Operation Scheduling Algorithms for Power, Energy and Resource Minimization in High-Level Synthesis

BY

OUWEN SHI
B.S., Xidian University, Xi'an, China, 2013

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2017

Chicago, Illinois

Defense Committee:

Shantanu Dutt, Chair and Advisor
Amit Ranjan Trivedi
Zhao Zhang

ACKNOWLEDGMENTS

I would like to express my greatest gratitude to my advisor, Professor Shantanu Dutt, who tirelessly guides me in algorithm study and development throughout the four years. His desire, passion and dedication in research always inspires me to strive for the highest possible standards and goals. The achievements in the thesis would not be possible without his continuous support and guidance.

I would also like to express my appreciation to my dissertation committee, Professor Amit Ranjan Trivedi and Professor Zhao Zhang, for their valuable suggestions on my thesis.

Besides, I would appreciate my colleague, Xiuyan Zhang, who worked with me in the same lab. He always positively influenced me by his diligence, optimism and courage. His cross-disciplinary knowledge in algorithms also expanded my vision.

In addition, thanks to the faculty and staff in the Electrical and Computer Engineering Department. In particular, I would like to thank Tina Alvarado, Ala Wroblewski and Evelyn Reyes, who helped me with various administration issues.

Finally, deep thanks given to my parents. They are always my source of support, encouragement and strength.

OS

PREFACE

The research cooperation with Professor Shantanu Dutt during the four years' study at the University of Illinois at Chicago provided me with knowledge on state-of-the-art high-level synthesis algorithms and opportunities to improve algorithmic development and problem-solving skills. This thesis is the outcome of the study, which includes two published papers, one paper accepted for publication and two DAC (Design Automation Conference) poster presentations as listed below. The operation scheduling algorithm proposed in each of the papers optimizes a different design objective in high-level synthesis.

1. S. Dutt and O. Shi, "A Fast and Effective Lookahead and Fractional Search Based Scheduling Algorithm for High-Level Synthesis", accepted for publication in Proceedings of the Design Automation and Test in Europe (DATE), Dresden, Germany, Mar. 2018.
2. S. Dutt and O. Shi, "A Fast and Effective Fractional Search and Lookahead Based List Scheduling Algorithm for High-Level Synthesis", poster presentation at 54th Design Automation Conference (DAC), Austin, TX, June 2017.
3. S. Dutt and O. Shi, "Power-delay product based resource library construction for effective power optimization in HLS," published at 2017 International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, Mar. 2017, pp. 229-236.
4. O. Shi and S. Dutt, "Co-exploration of unit-time leakage power and latency spaces for leakage energy minimization in high-level synthesis," published in Journal of Low Power Electronics (JOLPE), Vol. 12, No. 4, Dec. 2016, pp. 295-308.

5. J. Xu, S. Dutt and O. Shi, “Power optimization in HLS Using Multi-Design Function Units”, poster presentation at 51th Design Automation Conference (DAC), San Francisco, CA, June 2014.

Contribution of Authors

Chapter 1 introduces the topic area of the thesis, categorizes problems in the area, reviews previous research works for each problem and highlighted the significance of our research contribution. Chapter 2 provides more technical background of our works and presents several classical scheduling algorithms, which our works are compared to. Chapter 3 represents the power-driven scheduling and module-selection algorithm PSA in [1] [2] and a series of unpublished experimental results in [2]. I played a major role in significantly modifying and debugging on initial version of the PSA program developed by Jian Xu. Xiuyan Zhang worked with me to collect the experimental results and worked with Prof. Shantanu Dutt for writing the report [2]. Chapter 4 represents a published manuscript [3] for which I am the second author, and it has contributions of PSA from [1] [2]. The main ideas and theoretical analysis are due to Prof. Shantanu Dutt, as is the major writing of the manuscript. I was responsible for the program as I did in Chapter 3 and worked with Prof. Shantanu Dutt to refine the manuscript. Jian Xu was involved in coding, experimentation and literature search in an earlier version of [3]. Chapter 5 represents a published manuscript [4] for which I am the first author. The major algorithmic ideas were proposed by Prof. Shantanu Dutt. I played a major role in debugging and implementation, suggested some improvements to the algorithms, as well as wrote an initial version of the manuscript that Prof. Shantanu Dutt then significantly re-wrote. Chapter 6 represents a poster presentation [5] and an accepted paper [6] for which I am the second author. Following the main algorithmic ideas and their theoretical results and analysis proposed by Prof. Shantanu Dutt, I implemented, experimented and provided some algorithmic improvements. I also wrote the initial manuscript that Prof. Shantanu Dutt refined.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF ABBREVIATIONS

ACO	Ant Colony Optimization
ALAP	As Late As Possible
ALU	Arithmetic Logic Unit
ASAP	As Soon As Possible
ASIC	Application-Specific Integrated Circuit
cc	Clock Cycle
CLA	Carry-Lookahead Adder
CSA	Carry-Select Adder
DFG	Data Flow Graph
DG	Distribution Graph
DP	Dynamic Power
DVFS	Dynamic Voltage and Frequency Scaling
EDG	Effective Distribution Graph
FA	Full Adder
FALLS	FrActional search and Lookahead based List Scheduling
FDS	Force-Directed Scheduling
FT	Function Type
FU	Functional Unit
GPM	Global Probability Map
HLS	High-Level Synthesis
IC	Integrated Circuit

LIST OF ABBREVIATIONS (Continued)

