep } j \quad (5.3)$$

We can use a weighted sum of the average and peak power consumption as the objective function:

$$\text{Minimize:} \quad \alpha P_{peak} + \beta \left(\frac{1}{\lambda}\right) \sum_i \sum_j \sum_v x_{ijv} * d(i, v) * p(i, v) \quad (5.4)$$

Equations (5.3) and (5.4) and the set of constraints in Theorem 5.1 work as the MILP formulation for the simultaneous retiming and (peak and/or average) power optimization. This MILP formulation is suitable only for small-size circuits because of the increased number of the 0-1 variables. Thus, we propose an efficient two-step heuristic based on the idea of power-oriented retiming, as it will be clear in the next section.

5.4 The Two-Stage Power Optimization Solution

5.4.1 Motivating Example

Recall that retiming was introduced by Leiserson and Saxe [LS86] with the objective of minimizing the clock period of the synchronous sequential digital circuit or to minimize the number of register needed. Chabini and Wolf [CW04] have proposed a way of retiming with the objective of maximizing the total number of non-zero-delay (NZD) edges in order to maximize the parallelization among the graph nodes. Their approach results in many nodes being scheduled with lower supply-voltages and power saving. But just maximizing the total number of NZD edges without taking the graph structure into consideration to control the choice of the edges does not exploit potential room for the graph nodes to be scheduled with lower supply-voltages. Consider the SDFG in Figure 5.1, assuming that the circuit can be operated at two supply-voltages, v_H and v_L (high voltage and low voltage) and using the module library in Table 2.3 with v_H as the 5v and v_L as the 3.3v. Retiming SDFG (note that the delays used in the retiming process are the node delays at the highest voltage level) in Figure 5.1 with the objective of maximizing the total number of non-zero-delay (NZD) edges under the time constraint of 4 results in the retimed graph G_{ru} shown in Figure 5.2-(a). The maximum power saving can be obtained by scheduling nodes $\{1, 4\}$ at v_L (shaded nodes) and $\{2, 3, 5, 6\}$ at v_H and that results in an average-power saving = $(84-13) + (26-6) = 91 \mu\text{watt}$. Optimizing the resultant retimed graph for average and peak power results in the schedule shown in Figure 5.2-(b) with 111.5 μwatt and 181 μwatt average and peak power respectively. The SDFG in Figure 5.1 can be retimed differently with another objective, which we call *power-oriented* retiming, under the same time constraint of 4 to get the power-retimed graph, G_{rp} , shown in Figure 5.2-(c).

The maximum power saving for Figure 5.2-(c) can be obtained by scheduling nodes $\{1, 2, 4\}$ at v_L (shaded nodes) and $\{3, 5, 6\}$ at v_H and that results in an average-power saving = $2*(84-13) + (26-6) = 162 \mu\text{watt}$. Optimizing the power-oriented retimed graph for average and peak power

will result in the optimal schedule shown in Figure 5.2-(d) with 82.5 μ watt average power and 116 μ watt peak power consumption, which is significantly lower than that for Figure 5.2-(b). This example shows how power-oriented retiming can direct the choice of zero-delay edges for the sake of power saving (average and peak) which is the goal in this work.

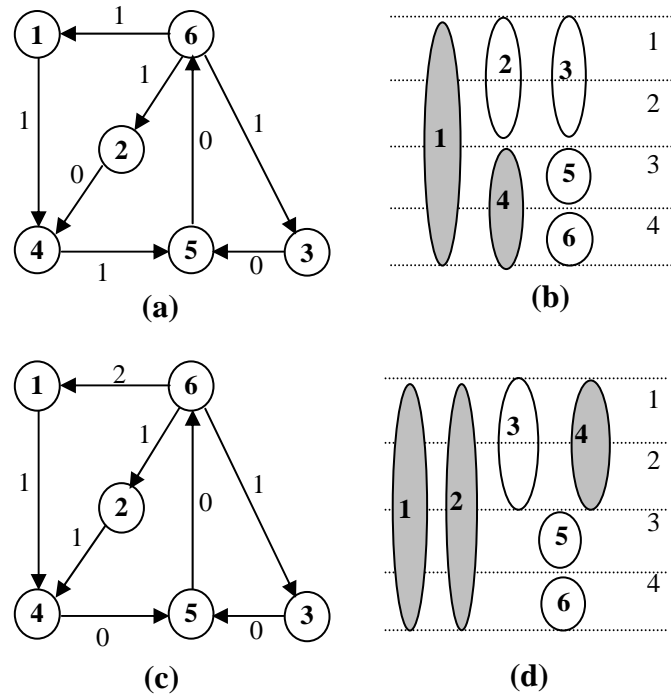


Figure 5.2: Results of retiming Figure 5.1 with two different retiming vectors for clock period = 4: (a) $r = (0\ 1\ 2\ 1\ 2\ 2)$, (b) power schedule of (a), (c) $r = (0\ 0\ 1\ 1\ 1\ 1)$, (d) power schedule of (c).

5.4.2 Power-Oriented Retiming

The main idea of power-oriented retiming is exploiting the circuit structure to impose control over edge selection in the retiming process in order to achieve power saving. This is achieved through associating with each edge (u,v) a cost function that captures the relative importance of edges according to the power consumption of the two end-nodes of this edge and achieve parallelism to favor the nodes with higher power consumption. The basic intuition here is that an edge with power-hungry nodes as end-points should not be on the critical path if at all possible. This allows power-expensive nodes to be scheduled at lower supply-voltages (which is reflected in average and peak power saving) and allows one to distribute the power-expensive nodes to different control steps obtain more peak power saving.

- $-f(u,v) + r(u) - r(v) \leq w(u,v) - 1 \quad \forall (u,v) \in E.$
- $r(u) - r(v) \leq W(u,v) - 1 \quad \forall u, v \in V \text{ such that } D(u,v) > \lambda.$

Proof: It is similar to the proof of Theorem 7 in [LS86].

The power-oriented cost function developed in Equation (5.5) when used in an objective function as shown in Equation (5.7) together with the set of inequalities in Theorem 5.2 work as the ILP formulation for the power-oriented retiming as follows:

$$\text{Minimize:} \quad \sum_{(u,v) \in E} f(u,v) * \text{cost}(u,v) \quad (5.7)$$

$$-f(u,v) \leq 0 \quad \forall (u,v) \in E \quad (5.8)$$

$$f(u,v) \leq 1 \quad \forall (u,v) \in E \quad (5.9)$$

$$-f(u,v) + r(u) - r(v) \leq w(u,v) - 1 \quad \forall (u,v) \in E. \quad (5.10)$$

$$r(u) - r(v) \leq W(u,v) - 1 \quad \forall u, v \in V \text{ such that } D(u,v) > \lambda. \quad (5.11)$$

It can be proved that the constraint matrix of the ILP formulation (5.7)-(5.11) for the power-oriented retiming is *totally unimodular* [NW88]. The reader is referred to [NW88] for more information about unimodular matrices. The practical impact of *total unimodularity* is that linear programming (LP) formulation without the integrality constraints produces the optimal solution, which can be obtained in a polynomial time.

As mentioned before, the second step in our solution is an MILP formulation for the DAG resulting from power-oriented retiming, which is discussed next.

5.4.3 Peak and Average Power Optimization for DAGs

The retimed graph is preprocessed to take off all the non-zero-delay edges. So, the resultant DFG $G(V, E)$ is a graph representation of a combinational circuit in which each vertex represents a computation node, and the edges represent the precedence relation among the vertices in the same iteration.

Problem 5.2: *Given a DFG representation of the design problem $G(V, E)$, the latency (greater than or equal to the minimum clock period of the retimed graph), a set of voltage levels for the operating resources, and a power/delay table that contains the average power consumption and the delay time needed for each resource operating on each voltage level. Find a schedule that minimizes the energy/power (average/peak) of the given DFG.*

Our solution for the multiple supply voltages scheduling (MVS) problem is a mixed integer linear programming (MILP) formulation targeting peak power consumption as well as average power and energy consumption for a given latency constraint same as the MILP formulation introduced in Section 2.2.1, Equations (2.1) through (2.5) and it is rewritten in Equations (5.12) through (5.16) for convenience. Moreover, we are trying to integrate a combination of these factors simultaneously. The solution to problem 2 is much less expensive than the solution of the *simultaneous retiming and (peak and/or average) power optimization* problem because in the resultant DFG, the structure of the graph is fixed and so a time-frame for each node can be obtained using the as-soon-as-possible (ASAP) and as-late-as-possible (ALAP) schedule (computed using the highest voltage level) as a preprocessing step for the MILP formulation. This helps greatly in reducing the time-frame (the range of steps from its ASAP to its ALAP) for each operation and so the number of 0-1 variables which it reflects in solution run-time.

$$\text{Minimize:} \quad \alpha P_{peak} + \beta(\frac{1}{\lambda}) \sum_i \sum_j \sum_v x_{ijv} d(i, v) \cdot p(i, v) \quad (5.12)$$

$$\sum_v \sum_j x_{ijv} = 1 \quad \forall i \in V \quad (5.13)$$

$$\sum_v \sum_j (j + d(i, v) - 1) x_{ijv} + \sum_v \sum_j j x_{l_jv} \leq -1 \quad \forall (i, l) \in E \quad (5.14)$$

$$\sum_i \sum_v \sum_{j_1=j-d(i,v)+1}^j x_{ijv} p(i, v) \leq P_{peak} \quad \forall j \in [1, \lambda] \quad (5.15)$$

$$\sum_v \sum_j (j + d(i, v) + 1) x_{ijv} \leq \lambda \quad \forall \text{node } i \text{ without successors} \quad (5.16)$$

5.5 Extensions

Power-oriented retiming strategy sometimes over-exploits circuit structure during retiming process, which may result in a retimed circuit that is not power-efficient. To achieve the best power minimal schedule, the circuit is optimized through two passes of the two-stage algorithm using two different cost functions in the objective of the mathematical formulation in each pass as shown in Figure 5.3. In one pass, the circuit is retimed using power-oriented retiming presented earlier; then the output retimed circuit is optimized for peak and/or average power consumption. In the other pass, the circuit is retimed using unit-weight cost in the objective function as presented in [CW04]; then the retimed circuit is also passed on to the same peak and/or average power optimization algorithm. The best power minimal solution is considered as the final schedule.

The circuit structure can also be exploited for area minimization through *area-oriented* retiming technique using a formulation similar to that used in power-oriented retiming process. Area-oriented retiming is based on the idea of maximizing resource utilization through forcing the graph nodes of the same type to be serialized (i.e., share a common path) rather than to be parallelized. This can be achieved by deriving the cost function associated with each edge in a way that gives more weight and priority to the edge that has its two end-nodes from the same type. Then, a scheduling algorithm can be used to obtain an area-minimal schedule that uses the retimed circuit as an input.

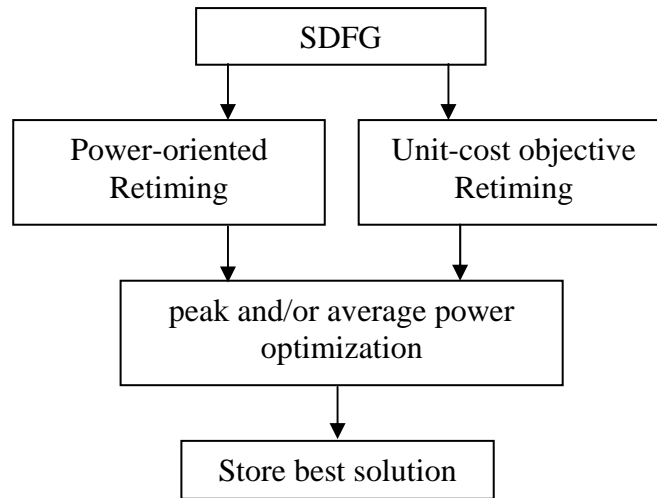


Figure 5.3: Procedure for extracting the minimal power schedule.

5.6 Experimental Results

Our MILP formulation for the exact solution as well as the two-stage heuristic (power-oriented retiming followed by MVS) for peak and average power minimization are tested on standard benchmarks such as the differential equation solver (HAL), and several DSP benchmarks such as the fifth-order elliptic wave filter (EWF), third-order direct-form filter (DFF) and lattice filter (LF) using the input module library introduced in Table 2.3. Although the solution of the combined retiming and peak power MILP formulation finds the optimal solution, it suffers from inordinately high solution times because of the large number of binary variables needed to model peak power. In order to overcome of this problem, the optimality can be sacrificed for the sake of short solution run-time. Thus, we set a time limit for the solver (CPLEX in our case) to terminate with a feasible integer solution if the optimal solution is not found or verified yet. Using this strategy, we first ran an experiment to compare our MILP

formulation of the combined retiming and peak power with a formulation that considers retiming only with average power consumption, like the work introduced by Chabini et al. [CC03b]. First, we calculate the peak power consumption for the resultant optimal solution of the unified retiming and average power from [CC03b] as tabulated in Table 5.1 under the heading “**Pavg objective**” for various benchmarks. Then, we set the average power obtained as a constraint in our MILP formulation and optimize for peak power consumption as shown in Table 5.1 under the heading “**Ppeak objective**”. Note that an “*” in Table 5.1 in the column labeled “time1” means that the solution is forced to terminate after one minute of time. For some benchmarks, the optimal solution is obtained within a fraction of second as in Table 5.1-(a) for DFF benchmark under time constraint of 6 and 10. Also, we ran another set of experiments using our MILP formulation for the combined retiming and power to optimize for peak power as well as average power consumption simultaneously; these results are tabulated under the heading “**Ppeak and Pavg**” in Table 5.1. The “*” beside the solution run-time in the “time2” column means that the solution is forced to terminate after this amount of time even though the optimality is not assured because the solution progress starts to freeze (i.e. it takes long time before it indicates further improvements that is measured using “gap” indicator with the CPLEX solver). The results in Table 5.1 show the usefulness of considering the peak power consumption in the formulation compared to a formulation that neglects it even with the cost of extra solution run-time.

The capability of our power-oriented retiming technique to restructure the sequential circuit to favor power minimization is compared to the unit-weight cost retiming (where the objective of the retiming process is just minimizing the number of non-zero-delay edges) by feeding both of the retimed DFGs to the same power optimization technique as depicted in Figure 5.3. The power optimization results for different benchmarks are tabulated in Table 5.2, using the same module library introduced in Chapter 2 in Table 2.3, for input timing constraints ranging from the critical path length to about twice the critical path length. As shown in Table 5.2, our power-oriented technique is more efficient in minimizing both peak and average power than the unit-weight retiming for most benchmarks (such as HAL, EWF, and DFF) under different time constraints. In some benchmarks such as LF in Table 5.2-(d), the resultant retimed DFGs from both retiming techniques are the same, while under few time constraints, the unit-weight retiming gives better results as in Table 5.2-(a) under a time-constraint of 8 csteps. This is

because power-oriented retiming strategy sometimes over-exploits circuit structure during retiming process. The solution of the MILP formulation for the combined retiming and peak and average power consumption (for the restricted solution run-time) is inserted in Table 5.2 under the title “**Exact Sol**” to show the quality of the two-phase heuristic solution for peak and average power using less run time. That is in most benchmarks the two-phase heuristic matches well the solution obtained from the combined MILP formulation.