ILP	Integer Linear Programming
LE	Leakage Energy
LP	Leakage Power
LPR-GPS	Latency times unit Power minimization via RMS-driven Global Probability map based Scheduling
LS	List Scheduling
ME-LCS	Minimum-Energy Latency-Constrained Scheduling
ML-RCS	Minimum-Latency Resource-Constrained Scheduling
MP-LCS	Minimum-Power Latency-Constrained Scheduling
MR	Mobility Range
MR-LCS	Minimum-Resource Latency-Constrained Scheduling
PDG	Power Distribution Graph
PDP	Power-Delay-Product
PS	Predecessor/Successor
PSA	Power Simulated Annealing
RCA	Ripple-Carry Adder
RMS	Root-Mean-Square
RTL	Register-Transfer Level
SA	Simulated Annealing
UIC	University of Illinois at Chicago

SUMMARY

Power, energy and resource minimization subject to a latency constraint are important optimization objectives in operation scheduling in high-level synthesis. The research work presented herein aims to address each of the objective as follows.

First, we proposed that the degree of optimization achievable in high-level synthesis (HLS) designs with functional unit (FU) or module selection is significantly dependent on how the FUs in the resource library are parameterized. For power minimization, our proposal is that appreciably more power optimization is possible when:

- the FUs for each function type (FT) have a wide range of both power and delay metrics;
- their pair-wise power-delay product ratios are close to 1, say, in the range [0.8, 1.25], than when these criteria are not satisfied.

We showed that it is possible to achieve these parameter ranges for arithmetic FTs due to design variety and flexibility to hierarchically combine different design approaches. We also provided a probabilistic rationale for our hypotheses and further bolster it empirically by constructing different FU libraries that either meet or do not meet the above FU parameter criteria. Using a new power-driven simulated annealing (SA) based algorithm PSA, we consistently found that the power consumption of designs using libraries that meet our criteria are significantly lower than those that do not.

Then, we proposed a leakage energy (LE) minimization scheduling algorithm LPR-GPS. It co-explores unit-time leakage power (LP) and latency spaces in order to minimize their product. LPR-GPS extends the classical force-directed scheduling (FDS) by:

- an initial probabilistic distribution graph (DG) based on a non-uniform probability-driven randomized scheduling that yields the final starting scheduling probabilities that are conducive to LE minimization;
- a root-mean-square (RMS) based estimation of the maximum FU usage distributed across cc's that contributes to LE minimization;
- a fast and greedy noncommittal scheduling algorithm for estimating the latency by scheduling output operations first.

Experimental results show LPR-GPS reduces total LE by an average of 44% compared to the power-driven FDS and 12% compared to a version of LPR-GPS that only minimizes unit-LP.

Finally, we proposed an iterative list scheduling (LS) type algorithm FALLS to minimize the total number of FUs allocated, and thus the total area, in HLS designs. FALLS incorporates a novel lookahead technique to selectively schedule available non-0-slack operations by allocating the needed FUs earlier or reserving available FUs for scheduling more timing-urgent operations later, such that no additional FU is needed and a higher FU utilization is obtained. Further, a fractional search framework is developed to iteratively estimate the number of FUs of each FT required in the final design based on the current scheduling and FU utilization, and reiterate the lookahead-based list scheduling with the new FU allocation estimate to further increase FU utilization. Experimental results comparing FALLS with several state-of-the-art algorithms using a non-trivial FU library show an average 18.9% to 71.4% FU reduction while only has 5.5% optimality gap compared to an optimal integer linear programming (ILP) formulation. FALLS also performs much better in architectural area (FU + mux/demux + register area), interconnect

congestion and number of interconnects than state-of-the-art approximate algorithms, and is at most 4.0% worse in these metrics than the optimal ILP method.

1. INTRODUCTION

1.1. High-Level Synthesis

As the semiconductor technology node keeps moving towards the nanometer regime, and the transistor count on a chip significantly increases, it becomes more and more challenging to effectively and efficiently design modern integrated circuits (ICs). Thus, IC designers pay their attention to specifying the design in high-level languages like systemC, Verilog and VHDL, while relying on design tools to automatically transform the high-level design specification into the corresponding digital hardware implementation. The process executed by the tool is known as *high-level synthesis* (*HLS*) or *behavioral synthesis*.

HLS is crucial in the IC design flow, as the optimization applied at an earlier level is much more effective and efficient than a later one. For power optimization surveyed in [7], power saving opportunity at HLS is 8 to 12.5 times more than that at register-transfer level (RTL) and 20 times more than that at logic or physical level. Further, optimization effort at HLS is hundreds of times less than that at RTL and at least thousands of times less than that at logic and physical level.

1.2. Operation Scheduling

HLS tools schedule operations in the design specification to control steps or clock cycles (cc's), perform module selection to determine which speed type (see Section 1.3 for definition) functional unit (FU) to be used for each operation, allocate FUs and bind the operations to allocated FUs, to maximize or minimize an objective function subject to various design constraints. The first functionality, known as *operation scheduling*, is an important optimization problem to be solved, since the scheduling solution determines FU allocation and an approximate

binding, and hence plays a key role in the IC design flow. The operation scheduling problem is known to be NP-hard, and thus requires effective and efficient heuristic or stochastic algorithms to be developed.

1.3. Problem Category

Conventional operation scheduling problems in HLS are:

- Minimum-latency resource-constrained scheduling (ML-RCS): minimizing latency of the scheduling solution given resource constraints;
- Minimum-resource latency-constrained scheduling (MR-LCS): minimizing the total number of resources or a weighted sum of the number of resources of each type (functional type combined with one or more parameter values, like speed, power and area) used in the scheduling solution given a latency constraint.

In modern semiconductor industry, performance is no longer a main design objective. Instead, power and energy have become first-order design consideration for most computational devices. High power-consuming ICs deplete battery energy rapidly and cause reliability problems due to localized hot spots and phenomenon such as electromigration. Area is another consideration for some wearable and size-sensitive devices. All these make ML-RCS a less attractive problem.

Latency-constrained operation scheduling problems are therefore more interesting problems to be solved. MR-LCS is important since resource usage is strongly correlated to leakage energy (LE) and area. It also reduces interconnect complexity. However, although MR-LCS has been investigated for decades, current algorithms and methods cannot achieve good optimization quality and low runtime simultaneously. Besides, performance-constrained power

and energy minimization have become new problems to be solved. Thus, in addition to the well-known ML-RCS and MR-LCS problems, we define:

- Minimum-power latency-constrained scheduling (MP-LCS): minimizing the total power (sum of dynamic power and leakage power) of the scheduling solution given a latency constraint;
- Minimum-energy latency-constrained scheduling (ME-LCS): minimizing the total energy (sum of dynamic energy [DE] and leakage energy [LE]) of the scheduling solution given a latency constraint;

If a resource library has only one combination of delay, area, dynamic power (DP) and leakage power (LP) for each function type (FT) in the library, we call such library a *single-speed library*. Likewise, if there are more than one combination of delay, area, dynamic power and leakage for each FT in a resource library, we call such library a *multi-speed library*. If the input to ME-LCS is a single-speed library, the total dynamic power of the design will be a constant. Therefore, ME-LCS with a single-speed library is equivalent to minimize the total LE of the scheduling solution given a latency constraint. More details are presented in Section 5.1.

1.4. Previous Works

There has been a considerable amount of work on operation scheduling. In this section, we will review the previous works in different problem categories.

1.4.1. Power Minimization

There is already a rich body of power minimization algorithms for HLS at different design levels. Some typical and recent works for each major approach are briefly discussed. Module selection in terms of multi-V_{dd} and multi-V_{th} assignment of FUs has been extensively studied. In [8], a near-optimal dynamic power optimization algorithm by achieving maximum number of low-V_{dd} operations and minimum switching activity by solving a min-cost network

flow problem is proposed. Works in [9] [10] present heuristic approaches for incremental high-V_{th} reassignment for minimizing LP. Other popular techniques like bus binding, clock/power gating and dynamic voltage and frequency scaling use extra control logic to slowdown or turn off inactive FUs or registers to save power. The work in [11] performs optimal bus binding optimization and rescheduling simultaneously to reduce total bus switching activity. An ILP formulation aiming at low-power area-efficient clock gating is proposed in [12]. All the algorithms and techniques discussed above have some similarities: they either adopt a scheduling solution from any known scheduling algorithm as one of the inputs to their algorithm, or iteratively modify an initial scheduling solution to explore new solution space for a different optimization objective. Apparently, a good scheduling solution can help these algorithms to achieve higher optimization quality on their objectives of interest.

There are a few works that are motivated by the well-known force-directed scheduling (FDS) [13] [14]. The technique in [15] enumerates all resource sharing combinations and evaluates the switching activity associated with them, to reduce the total dynamic power. In [16], multi-V_{dd} assignment is used to reduce average and peak dynamic power across all cc's. The iterative scheduling process of FDS is maintained while a dynamic power force is used to evenly schedule operations to cc's and hence minimizes peak dynamic power. Besides, a post-rescheduling process to rebind operations from high-V_{dd} to low-V_{dd} FUs is proposed to minimize the average power.

1.4.2. Energy Minimization

A few research works minimize energy from different directions. Early scheduling algorithms tried to indirectly minimize energy by minimizing a correlated optimization objective. In [17], a FDS-type algorithm was developed to iteratively schedule an operation to a cc, which

is greedily guided by an energy function. The energy function is in fact balancing operations across cc's and thus is similar to FDS for minimizing FUs. Later, an ILP formulation with multi-V_{dd} was proposed in [18] that yields optimal energy solutions. However, it is impractical for large designs due to its exponential runtime complexity. Recent heuristic works also utilized multi-V_{dd} to minimize energy. A convex cost network flow model and a branch and bound method to assign frequency for each cc were proposed in [19].

To the best of our knowledge, and as indicated in a recent survey of low-power HLS [7], no previous scheduling algorithm with a single-speed library has addressed the issue of directly minimizing the total LE of a computation, or, equivalently, of the corresponding data-flow graph (DFG), which is the most important metric to minimize in systems that do not operate continuously.