Table 5.1: Peak and average power under varying objectives for different benchmarks

(a) DFF

L	Pavg Objective			Ppeak Objective			Ppeak and Pavg		
	Avg	Peak	time	Avg	Peak	time1	Avg	Peak	time2
4	130.5	161	0.01	130.5	155	*	130.5	155	0.03
5	102.2	161	0.02	102.2	155	*	104.4	149	0.4*
6	67.67	81	0.01	67.67	78	0.05	67.67	78	0.03
10	38.4	78	0.02	38.4	39	0.33	38.4	39	0.03

(b) LF

L	Pavg Objective			Ppeak Objective			Ppeak and Pavg		
	Avg	Peak	time	Avg	Peak	time1	Avg	Peak	time2
8	79	123	0.02	79	116	*	79	110	9.0*
10	51.6	110	0.01	51.6	84	1.27	51.6	84	0.92
12	33.33	42	0.02	33.33	36	*	33.33	36	2.0*
16	22.25	39	0.02	22.25	26	*	22.25	26	18.0*

(c) HAL

L	Pavg Objective			Ppeak Objective			Ppeak and Pavg		
	Avg	Peak	time	Avg	Peak	time1	Avg	Peak	time2
6	125.5	187	0.03	125.5	181	*	143	181	2.0*
7	104.43	187	0.06	104.43	123	*	106	123	4.0*
8	63.75	109	0.02	63.75	103	*	66.5	97	1.0*
12	31	52	0.01	31	32	*	31	39	20.0*

(d) EWF

L	Pavg Objective			Ppeak Objective			Ppeak and Pavg		
	Avg	Peak	time	Avg	Peak	time1	Avg	Peak	time2
16	111.375	265	0.09	111.375	204	*	111.375	204	34.0*
17	96.7059	174	0.04	96.7059	168	*	111.647	168	70.0*
20	59.55	168	0.04	59.55	97	*	65.9	107	100.0*
22	43.5909	75	0.05	43.5909	75	*	54.136	91	90.0*

Table 5.2: Power results of both power-oriented and unit-weight retiming

(a): HAL

L	Exact Sol		Unit-weight		Power-oriented	
	Avg	Peak	Avg	Peak	Avg	Peak
6	143	181	162.3	252	144.8	181
7	106	123	124.3	194	124.2	181
8	66.5	97	78.3	110	92.75	168
12	31	39	42.5	84	32.8	39

(b): EWF

L	Exact Sol		Unit-weight		Power-oriented	
	Avg	Peak	Avg	Peak	Avg	Peak
16	111.375	204	111.4	204	111.4	204
17	111.647	168	96.7	168	96.7	168
20	65.9	107	83.85	181	77.5	116
22	54.136	91	80.5	168	55.63	97

(c): DFF

L	Exact Sol		Unit-weight		Power-oriented	
	Avg	Peak	Avg	Peak	Avg	Peak
4	130.5	155	159.5	220	130.5	155
5	104.4	149	104.4	149	106.6	149
6	67.67	78	67.67	78	67.67	78
10	38.4	39	38.4	39	38.4	45

(d): LF

L	Exact Sol		Unit-weight		Power-oriented	
	Avg	Peak	Avg	Peak	Avg	Peak
8	79	110	79	110	79	110
10	51.6	84	51.6	84	51.6	84
12	33.33	36	33.3	36	33.3	36
16	22.25	26	22.3	36	22.3	36

5.7 Chapter Summary

We have presented two methods for dynamic average power as well as peak power consumptions in sequential synchronous circuits under time constraints using a combination of basic retiming and multiple voltage scheduling (MVS) techniques. Retiming is used to restructure the SDFG representation of the sequential circuit in order to increase the parallelism between operations and thus to increase the number of operations off the critical path to be

candidates for scheduling at lower supply voltages. First, we devised an MILP formulation for optimal peak and/or average power consumptions scheduling problem through a unification of retiming and MVS techniques. Mathematical formulation for peak power dictates piece of information for each operation in order to capture its activeness in a certain time step. Thus, we used binary variables for that purpose, which turns to be very large because tight time-frame for an operation cannot be obtained apriori. Then, to alleviate the problem of variable explosion, we presented a two-stage algorithm for peak and/or average power optimization. First, power-oriented retiming is proposed to restructure the input SDFG in order to achieve parallelization to the favor of nodes with high power consumption. Second, an MILP formulation is presented that takes the retimed DFG as input and produces an optimal peak and/or average power schedule using MVS technique. Our proposed power-oriented retiming generates a graph structure candidate for better peak and average power saving than similar works that depends on only maximizing the parallelism among graph nodes as shown by the experimental results for many benchmarks.

CHAPTER 6

DYNAMIC MEMORY USAGE OPTIMIZATION

This chapter addresses the problem of memory usage optimization in the evaluation of data-dependent program regions involving large data objects. We are interested in situations where the data objects are so large to fit in memory that they have to be dynamically allocated and deallocated during expression evaluation. This problem arises in the scientific computing field such as electronic structure calculations, and in several other contexts. The given computation is represented as a data-flow graph (DFG) $G(V, E, M)$ whose nodes V represent large data objects of different sizes M (an expression tree is a special case of the problem we are addressing). Edges E in the DFG represent dependencies between these data objects where the evaluation of a data object cannot start until all its children are evaluated. Each data object has to be allocated a certain amount of memory before it can start to be evaluated and it needs to be kept in memory until all its parents are evaluated completely. It is also assumed that each data object is an integral entity that has to be allocated or deallocated as a whole in memory. Reserving memory space for all data objects at the same time requires huge amount of memory so that in most cases the available memory is inadequate. The dynamic memory allocation model (in which a data object is allocated memory when it is needed and lasting until all its parents are evaluated and then it is deallocated) is considered in solving this problem.

There are many different possibilities for the evaluation order of the DFG nodes (the large data objects) varying widely in the maximum memory usage. The problem is to find an evaluation order for these data objects to achieve the least memory usage. Two variations of the data objects evaluation order problem in the case of multiple-processors using shared memory are also considered. These problems are the performance-constrained evaluation (PCE) and memory-constrained evaluation (MCE) given below. Performance-constrained evaluation addresses the problem of finding an evaluation order for the input DFG nodes that achieves the least amount of memory space required to do the computations without violating the input total

execution time (assuming that the execution time required by each node is given). While the memory-constrained evaluation problem dealing with the reverse scenario; where the evaluation order is sought to achieve the minimum total execution time for the DFG under the given memory constraint.

We are proposing two solutions for the memory usage optimization problem. First, we present a mixed integer linear programming (MILP) formulation for the two problems stated above to obtain the exact solution. Then, we present a force-directed scheduling heuristic for memory optimization that is based on the one introduced by Paulin and Knight [PK89]. For a mixed integer linear programming formulation, we introduce a 0-1 (unknown) variable, x_{ij} , that takes a value 1 if node i starts to be evaluated at time-step j , and 0 otherwise.

6.1 Preliminaries

In case the graph representation of the problem is a DAG, transitive reduction is applied. For example if there are edges (a,b) , (b,c) and (a,c) , then edges (a,b) and (b,c) do not contribute to the memory usage because they are dominated by the edge (a,c) which defines the life-time for node a (times at which data object a is still occupying a space in memory). The edge (a,c) is called the live-range edge in this case. After all the edges dominated by live-range edges are eliminated, a dummy node, v_d is introduced if there is some node v with more than one outgoing edge. Edges are classified according to their usage in the formulation and they are given a type from the set $\{0, 1, 2, 3\}$. All the edges are initially classified as type-0 edges. For each node v with more than one outgoing edge, we introduce a live-range edge between node v and its dummy node v_d . Such edges (v, v_d) are classified as type-3 edges. For each node w , a successor of node v , edge (v, w) is called a *reflexive edge* and it is marked type-1. Then for each reflexive edge (v, w) , we introduce a dummy edge (w, v_d) and mark this edge as a type-2 edge. In Figure 6.1-(a), node G has two outgoing edges (G,E) and (G,H) . Thus, a dummy node G_d is added as well as two dummy edges (E,G_d) and (H,G_d) for the reflexive edges (G,E) and (G,H) , respectively, as shown in Figure 6.1-(b).

Based on the definitions of edge sets given above, edges of type-3 generate redundant inequalities if they are considered in the precedence; and edges of type-1 and type-2 do not contribute in the memory usage because their role is taken by type-3 edges. Thus, type-3 edges are not to be considered in the precedence constraints while type-1 and type-2 edges are not to be considered in the memory constraints.

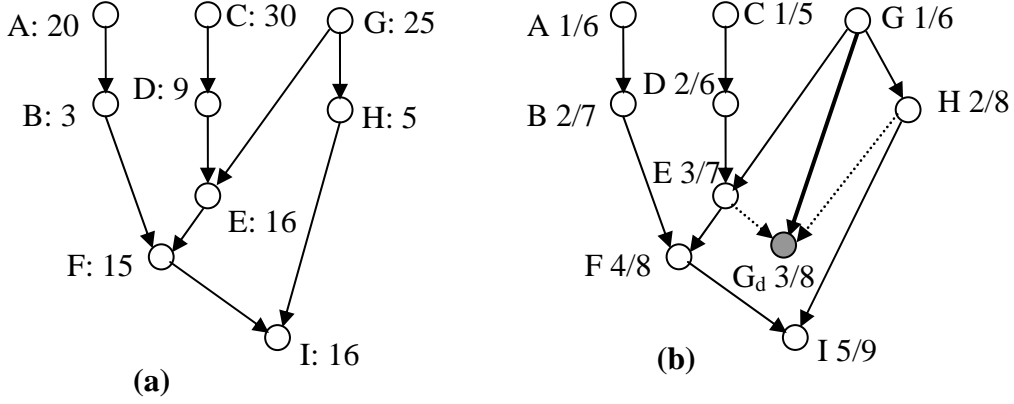


Figure 6.1: Sample DFG for memory evaluation:
(a) original DFG, **(b)** the preprocessed DFG.

Equation (6.1) below is developed to precisely define the memory usage at time-step j , mem_j . It considers the contribution of node i in memory usage during the set of time-steps starting directly after it finishes its processing until its last parent l is processed completely, which is captured by the first term inside the summation over the DFG edges E . In addition, the memory contribution of a node j to the memory usage while it is being processed is considered by the second term inside the summation over the set V of the DFG nodes:

$$mem_j = \sum_{(i,l) \in E} \left\{ M(i) \left(\sum_{k=1}^{j-D(i)} x_{ik} - \sum_{k=1}^{j-D(l)} x_{lk} \right) \right\} + \sum_{i \in V} \left(M(i) \sum_{k=j-D(i)+1}^j x_{ik} \right) \quad (6.1)$$

6.2 Evaluation Order and Performance-Constrained Evaluation (EOPCE)

Given a DFG representation of the evaluation order problem $G(V, E, M)$, the execution time needed for each node D , and the total time allowed to finish the computations (in number of time units) λ , the goal is to find an evaluation order for these data objects that achieves the least memory usage. The formulation is as follows.

▪ Objective Function

The objective is to minimize memory usage required at any time during the DFG evaluation process. This amount of memory space is expressed as the maximum memory usage at any time-step. Thus, the objective function is expressed as:

$$\text{Minimize:} \quad mem, \quad (6.2)$$

where mem will be used as a variable to be evaluated.

- **Uniqueness Constraints**

Each node should start at exactly one time-step within its time-frame. Inequality (6.3) is used to model this constraint.

$$\sum_j x_{ij} = 1 \quad \forall i, j \in R(i) \quad (6.3)$$

- **Precedence Constraints**

The fact that a data object can start to be evaluated only after all its data inputs (children) are ready is modeled as a precedence relation, one for each child-parent pair as shown in inequality (6.4), where each summation term in the left-hand side expresses the time-step at which a data object starts execution. Note that, in the case where the evaluation order is the objective, the delay, $D(i)$, is treated as one.

$$\sum_{j \in R(i)} jx_{ij} - \sum_{j \in R(l)} jx_{lj} \leq -D(i) \quad \forall (i, l) \in E \quad (6.4)$$

- **Memory Constraints**

The maximum memory space required during computation process is modeled by using a single variable that constrains total memory usage of all active nodes at each time-step as shown in inequality (6.5) and then minimizing that variable as the objective function.

$$\sum_{(i,l) \in E} \left\{ M(i) \left(\sum_{k=1}^{j-D(i)} x_{ik} - \sum_{k=1}^{j-D(l)} x_{lk} \right) \right\} + \sum_{i \in V} \left(M(i) \sum_{k=j-D(i)+1}^j x_{ik} \right) - mem \leq 0. \quad \forall j \in [1, \lambda] \quad (6.5)$$

In case the objective is the evaluation order without any constraint on the time needed, the execution time for each node is treated as one. In addition, another constraint is needed to force that at each time-step exactly one node is evaluated. This can be modeled as Equation (6.6).

$$\sum_{i \in V} x_{ij} = 1 \quad \forall j \in [1, \lambda] \quad (6.6)$$

6.3 Memory-Constrained Evaluation (MCE)

Given a DFG representation of the evaluation order problem $G(V, E, M)$, the execution time needed for each node D , and the maximum memory space allowed, $mem_constraint$, the goal is to find an evaluation order for these data objects that minimizes the total execution time needed.

The mathematical formulation is similar to the EOPCE problem described above. Uniqueness and precedence constraints are as described in inequalities (6.3) and (6.4), while the memory

constraint is different from the one in inequality (6.5) in which the maximum memory is explicitly posed as a constraint by the input *mem_constraint* value rather than by a variable as shown in inequality (6.7). Note that λ_{ub} used in inequality (6.7) is an upper-bound estimation of the total execution time. The total execution time, λ , is used twice as a variable, one to pose a constraint over the start time-step of the last node in the DFG (node without successors) as shown in inequality (6.8) and the other as the objective function to be minimized as shown in Equation (6.9).

$$\sum_{(i,j) \in E} \left\{ M(i) \left(\sum_{k=1}^{j-D(i)} x_{ik} - \sum_{k=1}^{j-D(i)} x_{jk} \right) \right\} + \sum_{i \in V} \left(M(i) \sum_{k=j-D(i)+1}^j x_{ik} \right) \leq mem_constraint. \quad \forall j \in [1, \lambda_{ub}] \quad (6.7)$$

▪ **Time Constraints**

$$\sum_j (j + D(i) - 1) x_{ij} \leq \lambda. \quad \forall \text{node } i \text{ without sucesors} \quad (6.8)$$

▪ **Objective function**

$$\text{Minimize: } Total \text{ execution time, } \lambda \quad (6.9)$$

6.4 Example for PCE Problem

Consider the DFG in Figure 6.1 assuming that each node has unit execution time and the computation has to be finished in five time-steps. Memory sizes as well as the ASAP and ALAP and the corresponding 0-1 variables of each node are as shown in Table 6.1. The complete listing of all constraints is shown in Figure 6.2. The set of constraints c1:c10 are the uniqueness constraints in Equation (6.3), the set of constraints c11:c21 are the precedence constraints as a result of applying inequality (6.4), and constraints c22:c26 are the maximum memory constraints, where maximum memory is treated as a variable *mem* that constitutes the objective function to be minimized. The output from the MILP solver “CPLEX” [CPX02] is as shown in Table 6.2.

Table 6.1: Memory sizes, ASAP and ALAP for the DFG in Figure 6.1

Node	A	B	C	D	E	F	G	H	I	G _d
Cost	20	3	30	9	16	15	25	5	16	0
ASAP	1	2	1	2	3	4	1	2	5	3
ALAP	2	3	1	2	3	4	2	4	5	4
associated variables	x ₁ , x ₂	x ₃ , x ₄	x ₅	x ₆	x ₇	x ₈	x ₉ , x ₁₀	x ₁₁ , x ₁₂ , x ₁₃	x ₁₄	x ₁₅ , x ₁₆

```

Minimize
  obj: mem
Subject To
  c1:  x1 + x2 = 1
  c2:  x3 + x4 = 1
  c3:  x5 = 1
  c4:  x6 = 1
  c5:  x7 = 1
  c6:  x8 = 1
  c7:  x9 + x10 = 1
  c8:  x11 + x12 + x13 = 1
  c9:  x14 = 1
  c10: x15 + x16 = 1

  c11: x1 + 2 x2 - 2 x3 - 3 x4 <= -1
  c12: x5 - 2 x6 <= -1
  c13: - 3 x7 + x9 + 2 x10 <= -1
  c14: x9 + 2 x10 - 2 x11 - 3 x12 - 4 x13 <= -1
  c15: 2 x3 + 3 x4 - 4 x8 <= -1
  c16: 2 x6 - 3 x7 <= -1
  c17: 2 x11 + 3 x12 + 4 x13 - 5 x14 <= -1
  c18: 3 x7 - 4 x8 <= -1
  c19: 4 x8 - 5 x14 <= -1
  c20: 3 x7 - 3 x15 - 4 x16 <= 0
  c21: 2 x11 + 3 x12 + 4 x13 - 3 x15 - 4 x16 <= 0

  c22: 20 x1 + 30 x5 + 25 x9 - mem <= 0
  c23: 20 x1 + 20 x2 + 3 x3 + 30 x5 + 9 x6 + 25 x9 + 25 x10 + 5 x11 - mem <= 0
  c24: 20 x1 + 20 x2 - 17 x3 + 3 x4 + 30 x5 - 21 x6 + 16 x7 + 25 x9 + 25 x10
      + 5 x11 + 5 x12 - mem <= 0
  c25: 20 x1 + 20 x2 - 17 x3 - 17 x4 + 30 x5 - 21 x6 + 7 x7 + 15 x8 + 25 x9
      + 25 x10 + 5 x11 + 5 x12 + 5 x13 - 25 x15 - mem <= 0
  c26: 20 x1 + 20 x2 - 17 x3 - 17 x4 + 30 x5 - 21 x6 + 7 x7 - 4 x8 + 25 x9
      + 25 x10 + 5 x11 + 5 x12 + 5 x13 + 16 x14 - 25 x15 - 25 x16 - mem <= 0

```

Figure 6.2: Complete listing of all constraints of the example at Table 6.1.