1.4.3. FU Minimization

The ILP formulation proposed in [20] [21] provides optimal scheduling solutions for FU minimization, but it is impractical for large designs due to its exponential runtime complexity. FDS presented in [13] [14] schedules operations iteratively by choosing the best scheduling option that best balances the operation execution distribution across all cc's using the concept of minimum "force", and thereby minimizes the number of FUs required. The sub-optimality of FDS stems from its greedy and sequential scheduling option selection and a lack of lookahead. In addition, the high runtime complexity of $O(n^3)$ (n is the number of operations) motivates several refinements [22] [23]. The technique in [22] gradually reduces the time frame in which an operation can be scheduled by eliminating the current worst scheduling option. In addition, they modify the formulation of the spring constant in FDS forces which only consider the operation distribution in local cc's, to a global spring constant that considers the maximum operation

distribution in all cc's. In [23], an incremental force calculation was proposed by utilizing the fact that many operations have their time frame unchanged after an operation is scheduled. Some stochastic methods to solve the FU minimization problem have also been widely investigated. A simulated annealing (SA) approach was proposed in [24] and its move set guarantees that the complete solution space can be explored (i.e., the solution space graph is connected). In [25], the authors develop an ant-colony based algorithm to gradually approach a good solution by iteratively and probabilistically generating scheduling solutions based on which the scheduling probabilities are updated. In [26], the scheduling order of operations are determined by a genetic algorithm and then they can be scheduled by a constructive scheduling technique. All the above stochastic methods have high runtime for a good quality solution to be found, which prevents them to be effective for large problem sizes.

List scheduling (LS) is a classical algorithm for latency-constrained FU minimization. It schedules operations in as early cc's as possible if FUs are available, while greedily avoiding allocating new FUs unless it is mandatory for satisfying the latency constraint. Though the scheduling solution of LS is far from optimal due to many FUs allocated in intermediate and late cc's being sparsely utilized, its runtime complexity of $O(n \log n)$ is very scalable. Therefore, several research works like [26] [27] [28] use LS or LS-type algorithms as an iterative internal sub-routine to achieve good optimization quality for their objectives of interests. Lookahead in LS has been studied in scheduling problems in non-HLS fields [29] [30]. Early lookahead in instruction scheduling like [29] helps LS with a bad "precedence function" (different from the one we will discuss) to avoid failed scheduling by tentatively scheduling some or all unscheduled operations. Recent works like [30] in heterogeneous computing use a lookahead approach to

exhaustively evaluate all candidate resources that can execute the scheduled operation, and chooses the resource that is most likely to lead to the smallest estimated latency.

1.4.4. Resource Library Construction

To the best of our knowledge, there is no previous work aiming at multi-speed library construction. Further, we note here that previous module selection works used multi-speed libraries without paying attention to having appropriately parameterized FUs in them. For example, in [31] [32] [33] library characterizations are impractical or sub-optimal: the parameters of FUs in the library are not based on design realities and set artificially to meet the requirements of their designs. For example, the three adders considered in [31] have the same power-delay-products (PDPs), which is unrealistic in practice. The work in [32] also uses three adders but with PDP ratios of 1: 1.5: 1.33, which is somewhat sub-optimal as discussed later in the thesis (it is possible to design arithmetic FUs to have PDP ratios closer to but a little higher than 1). These simplifications neglect the importance of resource library construction that will be discussed later in the thesis.

1.5. Contributions

In this thesis, we propose the following three operation scheduling algorithms aim to three different optimization objectives. The algorithms and their advantages are as below:

- Our Power-driven Simulated Annealing (PSA) algorithm solves the MP-LCS problem given a multi-speed library:
  - The simulated annealing (SA) framework of PSA can be combined with any good scheduling algorithm for enhancing different optimization objectives.
  - Compared to the optimal integer linear programming (ILP) for MP-LCS, it has an average optimality gap of only 5.9% and is 193 times faster.

- Latency times unit Power minimization via our RMS-driven Global Probability map based Scheduling (LPR-GPS) algorithm that solves ME-LCS problem given a single-speed library:
  - ME-LCS has never been investigated in previous works.
  - Compared to two algorithms that only optimize the leakage power, LPR-GPS significantly reduces total LE by an average 37.05% and 12.41%, respectively, with reasonable runtime overhead.
- Our FrActional search and Lookahead based List Scheduling (FALLS) algorithm to solve MR-LCS problem given a single-speed library:
  - Compared to several state-of-the-art algorithms, FALLS reduces the total number of FUs allocated by [18.9%, 71.4%].
  - Compared to the optimal ILP for MR-LCS, it has an average optimality gap of only 5.5%.
  - FALLS is extremely fast: it is 278k times faster than ILP and yields good solutions for a DFG of 1300 operations in less than a second.

Besides the algorithms above, we proposed power-delay criteria for multi-speed library construction for better power optimization, in a direction that no previous work has targeted. Using PSA as an experimental platform, designs using the libraries that meet our criteria has significantly lower power consumption than those that do not, including when the former have fewer combinations of power and delay per FT (i.e., fewer speeds and hence a smaller solution space) than the latter.

1.6. Thesis Outline

The rest of the thesis is organized as follows. In Chapter 1, we formally formulate the operation scheduling problem and review several conventional scheduling algorithms to which our algorithms will be compared to. Our power minimization scheduling algorithms PSA [1] [2]

and its experimental results are presented in Chapter 3. We then present our library construction hypotheses proposed in [1] [3] for power optimization and provide a theoretical justification for them in Chapter 4. In the same chapter, we present library construction examples and use PSA as a platform to empirically prove our hypotheses. Our energy minimization scheduling algorithm LPR-GPS [4] and its experimental results are presented in Chapter 5. Our scheduling algorithm FALLS [5] [6] for FU minimization and its experimental results are presented in Chapter 6. Finally, we conclude in Chapter 7.

2. BACKGROUND

In this chapter, we first give the formal and general formulation of operation scheduling problems with single-speed library and multi-speed library. Then several conventional scheduling algorithms that frequently appear in literatures are reviewed and discussed. Our algorithms introduced in later chapters are motivated and based on the algorithms in this chapter. Our experiments will compare our algorithms to some of these algorithms.