Table 6.2: Output memory schedule for the ILP formulation in Figure 6.2

time step	1	2	3	4	5
scheduled Nodes	C	A, D, G	B, E	F, H, G _d	I
memory usage	30	84	73	64	36

We have also developed a heuristic to find an evaluation order for the input DFG that uses the least amount of memory by distributing the memory usage evenly over the allowed computation time. Note that in the case of just finding an evaluation order without any time constraint, the number of time-steps is the number of nodes in the DFG. The algorithm is based on the one introduced by Paulin and Knight [PK89] but differs significantly in the forces types defined and in the distribution graph used. The heuristic presented addresses the Evaluation Order and Performance-Constrained Evaluation (EOPCE) problem stated earlier.

6.5 Memory Force-Directed Scheduling for Memory Minimization (MFDS)

The goal is to balance the memory usage through distributing the memory required evenly over the total number of time-steps (time constraint) needed for the computation. Applying the original form of the force-directed scheduling introduced by Paulin and Knight [PK89] will result in approximately even distribution of the nodes over the entire time but not for the memory usage. This is because each node requires different amounts of memory and still occupies (i.e., requires) this amount of memory till all of its successors are scheduled (assigned certain time-step from the set $\{1, 2, \dots, \lambda\}$). Thus we have developed a scheduling algorithm based on the one introduced by Paulin and Knight [PK89] that is different in the force-types and in the distribution graph used.

A memory distribution graph is constructed to represent the probabilistic memory usage at each time-step. At any time step j , the *memory distribution graph*, mDG , is given by:

$$mDG(j) = \sum_{i \in V} prob(i, j) * M(i) \quad (6.10)$$

where $prob(i, j)$ is the probability that node i is alive at time-step j in which it contributes to the memory usage at that time-step j , i.e., it captures the probability that a node contributes to the memory usage in each possible combination of relative time-steps between a node and its last successor. For example, consider Figure 6.3 assuming that ASAP and ALAP values are as shown beside each node of the edge (v, w) . Time-steps at which node v and node w can be located are $(3, 4)$, $(3, 5)$, $(3, 6)$, $(4, 5)$, $(4, 6)$, and $(5, 6)$, respectively. This results in the probability of node v being $3/6$, $5/6$, $5/6$, and $3/6$ at time-steps 3, 4, 5, and 6, respectively, and zero elsewhere. Figure 6.4 is a simple procedure to compute such probability, where $lambda$ is the total time-steps, and $s1$ and $s2$ are the ASAP times for nodes u and v of edge (u,v) , while $t1$ and $t2$ are their ALAP times, respectively.

Memory-forces are developed in such a way as to distribute the DFG nodes so that their contributions to memory usage at each time-step are as small as possible. The memory self-force, *mselfForce*, reduces the contribution of that node to the memory usage from that time-step until the time-step at which its last successor is scheduled. Similarly, the memory predecessor-force, *mpreforce*, reduces the contribution of the node-children to the memory usage from the time-step at which these children are scheduled until the time step being considered. Distributing the DFG

nodes so that their contributions to the memory usage at each time-step are as small as possible is achieved by scheduling the node in a time-step to balance the memory self-force (the force that tries to pull it close to its successors) with the memory predecessor-force (the force that tries to pull it back to be close to its predecessors) as given by Equation (6.11) and (6.12). This results in a schedule in which nodes are not distributed over all the allowed time-steps (i.e., the time constraint); rather they are become close to each other and clustered in a portion of the allowed time-steps equal to the length of the critical path increasing the maximum memory requirement.

$$mselfForce(i, j) = M(i) \left(\sum_{j_1} mDG(j_1) * newpr(i, j_1) - \sum_{j_2} mDG(j_2) * prob(i, j_2) \right) \quad (6.11)$$

$$mpreForce(i, j) = \sum_{k \in children(i)} M(k) \left(\sum_{j_3} mDG(j_3) * newpr(k, j_3) - \sum_{j_4} mDG(j_4) * prob(k, j_4) \right) \quad (6.12)$$

$$j_1 \in newtime\ frame(i), \quad j_2 \in initial\ time\ frame(i)$$

$$j_3 \in newtime\ frame(k), \quad j_4 \in initial\ time\ frame(k)$$

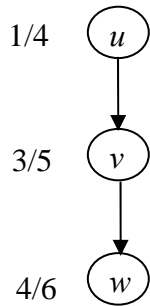


Figure 6.3: Portion of a sample DFG.

function [pr] = prob(lambda,s1,t1,s2,t2)

1. *pr = zeros(1, lambda);*
 2. *pCnt = 0;*
 3. *for j=s1:t1*
 - 3.1. *for j2=max(j+1, s2):t2*
 - 3.1.1. *pCnt = pCnt + 1;*
 - 3.1.2. *for s=j:j2*
 - 3.1.2.1. *pr(s) = pr(s) + 1;*
 4. *for j=s1:t2*
 - 4.1. *pr(j) = pr(j) / pCnt;*
- return*

Figure 6.4: Procedure for probability computation.

To overcome the inefficiency caused by applying memory-forces and to force the nodes to be distributed over the entire time-steps in such a way as to achieve minimum memory requirement, another type of forces, *node-forces*, similar to the forces introduced by Paulin and Knight [PK89] are added to the memory-forces. Note that our node-force is different from that introduced in [PK89] in which the effect of multi-time-step nodes is considered as shown in Equations (6.14) and (6.15). Node-forces affect the node distribution so that the memory requirements in specific time-steps in which these nodes are scheduled (not where the nodes are still alive) are minimal.

Thus node-forces require a different distribution graph reflecting the memory size required by a node i itself and by all its children, $ex_M(i)$. This distribution graph, DG , is formed as in Equation (6.13)

$$DG(j) = \sum_{ASAP(i) \leq j \leq ALAP(i)} \frac{1}{mobility(i)} * ex_M(i) \quad (6.13)$$

$$selfForce(i, j) = ex_M(i) \left(\sum_{j_1} DG(j_1) - (1/mobility(i)) \sum_{j_2} DG(j_2) \right) \quad (6.14)$$

$$psForce(i, j) = ex_M(i) \sum_{k \in ps(i)} \left(\begin{array}{l} (1/newmobility(k)) \sum_{j_3} DG(j_3) \\ - (1/mobility(k)) \sum_{j_4} DG(j_4) \end{array} \right) \quad (6.15)$$

where $j_1 \in [j-D(i)+1, j]$, $j_2 \in [ASAP(i), ALAP(i)+D(i)-1]$, $j_3 \in [ASAP(k), ALAP(k)+D(k)-1]$, and $j_4 \in [newASAP(k), newALAP(k)+D(k)-1]$; and $newASAP$, $newALAP$, and $newmobility$ are the ASAP, ALAP, and mobility of predecessors/successors of operation i , respectively, due to the change in their time-frames.

Total forces are now formed as a weighted sum of the two types of forces: (i) memory-forces that minimize memory requirement of a node at time-steps in which it is alive; and (ii) node-forces that minimize memory requirement at the time-steps in which a node is scheduled. Weighting factors in the total forces can be tuned to give more preference to one type of forces over the other. The best results are obtained when the weighting factors are chosen to normalize the values of both types and enforcing them to have equal effect.

While node-force consists of three components (self-force, predecessor-force and successor-force), memory-force consists of only two components (mselfForce and mpreForce). This is because memory-forces resulting from assigning a node to a certain candidate time-step do not include a force from its successors, because node successors do not contribute to the change of memory requirement when a node attempt to be assigned in different candidate time-steps. For example, consider Figure 6.5 that shows two different instances of scheduling the DFG in Figure 6.3; let us assume that the memory sizes of nodes u , v , and w are 20, 3, and 12, respectively, and nodes u and w are already scheduled at time-steps 1 and 6, respectively. Figure 6.5-(a) shows the memory distribution when node v is assigned time-step 3; while Figure 6.5-(b) shows the memory distribution when it is assigned time-step 4.

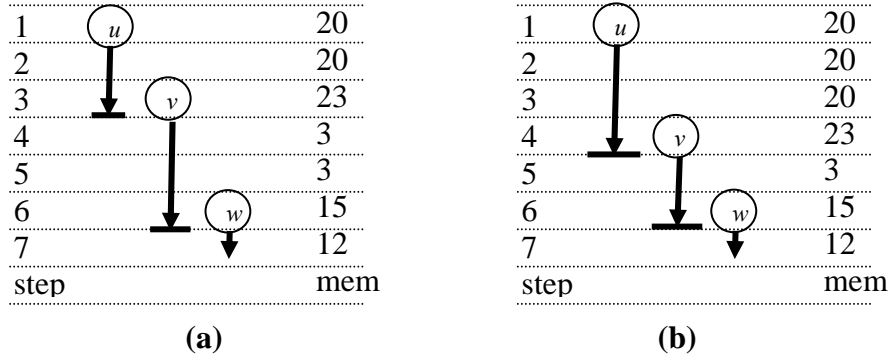


Figure 6.5: Sample scheduling for Figure 6.3: **(a)** node v is scheduled at time-step = 3, **(b)** node v is scheduled at time-step = 4.

6.6 Experimental Results

For design space exploration (DSE), we have adopted an efficient strategy in doing the experiments that can precisely identify the *pareto points* in the design space. A point in the design space that achieves the least memory usage for a given time constraint, while the time constraint is the minimum execution time for that memory usage is called a *pareto point*. This strategy is depicted in Figure 6.6. First, for a given time constraint L , we solve the PCE problem to find the least memory usage M . Then this memory M is used as a constraint in solving the MCE problem to find the minimum total execution time $L2$. If that execution time $L2$ is the same as the time constraint L , then this point (L, M) is a pareto point, otherwise $L2$ is fed back to the PCE as a constraint to find the least memory usage $M2$. Again, if this resultant memory $M2$ is the same as M , then $(L2, M)$ is a pareto point.

In DSE, the execution time taken by the ILP solver for each experiment as well as the total execution time needed to fully explore the design space depends on many factors. It depends on the DFG structure, how far the time constraint is from the critical path length, the number of nodes in the DFG, as well as the distribution of memory costs of the DFG nodes. The first two factors define the number of variables associated with each node while the number of DFG nodes contributes to the total number of variables. The memory cost distribution of DFG nodes as well as the DFG structure shape the search space in the ILP solver for optimal objective function and hence the solution time.

We have used four examples to test our presented ILP formulations as well as the proposed memory-force-directed heuristic (MFDS) for memory evaluation problem. These test examples are Exdag1 shown in Figure 6.1, Ex1 that is a tree structure of Exdag1 after the edge (G, E) is

removed, Ex2 that is the same as Ex1 but with different memory cost for each node, and Ex3 and Ex4 presented in Figure 6.7-(a) and 6.7-(b), respectively. Ex3 and Ex4 have the same number of nodes, memory costs, and execution time, but are different in the graph structure.

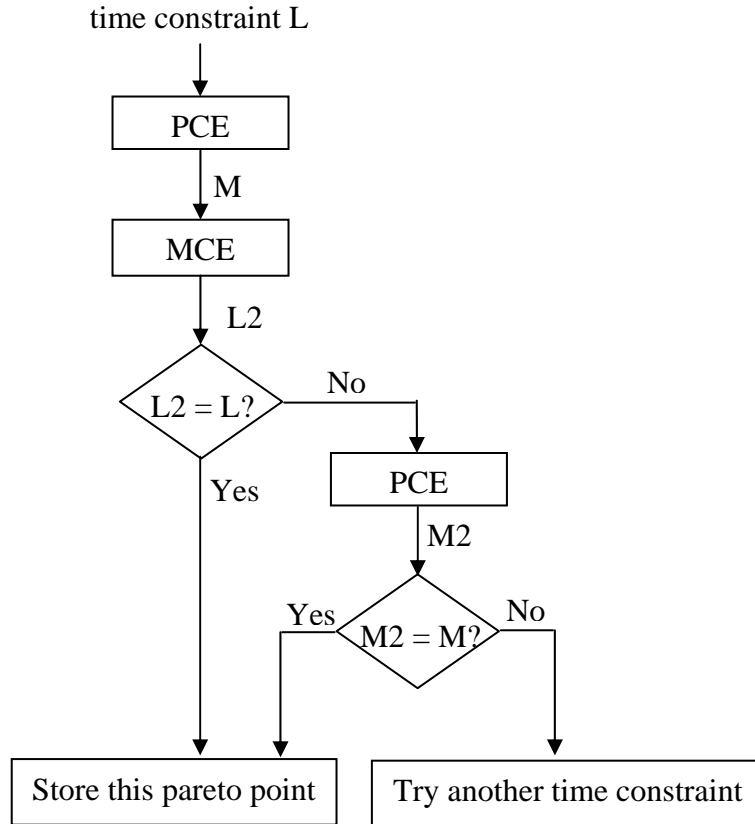
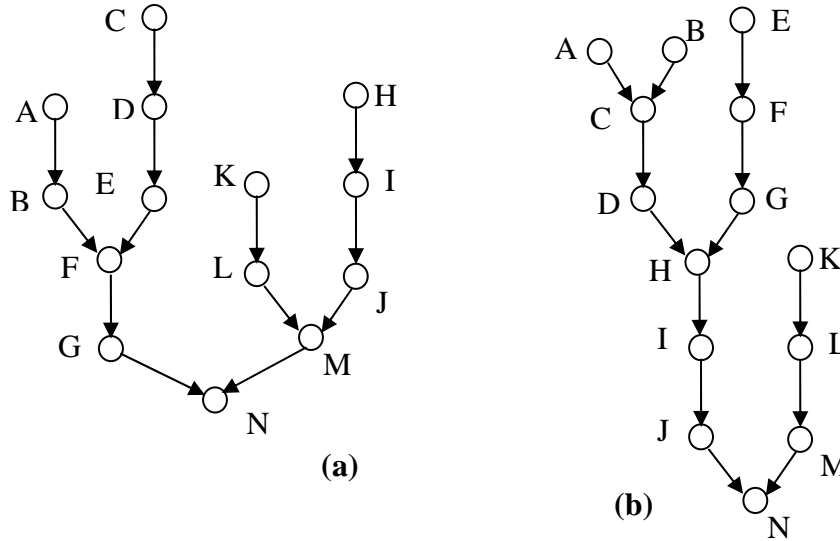


Figure 6.6: Flow chart for DSE strategy.

Results for memory evaluation under timing constraints when the computation elements (nodes) are assumed to have unit time delays are shown in Table 6.3, while the results for the general case in which the computation elements can take different times to be processed are shown in Table 6.4. The optimal ILP solution (maximum memory usage in case of PCE problem and the number of time units in case of the MCE problem) is tabulated under the heading “ILP” and the MFDS heuristic solution is tabulated under the heading “mfd”. In the result tables, “L” is the time constraint, “mem” is the memory constraint, “time” is the ILP solution time in seconds, and “error” is the percentage difference between the heuristic and the optimal ILP results. Experiments are done for different time-constraints ranging from the critical path length to twice the critical path length in the general case and for a time-constraint equaling the number of nodes in the case of a unit time delay. Optimal ILP results for memory-constrained evaluation

are presented in Table 6.5 for different values of memory constraints for each test example. Results show that the MFDS heuristic solution is very close to the optimal ILP solution and in many cases matches those optimal results. The ILP solution times are large only when the time constraint is far from the length of the critical path and when the memory constraints are very stringent.



node	A	B	C	D	E	F	G	H	I	J	K	L	M	N
M	2	20	12	9	11	7	9	5	12	16	11	13	5	8
d	3	2	1	1	2	1	1	2	1	2	2	1	2	1

(c)

Figure 6.7: Sample examples for memory evaluation: (a) Ex3, (b) Ex4, (c) memory size, M, and delay, d, for each node in (a) and (b).