2.1. General Problem Formulation

We consider the following general operation scheduling problem with the optimization objective unspecified. Given:

- An unscheduled DFG G (V, E), where V is the set of operations, and E is the set of arcs representing data dependencies between the operations;
- A legal upper-bound latency constraint L_c in number of cc's that is no smaller than the ASAP latency or the critical path delay;
- An FU library K that includes FU designs for each FT that appears in the operations in V. The library can be a single-speed library or a multi-speed library.

For any two operations u, $v \in V$ and a data dependency arc $(u, v) \in E$, if u and v are scheduled in cc's t_u and t_v, respectively, the *dependency constraint* is:

$$t_u + d_u \leq t_v \tag{2.1}$$

where $d_u \geqslant 1$ is the operation delay of u in number of cc's, u is the *predecessor* of v and v is the *successor* of u.

Our objective is to schedule each operation in V to a certain cc and choose an appropriate FU design to execute the operation if a multi-speed library is used. Meanwhile, the optimization

objective, *cost*, is minimized, the achieved latency $L \leq L_c$ and all dependency constraints are satisfied. The costs that we are interested in this thesis are power, energy and total number of FUs, which we will further define in later chapters.

2.2. As-Soon-As-Possible Scheduling

As-Soon-As-Possible (ASAP) scheduling is one of the two simplest scheduling algorithms in HLS. As indicated by its name, the algorithm schedules each operation in the DFG to the earliest cc in which the operation can possibly be scheduled. Thus, the solution yielded by ASAP scheduling has the minimum latency, called the *ASAP latency* or the *critical path delay*, but is very likely to unnecessarily allocate extra FUs. ASAP scheduling can be performed if only the operation delays are known.

The ASAP scheduling is formulated as follows. For any operation *u* that has no predecessors, i.e., *pred(u) = Ø*, its ASAP scheduled cc t_u^{ASAP} is simply:

$$t_u^{ASAP} = 1 \tag{2.2}$$

For *u* that has at least one predecessors, its ASAP scheduled cc t_u^{ASAP} is determined by:

$$t_u^{ASAP} = \max_{v \in pred(u)} (t_v^{ASAP} + d_v) \tag{2.3}$$

Let the number of operations in *V* be *n*. The algorithm analyzes all operations in *O(n)* time and in the worst case evaluates all predecessors of all operations in *O(n)* time. The time complexity is thus $O(n^2)$. However, each operation usually has limited predecessors that can be counted as a constant *c* that is far smaller than *n* in practice. Thus, ASAP scheduling is very fast. The pseudo code of ASAP scheduling is presented in Figure 1.

```
Algorithm ASAP(DFG G(V, E), a single-speed library K)
1.  While (there are unscheduled operations)
2.      If (the unscheduled operation u has no predecessor)
3.          Schedule u per Equation (2.2)
4.      Else if (the predecessors of u are all scheduled)
5.          Schedule u per Equation (2.3)
6.      End If
7.  End While
8.  Return the scheduling solution
```

Figure 1. The pseudo code of ASAP scheduling.

2.2.1. ASAP Time Updating

ASAP scheduling itself is not practically useful due to its excessive hardware usage. However, it provides important timing information for more advanced scheduling algorithms to make better scheduling decisions. For example, ASAP time provides the upper bound of the cc's in which an operation can possibly be scheduled in FDS and SA. In these algorithms, ASAP times often requires updating every time an operation is scheduled to provide the most up-to-date timing information. For example, let the ASAP time of operation u and v be 1 and 2, respectively, $\{u, v\} = V$, $(u, v) = E$ and $L_c = 4$. If u is scheduled in cc 2 and v is still unscheduled, the updated ASAP time of u and v is 2 and 3, respectively, since the ASAP time of u is equal to its scheduled cc and v can no longer be scheduled in cc 2. The ASAP time updating can be done by reperforming the entire ASAP scheduling with the timing of scheduled operations updated in Figure 1, but also can be done by an updating strategy that only explores the operations that are affected.

The ASAP time updating algorithm recursively explores the predecessors of current operation, which is the newly scheduled operation initially. For each current predecessor being explored, all its successors are examined: if any of them results in an updating of the current predecessor's ASAP time, the predecessors of the current predecessor are examined in the next

recursion. The recursion stops when no further predecessors' ASAP time need to be examined. In fact, the algorithm uses depth-first search to traverse the partial DFG that is only affected by the newly scheduled operation, and hence has a complexity of $O(n)$ which is much faster than reperforming ASAP scheduling. The pseudo code of the ASAP time updating algorithm is presented in Figure 2.

```
Algorithm updateASAP(newly scheduled operation u)
1.  If (this is not the initial function call)
2.      For (each predecessor v of u)
3.          If (t_v^ASAP + d_v > t_u^ASAP)
4.              t_u^ASAP = t_v^ASAP + d_v
5.          End if
6.      End for
7.      If (t_u^ASAP is not updated in the for-loop above)
8.          Return
9.      End if
10. End if
11. For (each successor w of u)
12.     If (w is not scheduled)
13.         updateASAP(w);
14.     End if
15. End for
16. Return
```

Figure 2. The pseudo code of ASAP time updating algorithm.

2.3. As-Late-As-Possible Scheduling

As-Late-As-Possible (ALAP) scheduling is the other simplest scheduling algorithms in HLS and is symmetry to ASAP scheduling. As indicated by its name, the algorithm schedules each operation in the DFG to the latest cc in which the operation can possibly be scheduled. Thus, the solution yielded by ALAP scheduling has the maximum latency that is equal to the latency constraint, but is also very likely to unnecessarily allocate extra FUs. ALAP scheduling can be performed if only the operation delays are known.

Given a legal latency constraint L_c that is no smaller than the ASAP latency, the ALAP scheduling is formulated as follows. For any operation *u* that has no successor, i.e., *succ(u)* = Ø, its ALAP scheduled cc t_u^{ALAP} is simply:

$$t_u^{ASAP} = L_c - d_u + 1 \tag{2.4}$$

For *u* that has at least one successor, its ALAP scheduled cc t_u^{ALAP} is determined by:

$$t_u^{ALAP} = \min_{v \in succ(u)} (t_v^{ALAP}) - d_u \tag{2.5}$$

Similar to ASAP scheduling, the algorithm also has a time complexity $O(n^2)$ and is very fast in practice. The pseudo code of ALAP scheduling is presented in Figure 3.

Algorithm ALAP(DFG *G(V, E)*, a single-speed library *K*, a legal latency constraint L_c)
1. **While** (there are unscheduled operations)
2. **If** (the unscheduled operation *u* has no successor)
3. Schedule *u* per Equation (2.4)
4. **Else if** (the successor of *u* are all scheduled)
5. Schedule *u* per Equation (2.5)
6. **End If**
7. **End While**
8. **Return** the scheduling solution

Figure 3. The pseudo code of ALAP scheduling.

2.3.1. ALAP Time Updating

Similar to ASAP scheduling, ALAP scheduling itself is not practically useful but provides important timing information for more advanced scheduling algorithms to make better scheduling decisions. For example, ALAP time provides the lower bound of the cc's in which an operation can possibly be scheduled in FDS and SA; it also provides timing-urgency of available operations in LS. In some of these algorithms, ALAP times often requires updating every time an operation is scheduled to provide the most up-to-date timing information. For example, let the

ALAP time of operation *u* and *v* be 3 and 4, respectively, $\{u, v\} = V$, $(u, v) = E$ and $L_c = 4$. If *v* is scheduled in cc 3 and *u* is still unscheduled, the updated ALAP time of *u* and *v* is 2 and 3, respectively, since the ALAP time of *v* is equal to its scheduled cc and *u* can no longer be scheduled in cc 3. Symmetry to ASAP time, the ALAP time updating can be done by reperforming the entire ALAP scheduling with the timing of scheduled operations updated in Figure 3, but also can be done by an updating strategy that only explores the operations that are affected as presented in Figure 4.

Algorithm updateALAP(newly scheduled operation *u*)
1. **If** (this is not the initial function call)
2. **For** (each successor *v* of *u*)
3. **If** ($t_v^{ALAP} - d_u < t_u^{ALAP}$)
4. $t_u^{ALAP} = t_v^{ALAP} - d_u$
5. **End if**
6. **End for**
7. **If (**t_u^{ALAP} is not updated in the for-loop above)
8. **Return**
9. **End if**
10. **End if**
11. **For** (each predecessor *w* of *u*)
12. **If** (*w* is not scheduled)
13. updateALAP(*w*);
14. **End if**
15. **End for**
16. **Return**

Figure 4. The pseudo code of ALAP time updating algorithm.

2.4. List Scheduling

Here we discuss the classical latency-constrained list scheduling (LS) algorithm for MR-LCS problem using a single-speed library. In each cc, LS always schedules the most timing-urgent operations to available FUs. Starting with a minimum FU allocation, a new FU is only allocated when there is an available operation that needs to be scheduled immediately to satisfy

the latency constraint, but there is no allocated FU of that FT currently available due to being busy executing other operations. By only allocating new FUs when it is mandatory, LS was expected to come close to minimizing the number of allocated FUs in the final scheduling solution. However, LS fails to achieve this goal due to the low FU utilization mentioned later in the section.

The pseudo code of LS is presented in Figure 5. Initially, only one FU per FT is allocated. The ALAP time t^{ALAP} is computed for each operation. Then in each cc t in chronological order, for each FT k, an available unscheduled operation set $U_{t,k} \in V$, which includes all unscheduled operations of FT k whose predecessors have all finished execution, is determined. The slack s_u of each operation u in $U_{t,k}$ is then computed as:

$$s_u = t_u^L - t \;\; u \in U_{t,k} \tag{2.6}$$

If $s_u = 0$, u is *0-slack* and must be scheduled in cc t, i.e., one additional FU needs to be allocated if all FUs of FT k are busy executing other operations. The other operations in $U_{t,k}$ are *non-0-slack*. If there are still available FUs after all 0-slack operations are scheduled, the non-0-slack operations are scheduled in t and bound to the available FUs in slack-increasing order. This slack-based scheduling process iterates for each t, until all operations are scheduled.

We term the FU allocation vector $\boldsymbol{r}$ (one element per FT) before any operation is scheduled as pre-allocation; it is only one FU per FT in LS; Similarly, the FU allocation vector $\boldsymbol{r}$ after all operations are scheduled is termed as post-allocation. In practice, the number of FUs in post-allocation is significantly more than that in pre-allocation in the solutions of LS, indicating many FUs are allocated in intermediate cc's. This results in the FUs allocated in later cc's being sparsely utilized. Due to the insufficient FU utilization, excessive FUs are likely to be allocated in the solutions of LS.