Table 6.3: Performance constrained evaluation with unit delay

(a) Ex1

L	ILP	mfds	error	time
9	39	39	0	0.44
8	45	59	23.73	0.20
7	53	59	10.17	0.11
6	59	69	14.49	0.05
5	64	87	26.44	0.02

(d) Exdag1

L	ILP	mfds	error	time
9	50	64	21.88	0.38
8	53	58	8.62	0.22
7	58	59	1.69	0.11
6	59	62	4.84	0.04
5	84	92	8.70	0.01

(b) Ex2

L	ILP	mfds	error	time
9	44	48	8.33	0.38
8	47	57	17.54	0.21
7	54	64	15.63	0.17
6	59	59	0	0.05
5	75	75	0	0.01

(d) Ex3

L	ILP	mfds	error	time
14	43	50	14	16.2
10	46	53	13.21	1.03
8	54	58	6.9	0.19
7	61	66	7.58	0.01
6	79	79	0	0.02

(Table 6.3 Continued)

(d) Ex4

L	ILP	mfds	error	time
14	34	52	34.62	11.4
12	34	52	34.62	2.7
10	35	46	23.91	0.79
8	46	46	0	0.08
7	52	52	0	0.01

Table 6.4: Performance constrained evaluation

(a) Ex1

L	ILP	mfds	error	time
20	39	64	39.06	5.53
15	53	63	15.87	0.75
12	64	67	4.48	0.08
11	67	67	0	0.03
10	78	78	0	0.03

(b) Ex2

L	ILP	mfds	error	time
20	44	48	8.33	8.35
18	47	57	17.54	2.2
15	57	70	18.57	1.0
12	59	70	15.71	0.12
10	84	84	0	0.06

(c) Exdag1

L	ILP	mfds	error	time
20	50	64	21.88	28.5
15	58	67	13.43	0.81
12	67	67	0	0.08
11	78	78	0	0.08
10	78	78	0	0.03

(d) Ex3

L	ILP	mfds	error	time
16	43	61	29.51	7.18
14	47	61	22.95	1.58
12	50	67	25.37	0.46
10	67	67	0	0.09
8	83	83	0	0.02

(e) Ex4

L	ILP	mfds	error	time
22	34	65	47.69	28.5
16	35	65	46.15	4.0
14	41	57	28.07	0.62
12	46	50	8	0.13
11	46	52	11.54	0.03

Table 6.5: ILP memory constrained evaluation

(a) Ex1

mem	L	time
39	20	4.9
48	17	3.7
50	17	3.7
64	12	0.11
78	10	0.04

(b) Ex2

mem	L	time
44	20	5.0
47	17	2.3
57	15	1.56
60	12	0.13
84	10	0.03

(c) Exdag1

mem	L	time
50	20	9.5
53	17	2.8
67	12	0.26
70	12	0.12
78	10	0.04

(Table 6.5 Continued)

(d) Ex3

mem	L	time
43	16	4.9
47	14	2.5
50	12	0.88
67	10	0.22
83	8	0.03

(e) Ex4

mem	L	time
34	17	6.8
35	16	1.94
41	13	0.37
46	11	0.02

6.7 Chapter Summary

In this chapter, we addressed the problem of memory usage optimization in the evaluation of expression trees involving large data objects. We are interested in the situations where the data objects are so large to fit in memory that they have to be dynamically allocated and deallocated during expression evaluation. This problem arises in electronic structure calculations and in several other contexts. We considered three different variations of the memory usage optimization problem. The first problem is finding an evaluation order for the dependent data objects in order to achieve the least amount of required memory. The second problem is the generalization of the evaluation order problem in which the data objects have different processing times and the least amount of memory is sought for a given total execution time as a constraint. The third problem is the complementary of the second problem in which the total execution time is minimized for a given maximum amount of memory as a constraint. We have developed an ILP formulation to optimally solve the three problems. In addition we presented a memory-force-directed heuristic to solve for minimal memory usage in the second problem. Solution results show that the ILP solution time is small especially when the time constraint is close to the critical path length or when the memory constraint is less stringent and the heuristic results are very close to those of the ILP solution using less run time.

CHAPTER 7

LOOP FUSION USING ILP FOR MEMORY MINIMIZATION

This chapter presents a technique for memory optimization for a class of computations that arises in the field of correlated electronic structure methods such as coupled cluster and configuration interaction methods in quantum chemistry. In this class of computations, loop computations perform a multi-dimensional sum of product of input arrays. There are many different ways to achieve the same final results that differ in the number of arithmetic operations required. In addition, for a given number of arithmetic operations, different loop structures have different memory requirements. Loop fusion is a plausible solution for reducing memory usage. By fusing loops between producer loop nest and consumer loop nest, the required storage of intermediate array is reduced by the range of the fused loop. Because resultant loops have to be legal after fusion, some loops can not be fused at the same time.

7.1 Problem Definition and Formulation

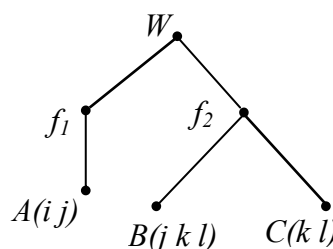
Figure 7.1 is an example of a multi-dimensional integral; Figure 7.1-(a) shows a multi-dimensional integral expressed as a sum of product of arrays. Figure 7.1-(b) shows the resultant operation-count-optimal formula sequence and Figure 7.1-(c) is its graph representation. This graph representation is the same as the one presented in [LS99a] except that the multiplication and summation nodes are combined together in one node.

$$W[k] = \sum_{i,j,l} A[i,j] \times B[j,k,l] \times C[k,l]$$

(a)

$$\begin{aligned} f_1[j] &= \sum_i A[i,j] \\ f_2[j,k] &= \sum_l B[j,k,l] \times C[k,l] \\ W[k] &= \sum_j f_1[j] \times f_2[j,k] \end{aligned}$$

(b)



(c)

Figure 7.1: An example of multi-dimensional integral: (a) a multi-dimensional integral, (b) a formula sequence for computing (a), (c) graph representation of (b).

7.1.1 Modified Fusion Graph

Another graph representation of the problem called the *fusion graph* $FG = (V, E, Indices, N)$ is extracted from the original problem graph, which is suitable for formulating the fusion problem at hand and it is a modified version of the one presented in [LS99a] in order to decrease the number of variables used. In the fusion graph FG,

- V is the sets of nodes where each set of nodes represents an array (intermediate, input, or output array) as described below,
- E is the set of potential fusion edges as described below,
- $Indices$ are the sets of loop indices associated with each node, and
- N is the set of loop ranges.

The fusion graph is constructed as follows:

1. Each node $v \in V$ in the original graph is converted to a set of vertices, one for each loop i , where i is a loop index of node v ($i \in Indices(v)$).
2. For each common loop index between a node and its parent, an edge $e \in E$ is introduced called a *potential fusion edge*; the common loop index in this case is said to be a candidate for fusion. If the common loop between a node and its parent is fused, the potential fusion edge is called a *fusion edge*.

Figure 7.2-(a) shows the potential fusion graph for the original graph in Figure 7.1-(c). The potential fusion edges are dotted edges and the fusion edges are shown as solid edges in the fusion graph. In the fusion graph, each connected component of fusion edges forms a fusion chain, which corresponds to a fused loop in the loop structure. In Figure 7.2-(b), there are three fusion chains, one for each of the j -, k -, and the l -loops. The set of nodes between and including nodes a and b , in which all the i -vertices of these nodes are connected through potential fusion edges is called the potential fusion scope of the i -loop between the two nodes, a and b , $pfscope(a, b, i)$. Similarly, the fusion scope of the i -loop between two nodes, a and b , $fscope(a, b, i)$, is defined as the set of nodes between and including a and b , in which all the i -vertices of these nodes are connected through fusion edges. Again, in Figure 7.2-(b), $fscope(B, W, j) = \langle B, f_2, W \rangle$.

In terms of the graph, fusing loop i between a node u and its parent v eliminates the i -dimension of the array u . In addition, if the total size of the array u is originally N and the range

of the loop i is N_i , then after fusion, the array size u becomes N/N_i . This is in effect, the product of the ranges of the loops of u that are not fused with its parent v . Only fusing common loops over nodes that form child-parent relations helps in reducing memory usage; for example, fusing the loops between children of the same parent without fusing the parent does not affect memory usage. For the class of loops that arise in our application domain of interest, dependences do not prevent loop fusion; the only requirement is that the child nodes are evaluated before parent nodes are.

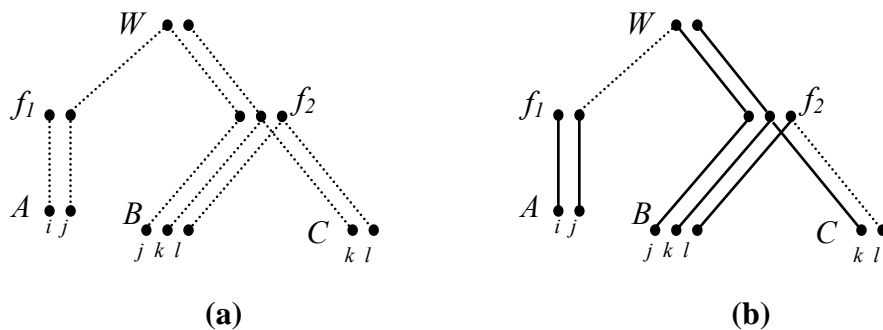


Figure 7.2: Fusion graph for operation-minimal sequence in Figure 7.1:
(a) potential fusion graph, **(b)** resulting fusion graph.

We have proposed a mathematical formulation for the optimal memory usage problem in which the objective is to minimize the total memory usage (static memory allocation model) for the given operation-count-optimal formula sequence. Our formulation is a novel integer linear programming (ILP) that is shown to be highly effective on a number of test cases producing the optimal solutions using very small execution times. The main idea in the ILP formulation is the encoding of legality rules for loop fusion of a special class of loops using logical constraints over binary decision variables and a highly effective approximation of memory usage. Constraints to assure legality for the resultant fusion graph are developed in a form of a set of linear inequalities. Because of the nature of the problem, the objective is formulated as a nonlinear function. Then, an efficient linearization technique has been developed that transforms the objective function to be linear and thus the memory usage problem is formulated as an integer linear programming (ILP) problem. Although the linearized objective function does not guarantee optimality, the solution is found to be matching the optimal one because the linearization is an effective approximation of the nonlinear objective function.

7.2 Legality of Fusion

The following theorem is a formal restatement of the fusion legality introduced in [Lam99]. The theorem states the basic definitions and the sufficient conditions for a fusion to be legal. Based on that theorem, fusion legality constraints are generated.

Theorem 7.1:

Let $FG = (V, E, Indices, N)$ be a fusion graph, and let a and b be any two nodes in FG . For any two loop-indices j and k , **fusion is legal if one of the following conditions is satisfied:**

1. $fscope(a, b, j) \cap fscope(a, b, k) = \emptyset$.
2. $fscope(a, b, j) \subseteq fscope(a, b, k)$.
3. $fscope(a, b, j) \supseteq fscope(a, b, k)$.

Proof: Since loops are not allowed to overlap (they must either be nested or separate), fusion is legal if the chains of any two loops in a fusion graph are not partially overlapped, i.e., they must be either disjoint or a subset/superset of each other, which can be mathematically rewritten as the conditions (1)-(3) above. ■

Figure 7.3 below shows different cases of illegal fusion and Figure 7.4 shows different configurations of legal fusion. To capture the legality of fusion in a set of linear inequalities, we introduce a 0-1 unknown variable, x_{ai} , to denote the potential fusion edge between node a and its parent. The unknown variable x_{ai} takes a value 1 if the i -loop is fused between node a and its parent, and 0 otherwise.

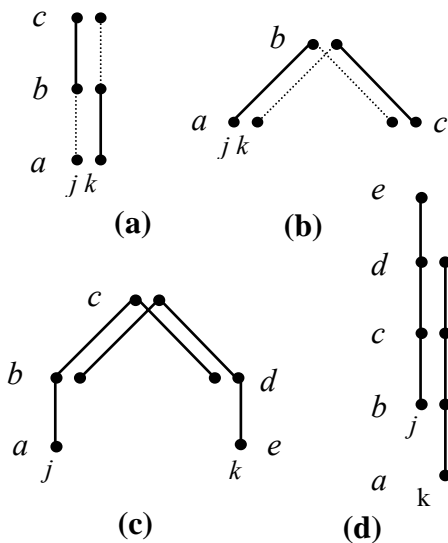


Figure 7.3: Illegal fusion configurations.

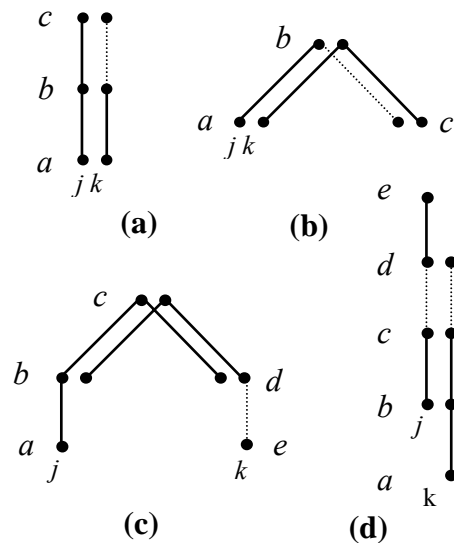


Figure 7.4: Legal fusion configurations.

Fusion legality described by Theorem 7.1 can be posed as constraints in a form of linear inequalities using inequality (7.1) shown below. That simply says: for each path $P(s,t)$ that starts at node s and ends at node t , and for any two loop indices j and k in the fusion graph, a constraint in the form of inequality (7.1) is generated as long as

- Both of these two loops are candidates for fusion (i.e., there are potential fusion edges between each intermediate node that belongs to that path and its parent for both j - and k -loop), and
- At least one potential fusion edge for node s or node t where the two loop indices are different, i.e., one node has potential fusion edge for the j -loop and the other for the k -loop.

Although the first term in the right-hand side of inequalities (7.1) and (7.2) is enough to guarantee legality for most of the fusion configurations, the second term is needed to take care of some legal configurations such as the one in Figure 7.4-(d); without the second term, the configuration in Figure 7.4-(d) appears to be illegal even though it is legal.

for each path $P(s,t)$ and

for each candidate pair of loop indices j and k :

$$(x_{sj} + x_{tk}) - (x_{sk} + x_{tj}) \leq 1 + \sum_{a \in P(s,t) - \{s,t\}} (1 - x_{ak})$$

and $(x_{sk} + x_{tj}) - (x_{sj} + x_{tk}) \leq 1 + \sum_{a \in P(s,t) - \{s,t\}} (1 - x_{aj})$ (7.1)

which can be rewritten as:

$$(x_{sj} - x_{tj}) + \left(-x_{sk} + x_{tk} + \sum_{a \in P(s,t) - \{s,t\}} x_{ak} \right) \leq 1 + m$$

and $(x_{sk} - x_{tk}) + \left(-x_{sj} + x_{tj} + \sum_{a \in P(s,t) - \{s,t\}} x_{aj} \right) \leq 1 + m$ (7.2)

where m is the number of intermediate nodes in the path $P(s,t)$.

A depth-first search algorithm is used in path construction for generating fusion legality constraints. The fusion graph is treated as an undirected acyclic graph during path traversal, i.e., the notion of parent or child is no longer considered during path traversal. At the same time, the fusion edge definition is still as it is in the original graph. For example, consider the fusion graph shown in Figure 7.2-(d) where node c is the parent of both node b and node d . In constructing the

path $P(b,e) = \langle b, c, d, e \rangle$ that originates at node b and ends at node e for loop indices j and k , the unknown variable corresponding to the candidate fusion edge (e,d) is x_{ek} .