Algorithm LS(DFG $G(V, E)$, a single-speed library K, a legal latency constraint L_c)
1. $\boldsymbol{r} = (1, 1, ..., 1)$, $t = 1$ //allocate one FU per FT before scheduling
2. Compute the ALAP times t^{ALAP} per L_c
3. **While** (there are unscheduled operations)
4. **For** (each FT k)
5. Determine the available unscheduled operation set $U_{t, k}$
6. Compute slack s_u for all $u \in U_{t, k}$ by Equation (2.6)
7. Schedule 0-slack operations in $U_{t, k}$ to t, allocate new FUs if needed, update r_k if new FUs are allocated
8. Schedule non-0-slack operations in $U_{t, k}$ to t in slack-increasing order and bind them to remaining available FUs
9. **End For**
10. $t = t + 1$
11. **End while**
12. **Return** the scheduling solution

Figure 5. The pseudo code of list scheduling.

2.4.1. Complexity

The time complexity of LS is $\Theta(n \log n)$, since each sorting or searching operation in a balanced binary search tree based on ALAP times (equivalent to slack) takes $\Theta(\log n)$ time, and the total number of searches = total number of available FUs across all clock cycles = $\Theta(n)$.

2.5. Force-Directed Scheduling

Force-directed scheduling (FDS) introduced in [13] [14] also target on MR-LCS problem with a single-speed library. As discussed earlier, FDS reduces the number of FUs in the design by evenly scheduling operations in all cc's to balance the FU distribution. In each scheduling iteration, FDS evaluates all *scheduling options*, the unscheduled operations and the candidate cc's where they can possibly be scheduled, and chooses the scheduling option to make scheduling that is probabilistically expected to minimize the sum of number of FUs allocated across all FTs. The fact, however, is that, conceptually and analytically speaking, FDS has some weakness in achieving this. The steps of FDS are summarized as follows and the pseudo code is presented in Figure 6.

2.5.1. Scheduling Probabilities Determination

First, the ASAP and ALAP times of all operations are determined by ASAP and ALAP scheduling, respectively. Then the *mobility* μ_u, the number of cc's in which an operation *u* can be scheduled, is:

$$\mu_u = t_u^{ALAP} - t_u^{ASAP} + 1 \tag{2.7}$$

and the *mobility range* (MR) of *u* MR_u is $[t_u^{ASAP}, t_u^{ALAP}]$. The uniform scheduling probability $p_u(i)$ of *u* in cc *i* is:

$$p_u(i) = \begin{cases} \frac{1}{\mu_u} & i \in MR_u \\ 0 & otherwise \end{cases} \tag{2.8}$$

Note that at any intermediate stage during the scheduling process, i.e., after some operations have been scheduled, the t_u^{ASAP} and t_u^{ASAP} of an unscheduled operation *u* are updated by substituting the scheduled cc t_v of each scheduled predecessor and successor *v*, if any, in place of t_v^{ASAP} and t_v^{ALAP} in Equations (2.2) to (2.5), respectively.

2.5.2. Distribution Graph Construction

For each FT, a distribution graph (DG) provides the expected FU usage in each cc. It is constructed by summing up all scheduling probabilities of operations of the same FT in each cc. The distribution value *DG(k, i)* of FT *k* in cc *i* is computed as:

$$DG(k,i) = \sum_{u \in V(k)} p_u(i) \tag{2.9}$$

where *V(k)* is the set of operations of FT *k*. Note that *DG(k, i)* is the expected number of operations and thus FUs of FT *k* executing in cc *i*. Also, $DG(k) = \{DG(k, i): 1 \le i \le L_c\}$ is the DG of FT *k*, and *DG = {DG(k): k i*s a FT in the DFG*}* is the overall DG for the given DFG. It should be noted that the largest distribution value $DG_{max}(k)$ gives a probabilistic estimate of the number of FUs of FT *k* to be allocated in the design. The value is more accurate with more operations

scheduled and finally becomes deterministic after all operations are scheduled. The goal of FDS is to minimize

$$\sum_{k \in K} DG_{max}(k) \tag{2.10}$$

after all operations are scheduled. In every scheduling iteration, in which an operation *u* is scheduled, all affected *DG(k, i)*'s are updated to account for the zeroing of scheduling probabilities of *u* in cc's other than the one in which it has been scheduled or in general will be executing (in which this probability is 1), the shrinking of the MRs of some predecessors and successors of *u* and thus changes in their corresponding scheduling probabilities in the cc's of their original MR's. At any stage of the scheduling process, the DG provides a snapshot of scheduling probabilities across the operation and cc spaces, and thus the expected number of operations and FUs of each FT *k* in each cc. This also means that for each scheduling decision, we can estimate the global metric of interest, like Equation (2.10).