7.3 ILP Formulation

7.3.1 Fusion Constraints

The fusion legality constraints in inequality (7.2) work as the set of constraints in the ILP formulations; the objective function developed below completes the formulation. The number of fusion legality constraints may appear to be large, but in practice from our experience with several benchmark expressions from computational chemistry indicates that most of the candidate pairs of loop indices do not exist in all nodes in the fusion graphs; this renders constraint (7.2) inapplicable to most of the paths and hence these constraints are not generated in the ILP formulation. Moreover, as shown in inequality (7.2), the coefficients of the constraint matrices are 1's or 0's, which plays a substantial role in decreasing the solution time from the ILP models as demonstrated in our experimental results.

7.3.2 Fusion Objective Function

Since the objective is to minimize memory usage, an expression for memory usage of an array needs to be developed. Equation (7.3) shows the memory usage for a multidimensional array “A” assuming that the fused loops and unfused loops are known. The memory requirement for array “A” is the product of the sizes along the unfused dimensions of array “A”. Using the associated 0-1 variables introduced in the ILP formulation, in the expression as a trial to get a mathematical formula eligible to be used in an objective function, Equation (7.4) results. Summing over all the arrays (nodes in the fusion graph), the total-memory usage can be used as an objective function as shown in Equation (7.5).

$$mem(A) = \prod_{\substack{i \in Indices(A) \\ i \text{ is unfused}}} N_i \quad (7.3)$$

$$mem(A) = \prod_{\substack{i \in Indices(A) \\ i \text{ is unfused}}} (1 - x_{Ai}) N_i \quad (7.4)$$

$$Minimize: \quad total_memory = \sum_A \left[\prod_{\substack{i \in Indices(A) \\ i \text{ is unfused}}} (1 - x_{Ai}) N_i \right] \quad (7.5)$$

Equation (7.5) is not a suitable form for a mathematical formulation to express an objective function to be minimized. This is because the unfused loops are not known apriori to restrict the

memory expression to include only the unfused loops. In addition, taking off the restriction and including all the loop indices in the memory expression as in Equation (7.6) creates another problem, in that only one loop to be fused in a multi-dimensional array is enough to make the memory contribution of this array in the objective function to be zero. In this way, the effect of fusing one loop in a multi-dimensional array has the same effect as fusing two or more loops, which is not the optimal solution. For example, consider a three-dimensional array A with loop indices $i, j,$ and k ; $\text{mem}(A) = (1-x_{Ai})(1-x_{Aj})(1-x_{Ak})N_iN_jN_k$. Fusing only loop i results in the same objective function value as fusing loops i and j . But in the first case $\text{mem}(A) = N_jN_k$, and in the second case $\text{mem}(A) = N_k$.

The exact memory expression of a multi-dimensional array should include all different combinations of resulting memory after fusion including all its loop indices as shown in Equation (7.7). For example, the memory expression for the three-dimensional array ‘‘A’’ above is as shown in Equation (7.8).

$$\text{mem}(A) = \prod_{i \in \text{Indices}(A)} (1 - x_{Ai}) N_i \quad (7.6)$$

$$\begin{aligned} \text{mem}(A) = & \text{resultant memory of array } A \text{ if none of the loops are fused.} \\ & + \text{if one loop is fused at a time.} \\ & + \text{if two loops are fused at a time} \\ & + \dots \\ & + \text{if all loops are fused.} \end{aligned} \quad (7.7)$$

$$\begin{aligned} \text{mem}(A) = & (1-x_{Ai})(1-x_{Aj})(1-x_{Ak})N_iN_jN_k + \\ & (x_{Ai})(1-x_{Aj})(1-x_{Ak})N_jN_k + (1-x_{Ai})(x_{Aj})(1-x_{Ak})N_iN_k + (1-x_{Ai})(1-x_{Aj})(x_{Ak})N_iN_j + \\ & x_{Ai}x_{Aj}(1-x_{Ak})N_k + x_{Ai}(1-x_{Aj})x_{Ak}N_j + (1-x_{Ai})x_{Aj}x_{Ak}N_i + \\ & x_{Ai}x_{Aj}x_{Ak} \end{aligned} \quad (7.8)$$

Using the memory expression in Equation (7.6) or (7.7) in the objective function results in a nonlinear objective function that needs a nonlinear solver, which is expensive and inefficient (in terms of solution time). Thus, we resort to linearization.

7.3.3 Objective Function Linearization

Linearization of memory expression in Equation (7.7) is very expensive because of the exponential nature of the expression. A less-expensive linear expression for memory usage can be obtained by linearizing Equation (7.6) through summing over all the complements of the $0-1$ variables x_{Ai} 's weighted by the corresponding loop-ranges N_i 's, as shown in Equation (7.9). Since

the objective is minimization, a maximum number of loops are fused as long as the fusion legality is satisfied giving more preference to the loops with larger dimensions.

$$\text{Minimize: } \sum_A \sum_{i \in \text{Indices}(A)} (1 - x_{Ai}) N_i \quad (7.9)$$

This can be rewritten as:

$$\text{Maximize: } \sum_A \sum_{i \in \text{Indices}(A)} x_{Ai} N_i \quad (7.10)$$

Linearization as defined in Equation (7.10) is exact only when all the arrays are one-dimensional arrays but this is not the general case. For example, consider a two-dimensional array A, and loop indices i and j in A with loop-ranges 10 and 15 respectively, and a one-dimensional array B that has loop index k with loop-range 20 and assume that the solver has to choose between j - and k -loops to fuse because of legality constraints. Applying Equation (7.10), the objective function f_{obj} will be: $f_{obj} = 10x_{Ai} + 15x_{Aj} + 20x_{Bk}$. Because the objective in Equation (7.10) is a maximization problem, the ILP solver will set x_{Bk} to 1 and x_{Aj} to 0, which results in $f_{obj} = 30$ and the total memory for this case is $150 + 1 = 151$. On the other hand, if it had set x_{Aj} to 1 and x_{Bk} to 0, $f_{obj} = 15$ (which is less than the other case), but this will result in an optimal memory usage with total memory = $10 + 20 = 30$. This is the key idea used in the efficient linearization of the objective function given by the following function

$$\text{Maximize: } \sum_A \sum_{i \in \text{Indices}(A)} x_{Ai} (\text{size}(A) - \text{rsize}(A, i)) \quad (7.11)$$

where $\text{size}(A)$ is the memory size of array A and $\text{rsize}(A, i)$ is the reduced memory size of array A if the i -loop is fused. The expressions for these turn out to be easily expressed as

$$\text{size}(A) = \prod_{i \in \text{Indices}(A)} N_i, \quad \text{rsize}(A, i) = \prod_{\substack{k \in \text{Indices}(A) \\ \text{and } k \neq i}} N_k$$

The expression $\text{size}(A) - \text{rsize}(A, i)$ expresses the reduction in memory for array A if the i -loop is fused between node A and its parent. From the previous example, $\text{size}(A) - \text{rsize}(A, i) = 150 - 15 = 135$, $\text{size}(A) - \text{rsize}(A, j) = 150 - 10 = 140$, and $\text{size}(B) - \text{rsize}(B, k) = 20 - 1 = 19$. On plugging these values in Equation (7.11), we get $f_{obj} = 135x_{Ai} + 140x_{Aj} + 19x_{Bk}$. Because the objective is maximization, the ILP solver will pick x_{Aj} to be one and x_{Bk} to be zero, this will result in an optimal memory usage for this example with total memory = $10 + 20 = 30$.

7.4 Example

Consider the potential fusion graph in Figure 7.2-(a) (assuming that the ranges for loops i , j , k , l are 10, 10, 12, and 10 respectively). The associated 0-1 variables for each potential fusion edge are shown in Table 7.1; the complete ILP formulation is as shown in Figure 7.5 below. The associate path and its pair of loop indices for each set of constraints in Figure 7.5 are as shown in Table 7.2. The output of the ILP solver is shown in Figure 7.2-(b), where the solid lines represent the resulting fused edges.

Table 7.1: Potential fusion edges and their associated 0-1 variables

edge	(A,f ₁)	(A,f ₁)	(f ₁ ,W)	(B,f ₂)	(B,f ₂)	(B,f ₂)	(C,f ₂)	(C,f ₂)	(f ₂ ,W)	(f ₂ ,W)
loop-index	i	j	j	j	k	l	k	l	j	k
variable	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10

Maximize:

obj: 90 x1 + 90 x2 + 9 x3 + 1080 x4 + 1100 x5 +
1080 x6 + 110 x7 + 108 x8 + 108 x9 + 110 x10

Subject To:

c1: x1 - x2 + x3 <= 1
c2: - x1 + x2 - x3 <= 1
c3: x4 - x5 - x9 + x10 <= 1
c4: - x4 + x5 + x9 - x10 <= 1
c5: - x7 - x9 + x10 <= 1
c6: x7 + x9 - x10 <= 1
c7: x4 - x6 - x9 <= 1
c8: - x4 + x6 + x9 <= 1
c9: - x8 - x9 <= 1
c10: x8 + x9 <= 1
c11: x5 - x6 - x10 <= 1
c12: - x5 + x6 + x10 <= 1
c13: x7 - x8 - x10 <= 1
c14: - x7 + x8 + x10 <= 1
c15: x4 - x5 + x7 <= 1
c16: - x4 + x5 - x7 <= 1
c17: x3 - x9 + x10 <= 1
c18: - x3 + x9 - x10 <= 1
c19: x3 - x4 + x5 + x10 <= 2
c20: - x3 + x4 - x5 + x9 <= 2
c21: x3 + x7 + x10 <= 2
c22: - x3 - x7 + x9 <= 2
c23: x4 - x6 + x8 <= 1
c24: - x4 + x6 - x8 <= 1
c25: x5 - x6 - x7 + x8 <= 1
c26: - x5 + x6 + x7 - x8 <= 1

Figure 7.5: Complete ILP formulation for the fusion graph in Figure 7.2-(a).

Table 7.2: The associated paths and their pairs of loop indices for the constraints in Figure 7.5

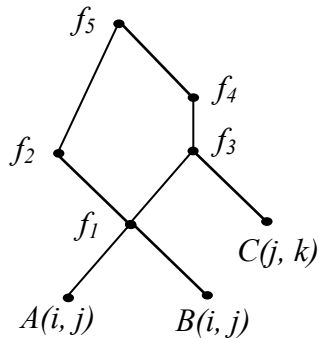
constraints	associated path	loop indices	constraints	associated path	loop indices
c1, c2	$\langle A, f_1, W \rangle$	i, j	c15, c16	$\langle B, f_2, C \rangle$	j, k
c3, c3	$\langle B, f_2, W \rangle$	j, k	c17, c18	$\langle f_1, W, f_2 \rangle$	j, k
c5, c6	$\langle C, f_2, W \rangle$	j, k	c19, c20	$\langle f_1, W, f_2, B \rangle$	j, k
c7, c8	$\langle B, f_2, W \rangle$	j, l	c21, c22	$\langle f_1, W, f_2, C \rangle$	j, k
c9, c10	$\langle C, f_2, W \rangle$	j, l	c23, c24	$\langle B, f_2, C \rangle$	j, l
c11, c12	$\langle B, f_2, W \rangle$	k, l	c25, c26	$\langle B, f_2, C \rangle$	k, l
c13, c14	$\langle C, f_2, W \rangle$	k, l			

7.5 Fusion in the Case of DAGs

When there is a common sub-expression in the formula sequence for the multi-dimensional summation, its graph representation turns out to be a directed acyclic graph (DAG) rather than an expression tree. A multi-parent node (which is the representation of the common sub-expression) may appear in the DAG with multiple different sets of loop indices because multiple references to an array may have different index-variables. Thus, fusion legality constraints in the form of inequalities (7.2) for the fusion in case of a tree representation are not sufficient and cannot be applied directly to the fusion graph resulting from a DAG. Figure 7.6-(b) is an example of a formula sequence for the multi-dimensional summation in Figure 7.6-(a) with its DAG representation shown in Figure 7.6-(c) and the corresponding potential fusion graph in Figure 7.6-(d).

$$W[j] = \sum_{i,k} (A[i, j] \times B[i, j] \times A[j, k] \times B[j, k] \times C[j, k])$$

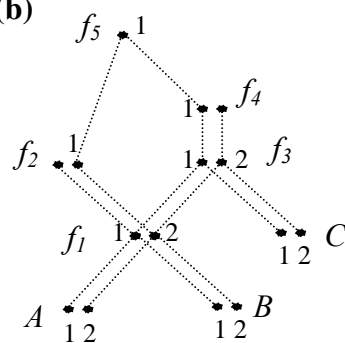
(a)



(c)

$$\begin{aligned} f_1[i, j] &= A[i, j] \times B[i, j] \\ f_2[j] &= \sum_i f_1[i, j] \\ f_3[j, k] &= f_1[j, k] \times C[j, k] \\ f_4[j] &= \sum_k f_3[j, k] \\ W[j] &= f_5[j] = f_2[j] \times f_4[j] \end{aligned}$$

(b)



(d)

Figure 7.6: An example of multi-dimensional summation with common sub-expression: (a) a multi-dimensional summation, (b) a formula sequence for computing (a), (c) the DAG representation of (b), (d) the potential fusion graph for (c).

configurations arising because of referencing the common sub-expression with different sets of index-variables like those in Figure 7.7-(d) can be eliminated by the preprocessing procedure. If after applying the unifying preprocessing, the index-variable sets of a common sub-expression still can not be unified (because of computation correctness), the node corresponding to that common sub-expression should be split into multiple nodes (one for each of its parents) as proposed by Lam [Lam99].

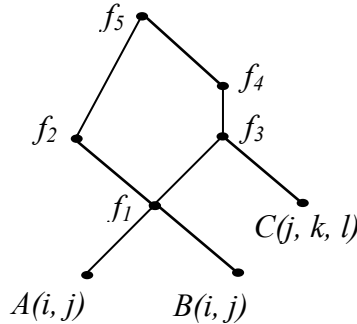
The formula sequence for the multi-dimensional summation needs to be processed to unify the set of index-variables for each array as a preparation for constructing the fusion graph. This can be done through traversing the formula sequence starting from the last formula backwards. Each time an array appears with a different set of index-variables than before, its set of index-variables are adjusted to match the one before. Not only the array with a different index-variable set needing to be adjusted but also all arrays in that formula are adjusted accordingly. For example, Figure 7.8-(c) is the resultant formula sequence for that in Figure 7.8-(b) after the index-variables of array f_l are adjusted. Notice that index-variables of arrays A and B are modified according to the change in those of array f_l .

$$W[j] = \sum_{i,k} (A[j, i] \times B[j, i] \times A[j, k] \times B[j, k] \times C[j, k, l])$$

(a)

$$\begin{aligned} f_1[j, i] &= A[j, i] \times B[j, i] \\ f_2[j] &= \sum_i f_1[j, i] \\ f_3[j, k, l] &= f_1[j, k] \times C[j, k, l] \\ f_4[j, l] &= \sum_k f_3[j, k, l] \\ W[j, l] &= f_2[j, l] \times f_4[j, l] \end{aligned}$$

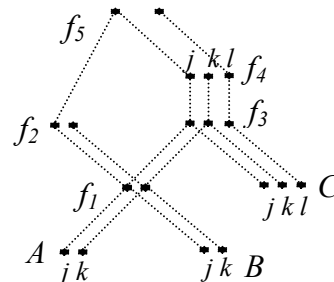
(b)



(d)

$$\begin{aligned} W[j, l] &= f_5[j, l] = f_2[j] \times f_4[j, l] \\ f_4[j, l] &= \sum_k f_3[j, k, l] \\ f_3[j, k, l] &= f_1[j, k] \times C[j, k, l] \\ f_2[j] &= \sum_k f_1[j, k] \\ f_1[j, k] &= A[j, k] \times B[j, k] \end{aligned}$$

(c)



(e)

Figure 7.8: An example of common sub-expression with unifiable sets of index-variables: (a) a multi-dimensional summation, (b) a formula sequence for computing (a), (c) the unified formula sequence for (b), (d) the DAG representation of (c), (e) the potential fusion graph for (d).

In addition to the fusion legality theorem stated before, when a fusion graph is constructed from a tree representation, two main requirements have to be satisfied when we develop a formulation for a fusion graph constructed from a DAG.

1. Due to multiple references to the same multiple-parent node (array), it must have the same fusion with all its parents (each index-variable must either be fused with all the parents or with none of them). This way the arrays are forced to be evaluated once and have only one size. Figure 7.9-(a) is an example of a violation of this property (illegal) while Figure 7.9-(b) is a legal configuration example taking this property into consideration.
2. When there are two different potential fusion scopes with the same loop index i between a multi-parent node v and one of its successors w (fork node defined below), then the two sets of nodes belonging to each one of the potential fusion scopes have to be both fully fused (all intermediate nodes that belong to each of them including node v are fused with their parents) or both be partially fused. If one of the potential fusion scopes is fully fused while the other is partially fused, there will be a node u belonging to the partially fused scopes that can not be located inside the same loop nest with nodes v and w . At the same time, it cannot be located outside that loop nest because node u has to be evaluated after node v and before node w (illegal). Figure 7.9-(c) is an example of illegal configuration where array f_3 belongs to a partially fused chain between f_1 and f_5 while f_1 , f_2 , and f_5 constitute a fusion chain. Figure 7.9-(d) is a legal configuration example that coincides with the requirements stated above.

In developing a mathematical formulation for achieving fusion legality when a fusion graph is constructed from a DAG, we first introduce the term fork node w ($w = \text{fork}(v, a, b)$). It is defined as the end node for the two shortest paths P_1 and P_2 originating at the multi-parent-node v such that: (1) edge $(v, a) \in P_1$ and edge $(v, b) \in P_2$ where a and b are two different parents for node v and (2) $P_1 \cap P_2 = \{v, w = \text{fork}(v, a, b)\}$ only.

The first requirement for fusion legality in case of a DAG where each loop index of a multi-parent node has to be fused with all its parents or none of them, can be achieved through posing a set of constraints like the one in Equation (7.12) for each pair of parents of a multi-parent node.

$$x_{a_{j_1}} - x_{a_{j_2}} = 0. \quad \forall j \in \text{loop_indices}(a) \quad (7.12)$$

where j_1 and j_2 are the two copies of loop index j at each parent of node a , respectively.

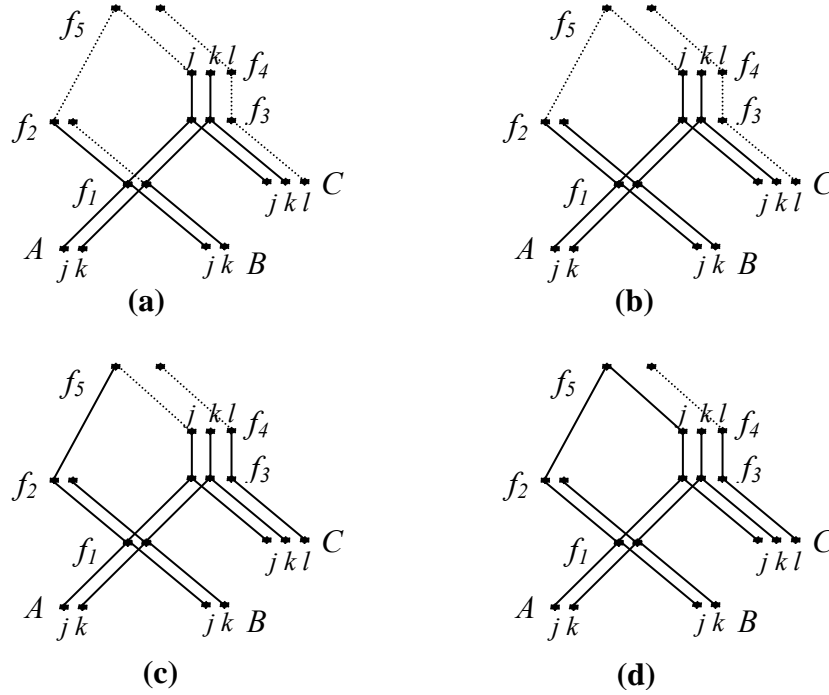


Figure 7.9: Illegal fusion graphs and their corresponding legal graphs for Figure 7.8: (a) illegal fusion graph, (b) legal fusion graph for (a), (c) illegal fusion graph, (d) legal fusion graph for (c).

The mathematical formulation for the second requirement of fusion legality is developed as follows:

1. For each pair of potential fusion scopes of loop index i between a multi-parent node v and its associated fork node $w = \text{fork}(v, a, b)$, define P_1 and P_2 to be those two potential fusion scopes without including node w and let $n(P_1)$ and let $n(P_2)$ be the number of nodes in P_1 and P_2 , respectively; then the following holds:

$$\text{if } \sum_{u \in P_1} x_{ui} = n(P_1), \text{ then } \sum_{u \in P_2} x_{ui} = n(P_2). \quad (7.13)$$

2. By introducing a decision variable that takes a value 1 if the first clause holds and 0 otherwise, condition (7.13) can be rewritten as:

$$\sum_{u \in P_1} x_{ui} - n(P_1) \geq 0 \Rightarrow d_1 = 1$$

$$AND \quad d_1 = 1 \Rightarrow \sum_{u \in P_2} x_{ui} - n(P_2) \geq 0. \quad (7.14)$$

3. Using a transformation from integer programming, we include the upper bound M of the first clause and the lower-bound m of the second clause as well as an adjusting quantity ε in (7.14), it can be rewritten as:

$$\left(\sum_{u \in P_1} x_{ui} - n(P_1) \right) - (M + \varepsilon) d_1 \leq 0.$$

$$AND \quad - \left(\sum_{u \in P_2} x_{ui} - n(P_2) \right) - m d_1 \leq -m. \quad (7.15)$$

4. The upper bound M of the first clause is zero while the lower-bound m of the second clause is $-n(P_1)$ and by using ε to be 1, (7.15) can be rewritten as:

$$\left(\sum_{u \in P_1} x_{ui} - n(P_1) \right) - d_1 \leq 0.$$

$$AND \quad - \sum_{u \in P_2} x_{ui} + n(P_2) d_1 \leq 0. \quad (7.16)$$

As a summery, the second requirement of fusion legality in case of a DAG can be formulated as follows:

*For each multi-parent node v and for each $i \in \text{Indices}(v)$,
For each pair of paths P_1 and P_2 ($= \text{pfscope}(v, w=\text{fork}(v,a,b), i) - \{w\}$) and $a \in P_1$ and $b \in P_2$, the following holds:.*

$$\left(\sum_{u \in P_1} x_{ui} - n(P_1) \right) - d_1 \leq 0.$$

$$- \sum_{u \in P_2} x_{ui} + n(P_2) d_1 \leq 0. \quad (7.17)$$

$$\left(\sum_{u \in P_2} x_{ui} - n(P_2) \right) - d_2 \leq 0.$$

$$- \sum_{u \in P_1} x_{ui} + n(P_1) d_2 \leq 0.$$

The optimal solution for the multi-dimensional summation in Figure 7.8-(a) is shown in Figure 7.9-(d).

7.6 Experimental Results

We have tested our ILP formulation on test examples that arise in the field of correlated electronic structure methods such as coupled cluster and configuration interaction methods in quantum chemistry (see Figure 7.10). Table 7.3 shows the comparison between the memory usage results from the optimal solution and our ILP formulation. It also shows that our formulation is efficient where the solution time is a fraction of seconds even for large test cases.

```

range 3000 a,b,c,d,e,f,mu,nu,om
range 100 i,j,k,la

sum[
  sum[
    sum[sum[sum[F1[mu,nu,om,la]*C1[mu,a],{mu}]]*C2[nu,f],{nu}]]*C3[om,b]
    ,{om}]]*C4[la,k],{la}]
  *
  sum[sum[sum[sum[F2[mu,nu,om,la]*C5[mu,c],{mu}]]*C6[nu,e],{nu}]]*C7[om,b]
    ,{om}]]*C8[la,k],{la}], {b,k}]
  *
  sum[T1[i,j,a,e]*T2[i,j,c,f],{i,j}], {a,e,c,f}]

```

(a)

```

range 3000 a,b,c,d,e,f,mu,nu,om
range 100 i,j,k,la

sum[
  sum[
    sum[sum[sum[F1[mu,nu,om,la,c,e]*
    C1[mu,a],{mu}]]*C2[nu,f],{nu}]]*C3[om,b],{om}]]*C4[la,k],{la}]
  *
  sum[sum[sum[sum[F2[mu,nu,om,la,a,f]*C5[mu,c],{mu}]]*C6[nu,e],{nu}]]*C7[om,
    b],{om}]]*C8[la,k],{la}], {b,k}]
  *
  sum[T1[i,j,a,e]*T2[i,j,c,f],{i,j}], {a,e,c,f}]

```

(b)

```

range 3000 a,b,c,d,e,f,mu,nu,om
range 100 i,j,k,la

sum[
  sum[
    sum[sum[sum[F1[mu,nu,om,la,c,e]*C1[mu,a,f],{mu}]]*C2[nu,f],{nu}]]*C3
    [om,b],{om}]]*C4[la,k],{la}]
  *
  sum[sum[sum[sum[F2[mu,nu,om,la,a,f]*C5[mu,c,e],{mu}]]*C6[nu,e],{nu}]]*C7
    [om,b],{om}]]*C8[la,k],{la}],{b,k}]
  *
  sum[T1[i,j,a,e]*T2[i,j,c,f],{i,j}], {a,e,c,f}]

```

(c)

Figure 7.10: Test examples: (a) test 1, (b) test 2, (c) test 3, (d) test 4, (e) test 5.

```

range 3000 a,b,c,d,e,f,mu,nu,om
range 100 i,j,k,la

sum[
  sum[
    sum[sum[sum[F1[mu,nu,om,la,c]*C1[mu,a],{mu}]*C2[nu,f],{nu}]*C3[om
      ,b],{om}]*C4[la,k],{la}]
    *
    sum[sum[sum[sum[F2[mu,nu,om,la,a]*C5[mu,c],{mu}]*C6[nu,e],{nu}]*C7[om
      ,b],{om}]*C8[la,k],{la}],{b,k}]
    *
    sum[T1[i,j,a,e]*T2[i,j,c,f],{i,j}], {a,e,c,f}]

```

(d)

```

range 3100 la,mu,nu,om
range 3000 a,b,c,d,e,f
range 100 i,j,k

sum[
  sum[
    sum[sum[sum[F1[mu,nu,om,la,c]*C4[la,k],{la}]*C1[mu,a],{mu}]*C2[nu
      ,f],{nu}]*C3[om,b],{om}]
    *
    sum[sum[sum[sum[F2[mu,nu,om,la,a]*C8[la,k],{la}]*C5[mu,c],{mu}]*C6[nu
      ,e],{nu}]*C7[om,b],{om}], {b,k}]
    *
    sum[T1[i,j,a,e]*T2[i,j,c,f],{i,j}], {a,e,c,f}]

```

(e)

fig. cont'd

Table 7.3: Memory usage of some benchmarks after fusion

Problem	Optimal Sol (Lam)	Our ILP	time(sec)
FG in Figure 7.1	23	23	0.01
Test 1	2.700909300006 e+12	2.700909300006 e+12	0.05
Test 2	2.700900000206000 e+12	2.700900000206000 e+12	0.17
Test 3	600008	600008	0.10
Test 4	1.809003105 e+9	1.809003105 e+9	0.09
Test 5	1.62027018006003 e+14	1.62027018006003 e+14	0.18

7.7 Chapter Summary

This chapter has presented a novel ILP formulation for the problem of minimizing memory usage of arrays in loop computations that express multi-dimensional integrals in computational chemistry and materials characterization. In these computations, different ways of performing the loop computations has vastly different memory requirements on the code. Unlike other approaches to the problem that rely on exhaustive search, our ILP formulation is shown to be very effective on several test cases.

CHAPTER 8

CONCLUSION AND FUTURE WORK

This thesis has addressed two important factors in embedded system design, namely, power consumption minimization and memory optimization. Chapters 2, 3, 4 and 5 considered the problem of power consumption (peak power as well as average power) minimization in high-level synthesis. The techniques developed in this work are mainly data-flow-based synthesis techniques that handle the data-dominated applications such as signal and image processing. Chapters 6 and 7 dealt with the memory usage optimization mainly targeting a restricted class of computations that arise in scientific computing such as electronic structure calculations in quantum chemistry.

8.1 Contributions of This Work

In Chapter 2, we have presented a unique mixed integer-linear programming (MILP) formulation for the scheduling problem using multiple supply-voltages in order to optimize peak power as well as average power and energy consumption. Our exact solution for optimal peak and/ or average power scheduling uses the smallest number of variables among the existing formulations in literature and is considered under two scenarios: time constraint alone, and both time and resource constraints. Then, we have devised a novel two-phase heuristic to solve the multiple supply-voltages scheduling for peak and average power minimization for the same scenarios. First, we developed a guided linear programming (LP) relaxation (one in which the requirement that certain variables is relaxed) solution in which the MILP formulation is iteratively relaxed to obtain a solution for minimum peak and/or average power. Then in the second phase, a power-resources saving procedure is developed for several reasons: (i) to overcome any violation of resource constraints in the solution from the first phase for a TRCS problem; (ii) to achieve minimal resource usage in case of TCS; and (ii) to restructure the output LP schedule in order to obtain larger power saving. The results for peak and average power of

our two-phase heuristic well match those obtained by the optimal solution and have been validated by extensive experiments on several benchmarks.

The problem of scheduling for peak and average power minimization in HLS using multiple supply voltages under a time-constraint is again considered in Chapter 3. We have developed a two-phase heuristic to obtain near-optimal solutions with small execution times. The first phase is a modified power-force-directed scheduling (MPFDS) heuristic based on the well-known force-directed scheduling technique. The MPFDS heuristic tries to minimize power (peak and average) consumption by assigning the smallest voltage level possible to each operation from the given set of voltage levels without violating the time-constraint; at the same time, it distributes the operations in the data-flow graph over the total allowed time in order to balance power consumption and to achieve minimum peak power. The second phase is a post-processing procedure (power-resource-saving) that analyzes the output schedule from the first phase to exploit the available room to get more power and/or operating resources minimization. Results show that our proposed heuristic is capable of achieving near-optimal results with polynomial-time complexity.

In Chapter 4, we have addressed the problem of power optimization during a combined module-selection and scheduling process. We have presented a unified MILP formulation for these two inter-related tasks (module selection and scheduling) in HLS, which targets peak power as well as average power optimization. We considered the problem of module-selection and scheduling under two cases: time-constraint alone (TCMSS), and time and area constraints (TACMSS). First, we presented an MILP formulation for finding the optimal solution for peak and average power. Next, we have devised a two-phase heuristic for the same problem. In the first phase of our heuristic, the integrality in the MILP formulation is relaxed and a constructive LP solution is developed to get a near-optimal solution in a reasonable amount of time. The second phase is a power-area saving procedure that attempts to exploit any available room in the resulting LP solution to achieve further saving in peak and average power; and at the same time restructure the resulting LP solution in order to remove any violation of the area constraint in the LP solution. Extensive experiments on several benchmarks show that our developed heuristic achieves peak and average power minimization that are competitive with the results obtained by our MILP formulation.

In Chapter 5, we have presented two methods for optimizing dynamic average power as well as peak power in scheduling sequential synchronous designs under time-constraint using a combination of basic retiming and multiple voltage scheduling (MVS) techniques. Retiming is used to restructure the input (SDFG) representation of a sequential circuit in order to increase the parallelism between operations and thus to increase the number of operations that are not on the critical path to be candidates for scheduling at lower supply voltages. First, we devised an MILP formulation for optimal peak and/or average power consumptions scheduling problem through a unification of retiming and MVS techniques. The mathematical formulation for peak power requires capturing the information on the activity of each operation in each time step. Thus, we used binary variables for that purpose, which results in very large problem sizes because tight time-frames for operations cannot be obtained a priori. Then, to alleviate the problem of variable explosion in the MILP formulation, we presented a two-stage algorithm for peak and/or average power optimization. First, power-oriented retiming is proposed to restructure the input SDFG in order to achieve parallelization to the favor of nodes with high power consumption. Second, an MILP formulation is presented that takes the retimed DFG as an input and produces an optimal peak and/or average power schedule using MVS technique. Our proposed solution for power-oriented retiming generates a graph structure candidate better suited for peak and average power saving than other works in the literature which solely rely on maximizing parallelism among graph nodes. This is demonstrated by experimental results for several benchmarks.

In Chapter 6, we addressed the problem of memory usage optimization that arises in the evaluation of expression trees involving large data objects. We are interested in the situations where the data objects are so large to fit in memory that they have to be dynamically allocated and de-allocated during expression evaluation. This problem arises in electronic structure calculations in quantum chemistry and in several other contexts. We considered three different variations of the memory usage optimization problem. The first problem is one of finding an evaluation order for the dependent data objects in order to achieve the least amount of required memory. The second problem is the generalization of the evaluation order problem in which the data objects have different processing times and the least amount of memory is sought for a given total execution time given as a constraint. The third problem is the complement of the second problem: the total execution time must be minimized for a given maximum amount of memory specified as a constraint. We have developed an ILP formulation to optimally solve

these three problems. In addition we presented a memory-force-directed heuristic (motivated by the well-known force-directed scheduling heuristic) to solve for minimal memory usage in the second problem. Results show that the ILP solution time is small especially when the time-constraint is close to the critical path length or when the memory-constraint is less stringent. The results from the heuristic are very close to those from the ILP formulations, but require much smaller execution times.

In Chapter 7, we have presented a novel ILP formulation for the problem of minimizing memory usage of arrays in loop computations that express multi-dimensional integrals in computational chemistry and materials characterization. In these computations, different ways of performing the loop computations have vastly different memory requirements on the code. Unlike other current approaches to the problem that rely on exhaustive search, our ILP formulation is shown to be very effective on several test cases. Work is in progress in integrating the effects of other transformations such as tiling into our ILP framework.

8.2 Future Work

8.2.1 Power Optimization for Control-Flow Applications

The developed techniques in the first part of this thesis are data-flow-based synthesis techniques that handle the data-dominated applications such as DSP and image processing. Data-flow-based approach has its limitations. It can only deal with a straight-line sequence of operations inside a basic block, and cannot handle branching or loops. There are many other applications that include nested loops and conditionals, e.g., control-intensive applications such as industrial controllers and network traffic monitors. These applications dictate the integration of both data-flow and control-flow approaches, and they should be targeted by the low-power design in HLS.

Although a large body of research is devoted to the data-flow-based synthesis, only a small amount of work incorporates branching and loops in the synthesis; very few of these works target power minimization. Most of the research work that handle the control-flow-intensive applications addressed performance and area optimization. The algorithms that optimize performance can be classified as *path-based* scheduling approaches that aim for scheduling each individual path as fast as possible [Cam91, BR97], hierarchical approaches [KY94, MW01], or *code-transformational* approaches that try to restructure the syntactic of the input code while the

semantics are fully reserved. This is done through moving some operations between basic blocks [RF95, MW97, GD03], or using a speculative execution approach [KW99, GK01, GG04]. Other algorithms handle area minimization through maximization of resource sharing [HC88, WY89, WT92, RB95, SL98]. The main idea of resource sharing is to schedule the operations in the mutually exclusive parts of conditional branches in the same control step to share the same resource since they will not be executed at the same time. None of these works considered power optimization. Power reduction in control-intensive applications has been addressed in some of the research work from different aspects. Some of these algorithms optimize power consumptions by tackling the switching activity factor through controller re-specification [RD97], through restructuring the multiplexers tree [KL99], or through code transformation [LR99]. Others addressed power reduction by using the power management technique [LR98, CS02].

Although the largest power reduction gain can be achieved through voltage scaling, there is no work (to the best of our knowledge) has tackled the power consumption minimization in control-flow intensive applications through using this efficient technique. Multiple voltages scheduling technique can be used to reduce the dynamic power consumption in control-flow intensive applications in different approaches. One approach can be achieved through applying the MVS to each basic block individually for different values of input time constraints first, then the whole CDFG (control- and data-flow graph) is solved using the MVS technique by treating each basic block as a node with different pairs of delay and power consumptions obtained from the first step. This approach lacks the incorporation of resource sharing and it might result in suboptimal solution even if the MVS used the MILP formulation in the two steps. Another potential solution is applying the MVS technique to the original CDFG with careful consideration of resource sharing. Moreover, research needs to be conducted for optimizing peak power consumption in control-flow intensive applications in some way similar to and alongside with resource sharing.

8.2.2 Tight Time-Frames in the MILP Formulation for Unified Peak Power and Retiming

As has been presented in Chapter 5, that combining peak power consumption with retiming in an ILP formulation dictates a huge number of binary variables because of the difficulty to impose a restriction on time-frames of the SDFG nodes. An investigation needs to be done to understand and to correlate the relationship between the SDFG structure and the lower and upper

bound on the control-steps that a node is potentially scheduled to. Doing so will restrict the time frames and thus reduce the number of binary variables associated with each node. Thus, a larger design sizes can be solved with affordable solution time without scarifying optimality.

8.2.3 Modeling Leakage Power

Leakage power in CMOS is a form of static power that is consumed even when the integrated circuit is not active. The non-ideal switching behavior of a CMOS transistor in the off state results in a current flow from the power supply to ground through the transistor even though the transistor is logically in the off state. Incorporating the modeling of leakage power in our solutions is a challenging problem worth exploring.

8.2.4 Incorporating Disk Access Cost with Fusion

In Chapter 7, we have developed a mathematical formulation for static memory minimization using loop-fusion technique for those applications that require an interaction between large data arrays such as Fast Fourier Transform and coupled cluster and configuration interaction methods in quantum chemistry. In many occasions, data cannot fit in the memory, even after loop fusion. In such context, it is necessary to develop so called *out-of-core* algorithms that explicitly manage the movement of subsets of the data between main memory and secondary disk storage [BK04].

Cociorva et al. [CB01a, CB01b] have addressed this problem through developing an approach to fusion and tiling transformation. They focused on minimizing memory-to-cash movement, which is similar to disk-to-memory movement. The introduced algorithm in [CB01b] decouples the fusion step from tiling step that results in sub-optimal solution. Bibireata et al. [BK04] build on the work in [CB01b] by trying to consider an integrated approach for fusion and tiling optimization, but their developed algorithm did not consider a full integration of fusion and tiling. Instead, they produced a set of candidate loop structures first, then for each set of these loop structures, they decide on which temporary arrays to be written to the disk and they determined the tile size that minimize the disk access cost. This limitation drives the need for research work to be conducted to optimize for memory and disk access costs simultaneously.

REFERENCES

- [AD95] I. Ahmad, M. K. Dhodhi, and C. Y. R. Chen, "Integrated Scheduling, Allocation and Module Selection for Design-Space Exploration in High-Level Synthesis," *IEE Proc. on Computers and Digital Techniques*, vol. 142, no. 1, pp. 65-71, Jan. 1995.
- [Aul96] W. Aulbur, *Parallel Implementation of Quasi-Particle Calculations of Semiconductors and Insulators*, Ph.D. Dissertation, Ohio State University, Oct. 1996.
- [BC95] F. Balasa, F. Catthoor, and H.D. Man, "Background Memory Area Estimation for Multidimensional Signal Processing Systems," *IEEE Trans. on VLSI Systems*, vol. 3, no. 2, pp.157-172, June 1995.
- [BK04] A. Bibireata, S. Krishnan, G. Baumgartner, D. Cociorva, C. Lam, P. Sadayappan, J. Ramanujam, D. Bernholdt, and V. Choppella, "Memory-Constrained Data Locality Optimization for Tensor Contractions," in *Languages and Compilers for Parallel Computing*, Lecture Notes in Computer Science, vol. 2958, pp. 93-108, Springer-Verlag, 2004.
- [Bly97] S. Blythe, *The Exploration of Extended Search Spaces in High-Level Synthesis*, PhD thesis, Computer Science Department, Rensselaer Polytechnic Institute, 1997.
- [BM00] L. Benini, and G. De Micheli, "System Level Power Optimization: Techniques and Tools," *ACM Trans. Design Automation of Electronic Systems*, vol. 5, no. 2, pp. 115-192, Apr. 2000.
- [BR97] R. Bergamaschi, S. Raje, and L. Trevillyan, "Control-Flow Versus Data-Flow Based Scheduling: Combining Both Approaches in an Adaptive Scheduling System," *IEEE Trans. Very Large Scale Integration Systems*, vol. 5, no. 1, pp. 82-100, 1997.
- [Cam91] R. Camposano, "Path-Based Scheduling for Synthesis," *IEEE Trans. on Computer-Aided Design*, vol. 10, no. 1, pp. 85-93, Jan. 1991.
- [CB01a] D. Cociorva, J. Wilkins, C.-C. Lam, G. Baumgartner, P. Sadayappan, and J. Ramanujam, "Loop Optimization for a Class of Memory-constrained Computations," in *Proc. 15th ACM International Conference on Supercomputing (ICS'01)*, pp. 103-113, June 2001.
- [CB01b] D. Cociorva, J. Wilkins, G. Baumgartner, P. Sadayappan, J. Ramanujam, M. Nooijen, D. Bernholdt, and R. Harrison, "Towards Automatic Synthesis of High-Performance Codes for Electronic Structure Calculations: Data Locality Optimization," in *Proc.*

- Intl. Conf. on High Performance Computing*, Lecture Notes in Computer Science, vol. 2228, pp. 237-248, Springer-Verlag, 2001.
- [CB97] S. Chaudhuri, S. A. Blythe, and R. A. Walker, "A Solution Methodology for Exact Design Space Exploration in a Three-Dimensional Design Space," *IEEE Trans. on VLSI Systems*, vol. 5, no. 1, pp. 69-81, Mar. 1997.
- [CC03a] N. Chabini, I. Chabini, E.-M. Aboulhamid, and Y. Savaria, "Methods for Minimizing Dynamic Power Consumption in Synchronous Designs with Multiple Supply Voltages," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 22, no. 3, pp. 346-351, Mar. 2003.
- [CC03b] N. Chabini, I. Chabini, E.-M. Aboulhamid, and Y. Savaria, "Unification of Basic Retiming and Supply Voltage Scaling to Minimize Dynamic Power Consumption for Synchronous Digital Designs," *Proc. Great Lakes Symp. on VLSI*, pp. 221-224, Apr. 2003.
- [CG93] S. Chatterjee, J. R. Gilbert, R. Schreiber, and S.-H. Teng, "Automatic Array Alignment in Data-Parallel Programs," *Proc. 20th Annual ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages*, New York, pp. 16-28, 1993.
- [CG95] S. Chatterjee, J. R. Gilbert, R. Schreiber, and S.-H. Teng, "Optimal Evaluation of Array Expressions on Massively Parallel Machines," *ACM Transactions on Programming Languages and Systems*, vol. 17, no. 1, pp. 123-156, Jan. 1995.
- [CHd95] S. Chaudhuri, *Scheduling and Design Space Exploration in High-Level Synthesis*, PhD thesis, Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, 1995.
- [Cho93] L.-F. Chao, *Scheduling and Behavioral Transformations for Parallel Systems*, PhD thesis, Dept. of Computer Science, Princeton University, 1993.
- [CJ91] L.-G. Chen and L.-G. Jeng, "Optimal Module Set and Clock Cycle Selection for DSP Synthesis," in *Proc. of IEEE Int'l. Conf. on Circuits and Systems (ISCAS'91)*, pp. 2200-2203, June 1991.
- [CM98] P.-Y. Calland, A. Mignotte, Y. Robert, and F. Vivien, "Retiming DAGs", *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 17, no. 12, Dec. 1998.
- [CP95] J. Chang and M. Pedram, "Register Allocation and Binding for Low Power" in *Proc. of Design Automation Conference*, pp. 29-35, June 1995.
- [CP96] J. Chang and M. Pedram, "Module Assignment for Low Power," in *Proc. of EURODAC 96*, pp. 376-381, Sep. 1996.
- [CP97] J. Chang and M. Pedram, "Energy Minimization Using Multiple Supply Voltages," *IEEE Trans. on VLSI Systems*, vol. 5, no. 4, pp. 436-443, Dec. 1997.

- [CPX02] ILOG Inc., *ILOG CPLEX 8.1 Reference Manual*, Mountain View, CA, <http://www.ilog.com/products/cplex>, 2002.
- [CR95] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. W. Brodersen. "Optimizing Power Using Transformation," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 14, no. 1, pp. 12--30, Jan. 1995.
- [CS00] C. Chantrapornchai, E. Sha, and X. Hu, "Efficient Design Exploration Based on Module Utility Selection," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 1, pp. 19-29, Jan. 2000.
- [CS02] C. Chen and M. Sarrafzadeh, "Power-Manageable Scheduling Technique for Control Dominated High-Level Synthesis," in *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 1016-1020, 2002.
- [CT90] R. J. Cloutier and D. E. Thomas, "The Combination of Scheduling, Allocation and Mapping in a Single Algorithm," in *Proc. 27th ACM/IEEE Design Automation Conference*, pp. 71-76, June 1990.
- [CW04] N. Chabini and W. Wolf, "Reducing Dynamic Power Consumptions in Synchronous Sequential Digital Designs Using Retiming and Supply Voltage Scaling," *IEEE Trans. on VLSI Systems*, vol. 12, no. 6, pp. 573-589, June 2004.
- [CW96] S. Chaudhuri and R. A. Walker, "Computing Lower Bounds on Functional Units before Scheduling," *IEEE Trans. on VLSI Systems*, vol. 4, no. 2, pp. 273-9, 1996.
- [FL91] C. N. Fischer and R. J. LeBlanc Jr., *Crafting a Compiler*, Benjamin/Cummings, Menlo Park, CA, 1991.
- [FS95] G. E. Tellez, A. Farrahi, and M. Sarrafzadeh, "Activity Driven Clock Design for Low Power Circuits," in *Proc. Int. Conf. Computer-Aided Design*, pp. 62--65, Nov. 1995.
- [GD03] S. Gupta, N. D. Dutt, R. K. Gupta, and A. Nicolau, "SPARK: A High-Level Synthesis Framework for Applying Parallelizing Compiler Transformations," in *Proc. Int. Conf. VLSI Design*, pp. 461--466, 2003.
- [GE93] C. Gebotys and M. Elmasry, "Global Optimization Approach for Architectural Synthesis," *IEEE Trans on CAD of Integrated Circuits and Systems*, vol. 12, no. 8, pp. 1266-1278, Sep. 1993.
- [GG04] S. Gupta, N. Savoiu, N. D. Dutt, R. K Gupta, and A. Nicolau, "Using Global Code Motions to Improve the Quality of Results for High-Level Synthesis," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 23, no. 2, Feb. 2004.
- [GK01] S. Gupta, N. Savoiu, S. Kim, N. D. Dutt, R. K. Gupta and A. Nicolau, "Speculation Techniques for High-Level Synthesis of Control Intensive Designs," in *Proc. of Design Automation Conference (DAC)*, pp. 269-272, Jun. 2001.

- [GK02] S. Gupta and S. Katkoori, "Force-Directed Scheduling for Dynamic Power Optimization," in *Proc. IEEE Computer Society Annual Symposium on VLSI*, pp. 68-73, 2002.
- [GK83] D. Gajski and R. H. Kuhn, "Guest Editors Introduction-New VLSI Tools," *IEEE Computer*, vol. 16, no. 2, pp. 14-17, 1983.
- [GM92] P. Gutberlet, J. Muller, H. Kramer, W. Rosenstiel, "Automatic Module Allocation in High Level Synthesis," in *Proc. of EURODAC 92*, pp. 328-333, Sep. 1992.
- [GO92] G. Gao, R. Olsen, V. Sarkar, and R. Thekkath, "Collective Loop Fusion for Array Contraction," in *Languages and Compilers for Parallel Computing*, Lecture Notes in Computer Science, Springer-Verlag, Aug. 1992.
- [GS97] S. Gailhard, O. Sentieys, N. Julien, and E. Martin, "Area/Time/Power Space Exploration in Module Selection for DSP High-Level Synthesis," in *Proc. Int. Workshop, PATMOS'97*, pp. 35-44, Sep. 1997.
- [GW78] L. J. Guibas and D. K. Wyatt, "Compilation and Delayed Evaluation in APL," in *Proc. 5th Annual ACM Symp. Principles of Programming Languages*, pp. 1-8, 1978.
- [HC02] R. Henning and C. Chakrabarti, "An Approach to Switching Activity Consideration During High Level Low Power Design Space Exploration," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 49, no. 5, pp. 339-351, May 2002.
- [HC88] K. S. Huang, A. E. Casavant, C. Chang, and M. A. d'Abreu, "Constrained Conditional Resource Sharing in Pipeline Synthesis," in *Proc. of IEEE Int'l Conf. on Computer-Aided Design*, pp. 52- 55, 1988.
- [HL86] M. S. Hybertsen and S. G. Louie, "Electronic Correlation in Semiconductors and Insulators: Band Gaps and Quasi-particle Energies," *Phys. Rev. B*, 34, pp. 5390, 1986.
- [HP83] L. J. Hafer and A. C. Parker, "A Formal Method for the Specification, Analysis, and Design of Register-Transfer Level Digital Logic," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 2, no. 1, pp. 4-18, Jan. 1983.
- [HR03] S.W. Haga, N. Reeves, R. Barua, and D. Marculescu, "Dynamic Functional Unit Assignment for Low Power," in *Proc. IEEE Design, Automation and Test in Europe Conf. (DATE)*, Munich, Germany, March 2003.
- [Jai90] R. Jain, "MOSP: Module Selection for Pipelined Designs with Multicycle Operations," in *Proc. Intl. Conf. Computer Aided Design*, pp. 212-215, 1990.
- [JR97] M. C. Johnson and K. Roy, "Datapath Scheduling with Multiple Supply Voltages and Level Converters," *ACM Trans. on Design Automation and Electronic Systems*, pp. 227-248, Jul. 1997.

- [KK95] N. Kumar, S. Katkooi, L. Rader, and R. Vemuri, "Profile-Driven Behavioral Synthesis for Low Power VLSI Systems," *IEEE Design and Test of Computers*, vol. 12, no. 2, pp. 70-84, 1995.
- [KL99] K. S. Khouri, G. Lakshminarayana, and N. K. Jha, "High-Level Synthesis of Low Power Control-Flow Intensive Circuits," *IEEE Trans. Computer-Aided Design*, vol. 18, no. 12, pp. 1715–1729, Dec. 1999.
- [KM93] K. Kennedy and K. S. McKinley, "Maximizing Loop Parallelism and Improving Data Locality via Loop Fusion and Distribution," in *Languages and Compilers for Parallel Computing*, pp. 301–320, Springer-Verlag, Aug. 1993.
- [KN92] G. Krishnamoorthy, and J. A. Nestor, "Data Path Allocation using an Extended Binding Model," in *Proc. 29th ACM/IEEE Design Automation Conference*, pp.279-284, 1992.
- [KW99] A. Kountouris and C. Wolinski, "Combining Speculative Execution and Conditional Resource Sharing to Efficiently Schedule Conditional Behaviors," in *Proc. of ASP-DAC*, pp. 343-346, Jan. 1999.
- [KY94] T. Kim, N. Yonezawa, J. Liu, and C. Liu, "A Scheduling Algorithm for Conditional Resource Sharing—A Hierarchical Reduction Approach," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 13, no. 4, Apr. 1994.
- [Lam99] C. Lam, *Performance Optimization of a Class of Loops Implementing Multi-Dimensional Integrals*, PhD thesis, Technical Report OSU-CISRC-8/99-TR22, Dept. of Computer and Information Science, The Ohio State University, Columbus, Ohio, Aug. 1999.
- [Lin97] Y.-R. Lin, C.-T. Hwang, and A. C.-H. Wu, "Scheduling Techniques for Variable Voltage Low Power Designs," *ACM Trans. on Design Automation and Electronic Systems*, vol. 2, no. 2, pp. 81–97, Apr. 1997.
- [LJ99] G. Lakshminarayana and N. K. Jha, "FACT: A Framework For Applying Throughput and Power Optimizing Transformations to Control-Flow Intensive Behavioral Descriptions," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 18, no. 11, pp. 1577-1594, Nov. 1999.
- [LK03] C.-G. Lyuh and T. Kim. "High-level Synthesis for Low-Power Based on Network Flow Method," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 3, pp. 364-375, June 2003.
- [LM01] M. Lundberg, K. Muhammad, K. Roy, and S. K. Wilson. "A Novel Approach to High Level Switching Activity Modeling with Applications to Low Power DSP System Synthesis." *IEEE Transactions on Signal Processing*, vol. 49, no. 12, pp. 3157-3167, Dec. 2001.

- [LM96] P. Landman, R. Mehra, J. M. Rabaey, "An Integrated CAD Environment for Low-Power Design," *IEEE Design and Test of Computers*, vol. 13, no. 2, pp. 72-82, 1996.
- [LP93] L. E. Lucke and K. K. Parhi, "Data-Flow Transformations for Critical Path Time Reduction in High-Level DSP Synthesis," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 12, no. 7, pp. 1063-1068, Jul. 1993.
- [LR98] G. Lakshminarayana, A. Raghunathan, N. K. Jha, and S. Dey, "Transforming Control-Flow Intensive Designs to Facilitate Power Management," in *Proc. of IEEE Int'l Conf. on Computer-Aided Design*, pp. 657-664, Nov. 1998.
- [LR99] G. Lakshminarayana, A. Raghunathan, N.K. Jha, and S. Dey, "Power Management in High Level Synthesis." *IEEE Trans. on VLSI Systems*, vol. 7, no. 1, pp. 7-15, Mar. 1999.
- [LS86] C. E. Leiserson, J. B. Saxe, "Retiming Synchronous Circuitry," Technical Report: SRC-RR-13, Digital Equipment Corporation Systems Research Center, Aug. 1986.
- [LS97] C. Lam, P. Sadayappan, and R. Wenger, "On Optimizing a Class of Multi-Dimensional Loops with Reductions for Parallel Execution," *Parallel Processing Letters*, vol. 7, no. 2, pp. 157-168, 1997.
- [LS99a] C. Lam, P. Sadayappan, D. Cociorva, M. Alouani, and J. Wilkins, "Performance Optimization of a Class of Loops Involving Sums of Products of Sparse Arrays," in *Proc. Ninth SIAM Conf. on Parallel Processing for Scientific Computing*, Mar. 1999.
- [LS99b] C. Lam, D. Cociorva, G. Baumgartner, and P. Sadayappan, "Memory-Optimal Evaluation of Expression Trees Involving Large Objects," Technical Report OSU-CISRC-5/99-TR13, Dept. of Computer and Information Science, The Ohio State University, May 1999.
- [MA95] N. Manjikian and T. S. Abdelrahman, "Fusion of Loops for Parallelism and Locality," in *Proc. International Conference on Parallel Processing*, pp. II:19-28, Aug. 1995.
- [MC02] A. Manzak and C. Chakrabarti, "A Low Power Scheduling Scheme with Resources Operating at Multiple Voltages." *IEEE Trans. on VLSI Systems*, vol. 10, No 1, pp. 6-14, Feb. 2002.
- [MC95] E. Musoll and J. Cortadella, "Scheduling and Resource Binding for Low Power", in *Proc. IEEE International Symposium on System Synthesis*, pp. 104-109, Apr. 1995.
- [MC99] C. A. Mandal, P. P. Chakrabarti, and S. Ghose, "A Design Space Exploration Scheme for Data Path Synthesis," *IEEE Transactions on VLSI*, vol. 7, no. 3, pp. 331-338, 1999.

- [MD93] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming Sequential Circuits for Low Power," *Proc. IEEE/ACM Int'l Conference on Computer-Aided Design*, pp. 398-402, Nov. 1993.
- [MD96] J. Monteiro, S. Devadas, P. Ashar, and A. Mauskar, "Scheduling Techniques to Enable Power Management," in *Proc. Design Automation Conf.*, pp. 349-352, June 1996.
- [MP98] E. Macii, M. Pedram, and F. Somenzi, "High-Level Power Modeling, Estimation and Optimization," *IEEE Trans. on CAD*, vol. 17, no. 11, pp. 1061-1079, 1998.
- [MR03] S. P. Mohanty, N. Ranganathan, and S. K. Chappidi, "Simultaneous Peak and Average Power Minimization During Datapath Scheduling for DSP Processors," in *Proc. ACM Great Lakes Symposium on VLSI*, pp. 215-220, 2003.
- [MW01] C. Murphy and X. Wang, "Most Often Used Path Scheduling Algorithm," in *Proc. of the 5th World Multi-Conf. on Systemic, Cybernetics and Informatics (SCI)*, vol. 12, pp. 289-295. Jul. 2001.
- [MW97] M Munch, N. Wehn, and M. Glesner, "An Efficient ILP-Based scheduling Algorithm for Control-Dominated VHDL descriptions." *ACM Trans. on Design Automation of Electronic Systems*, vol. 2, no. 4, pp. 344-364, 1997.
- [NK93] J. A. Nestor and G. Krishnamoorthy, "SALSA: A New Approach to Scheduling with Timing Constraints," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 12, no. 8, pp. 1107-1122, Aug. 1993.
- [NS01] T. W. O'Neil, and E.H.-M. Sha, "Retiming Synchronous Data-Flow Graphs to Reduce Execution Time," *IEEE Trans. on Signal Processing*, vol. 49, no. 10, pp. 2397-2407, Oct. 2001.
- [NT99] T. W O'Neil, S. Tongshima, and E.H.-M. Sha, "Extended Retiming: Optimal Scheduling Via A Graph-Theoretical Approach," in *Proc. IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing*, pp. 2001--2004, Mar. 1999.
- [NW88] G. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization*, John Wiley, New York, 1988.
- [PK89] P. G. Paulin and J. P. Knight, "Force Directed Scheduling for the Behavior Synthesis of ASICs," *IEEE Transactions on CAD*, vol. 8, pp. 661-679, June 1989.
- [PR94] M. Potkonjak and J.M. Rabaey, "Optimizing Resource Utilization Using Transformations," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 13, no. 3, pp. 277-292, Mar. 1994.
- [PS02] G. Papa and J. Silc, "Automatic Large-Scale Integrated Circuit Synthesis Using Allocation-Based Scheduling Algorithm", *Microprocessors and Microsystems*, vol. 26, no. 3, pp. 139-147, 2002.

- [PS98] M. Potkonjak and M.B. Srivastava, "Behavioral Optimization Using the Manipulation of Timing Constraints", *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 936-947, Oct. 1998.
- [RV03] B. Radhakrishnan and M. Venkatesan, "Multiple Voltage and Frequency Scheduling for Power Minimization," in *Proc. Euromicro Symp. on Digital System Design*, pp. 279-285, Sep. 2003.
- [RB95] I. Radivojevic and E Brewer, "Analysis of Conditional Resource Sharing Using a Guard-Based Control Representation," in *Proc. of the ICCD'95*, pp. 434-439, Oct. 1995.
- [RD97] A. Raghunathan, S. Dey, N. K. Jha, and K. Wakabayashi, "Power Management Techniques for Control-Flow Intensive Designs," in *Proc. IEEE Design Automation Conference*, pp. 429-434, Jun. 1997.
- [RJ95a] A. Raghunathan and N. K. Jha, "Minimizing Switched Capacitance During Data Path Allocation", in *Proc. IEEE Symposium on Circuits and Systems*, 1995.
- [RJ95b] A. Raghunathan and N. K. Jha, "Iterative Improvement Algorithm for Low Power Datapath Synthesis", in *Proc. IEEE/ACM Int'l Conference on Computer-Aided Design*, Nov. 1995.
- [RD98] A. Raghunathan, N.K. Jha, and S. Dey, *High-Level Power Analysis and Optimization*. Kluwer Academic Publishers, 1998.
- [RF95] M. Rim, Y. Fann, and R. Jain, "Global Scheduling with Code-Motions for High-Level Synthesis Applications," *IEEE Trans. on VLSI Systems*, vol. 3, pp. 379-392, Sep. 1995.
- [RG95] H. N. Rojas, R. W. Godby, and R. J. Needs, "Space-Time Method For Ab-Initio Calculations of Self-Energies and Dielectric Response Functions of Solids," *Phys. Rev. Lett.*, 74, pp. 1827, 1995.
- [SC00a] W.-T. Shiue and C. Chakrabarti, "Low Power Scheduling with Resources Operating at Multiple Voltages", *IEEE Transactions on Circuit and Systems Part II: Analog and Digital Signal Processing*, vol. 47, no. 6, pp. 536-543, June 2000.
- [SC00b] W.-T Shiue and C. Chakrabarti, "ILP-Based Scheme for Low Power Scheduling and Resource Binding," in *Proc. IEEE Intl. Symp. on Circuits and Systems*, pp. 279--282, May 2000.
- [SD02] A. M Sllame and V. Drabek, "An Efficient List-Based Scheduling Algorithm for High-Level-Synthesis," in *Proc. EUROMICRO Symp. on Digital System Design*, IEEE Computer Society, pp. 316-323, 2002.

- [SG97] H. Singh and D. D. Gajski, "A Design Methodology for Behavioral Level Power Exploration: Implementation and Experiments," University of California-Irvine, Technical Report ICS-TR-97-28, 1997.
- [Shi00] W.-T. Shiue, "High Level Synthesis for Peak Power Minimization Using ILP," in *Proc. IEEE Int'l Conf. on Application Specific Systems, Architectures and Processors*, pp. 103–112, 2000.
- [Shn97] Z. X. Shen and C. C. Jong, "Exploring Module Selection Space for Architectural Synthesis of Low-Power Designs," in *Proc. Int'l Symp. Circuits and Systems*, pp. 1532–1535, 1997.
- [SL98] J. Siddhiwala and C. Liang-Fang, "Scheduling Conditional Data-Flow Graphs with Resource Sharing," in *Proc. Great Lakes Symposium on VLSI*, Aug. 1998.
- [SM96] S. Singhai and K. McKinley, "Loop Fusion for Data Locality and Parallelism," in *Proc. Mid-Atlantic Student Workshop on Programming Languages and Systems*, Apr. 1996.
- [SL02] H.-J. Song, J.-J. Lee, G.-Y. Song, and I.-J. Hwang, "A Resource-Constrained Scheduling Algorithm for High-level Synthesis," in *Proc. Int'l Technical Conf. on Circuits/Systems, Computers and Communications (ITC-CSCC)*, pp. 1728-1731, Jul. 2002.
- [TH93] A. Timmer, M. Heijligers, L. Stok, and J. Jess, "Module Selection and Scheduling Using Unrestricted Libraries," in *Proc. European Design Automation Conf.*, pp. 547–551, 1993.
- [VA95] W. F. J. Verhaegh, E. H. L. Aarts, J. H. M. Korst, and P. E. R. Lippens, "Improved Force-Directed Scheduling in High-Throughput Digital Signal Processing," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 14, no. 8, pp. 945-960, Aug. 1995.
- [VG02] F. Vahid and T. D. Givargis, *Embedded system Design, A unified Hardware/Software Introduction*. John Wiley and Sons, 2002.
- [WJ04] L. Wang, Y. Jiang, and H. Selvaraj, "Synthesis Scheme for Low Power Designs with Multiple Supply Voltages by Heuristic Algorithms," in *Proc. ITCC*, pp. 826-829, 2004.
- [WP95] C.-Y. Wang and K. K. Parhi, "High-Level DSP Synthesis Using Concurrent Transformations, Scheduling, and Allocation," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 14, no. 3, pp. 274-295, Mar. 1995.
- [WT92] K. Wakabayashi and H. Tanaka, "Global Scheduling Independent of Control Dependencies Based on Condition Vectors," in *Proc. Design Automation Conf.*, pp. 112–115, 1992.

- [WY89] Z. Wakabayashi and T. Yoshimura, "A Resource Sharing and Control Synthesis Method for Conditional Branches," *Proc. of International Conference on Computer-Aided Design*, pp. 62-65, Nov. 1989.

VITA

Atef Khalil Allam is a native of Egypt. He received his Bachelor of Science and Master of Science in electrical engineering (Computers and Control), Assiut University, Egypt, in 1994 and 1998, respectively. He worked as a demonstrator from 1996 to 1998 and as an assistant lecturer in 1998 both with the Electrical Engineering Department, Assiut University. In the fall of 2001, he joined the graduate program in the Department of Electrical and Computer Engineering at Louisiana State University. He expects to receive the degree of Doctor of Philosophy in August 2005.