2.5.3. Self-Force Formulation

A self-force is calculated for each scheduling option. The self-force $SF_u(i)$ for scheduling operation *u* of FT *k* in cc *i* is defined as:

$$SF_u(i) = \sum_{x \in MR_u} \sum_{y \in DR_u(x)} DG(k, y) \times \Delta p_u(x) \tag{2.11}$$

where $DR_u(x)$ is the delay or execution range $[x, x + d_u - 1]$ of *u* scheduled in cc *x*, and

$$\Delta p_u(x) = \begin{cases} 1 - p_u(x) & x = i \\ -p_u(x) & x \in MR_u - \{i\} \end{cases} \tag{2.12}$$

The self-force formulation assigns high weights to the probability changes $\Delta p_u(x)$*'s* in cc's that have high DG values, indicating scheduling in such cc's increases the risk to have an unbalanced FU distribution. On the other hand, if an operation is scheduled in a cc that has a low DG value, the weighted positive probability increment is comparably small, and the weighted negative

probability decrement in other cc's in *u*'s MR are comparably large, reflecting a good choice to flatten the DG and reduce the high DG values. Thus, a scheduling option with the smallest self-force is expected to minimize the maximum DG value, and hence the number of FUs of *u*'s FT.

2.5.4. Predecessor/Successor and Total Force Formulation

A predecessor/successor (PS) force is calculated for each scheduling option in a way similar to the self-force. Scheduling an operation is very likely to shrink the MR of its unscheduled predecessors and successors (by potentially decreasing their ALAP times or increasing their ASAP times, respectively), resulting in fewer opportunities to schedule them in cc's of low DG values at later scheduling iterations. Such an effect should also be considered by an overall force formulation. The PS-force $PSF_u(i)$ for scheduling operation *u* in cc *i* is defined as:

$$PSF_u(i) = \sum_{v \in PS_u} \sum_{x \in MR_v} \sum_{y \in DR_v(x)} DG(type(v), y) \times \Delta p_v(x) \qquad (2.13)$$

where $PS_u = pred(u) \cup succ(u)$ and $type(v)$ is the FT of a PS operation *v*, and

$$\Delta p_v(x) = p'_v(x) - p_v(x) \qquad (2.14)$$

where $p'_v(x)$ is the tentatively updated scheduling probability in cc *x* of the PS operation *v* for evaluating the scheduling option and $p_v(x)$ is *v*'s current scheduling probability in cc *x*. For example, suppose operation *v*, a predecessor of operation *u*, has a MR of 3 cc's, i_1, i_2 and i_3. According to Equation (2.8), the current scheduling probabilities $p_v(i_1) = p_v(i_2) = p_v(i_3) = 1/3$. If *u* is tentatively scheduled and hence *v*'s MR temporarily shrinks to 2 cc's, i_1 and i_2, due to a decrease in *v*'s ALAP time. The tentatively updated MR of *v* leads to a tentatively updated scheduling probability $p'_v(i_1) = p'_v(i_2) = 1/2$ and $p'_v(i_3) = 0$. Similar to the self-force, the PS-force of a scheduling option measures its probabilistic effect on the sum of DG values of all FTs among its predecessors and successors; the smaller the PS force, the smaller the sum of weighted

probability changes of the local (within the MRs of *u*'s predecessors and successors) DG values among all related FTs.

Finally, the total force $F_u(i)$ to evaluate the scheduling of operation *u* in cc *i* is given as:

$$F_u(i) = SF_u(i) + PSF_u(i) \quad (2.15)$$

Algorithm FDS(DFG *G(V, E)*, a single-speed library *K*, a legal latency constraint L_c)
1. **While** (there are unscheduled operations)
2. Update ASAP and ALAP times of unscheduled operations
3. Update the distribution graph per Equation (2.9) based on the updated ASAP and ALAP times and the scheduled cc's of scheduled operations
4. **For** (each unscheduled operation *u*)
5. **For** (each cc *i* in MR_u)
6. Calculate self-force, PS-force and total force per Equations (2.11) to (2.15)
7. **End For**
8. **End For**
9. Pick the scheduling option with the minimum total force to schedule
10. **End while**
11. **Return** the scheduling solution

Figure 6. The pseudo code of force-directed scheduling.

2.5.5. Complexity and Optimality Analysis

The time complexity of FDS is analyzed as follows. FDS schedules one operation per iteration, therefore there are $\Theta(n)$ scheduling iterations. In each scheduling iterations, there are $O(n)$ unscheduled operations to be evaluated: the worst case happens in the first iteration when all operations are unscheduled. Finally, there are $O(n)$ PS operations that need to consider in the PS-force. The nested triple *n*'s results in a total time complexity of $O(n^3)$.

As suggested by Equation (2.10), the objective of FDS is minimizing the sum of maximum distribution values of all FTs, a min-max goal. The problem with the FDS force formulation is that it essentially captures the difference between the DG value of the to-be-scheduled cc and the average DG value of all other cc's in the MR of each unscheduled

operation being evaluated, while any estimate of the increase or decrease in the maximum DG value(s) is not accounted for unless the maximum DG value lies in the MR of an operation. Further, across the forces of all scheduling options, the least force generally does not correspond to the largest decrease or smallest increase of the maximum DG value due to the aforementioned property of what the force measures. Thus, the scheduling decisions are actually not made based on this most important consideration, and hence FDS cannot guarantee a good approximation of the min–max goal.

To frame the above discussion more analytically, the self-force expression of Equation (2.11) for scheduling operation u in cc i, assuming uniform probabilities and a single cc delay, can be re-formulated as:

$$SF_u(i) = DG(k,i) - \sum_{x \in MR_u} DG(k,x) \times p_u(x)$$

$$= DG(k,i) - \frac{1}{|MR_u|} \sum_{x \in MR_u} DG(k,x) \qquad (2.16)$$

This shows that the force formulation of FDS yields the lowest force among all scheduling possibilities to be the one that corresponds to the largest difference between the average DG value in an operation's MR and the DG value in its scheduling cc. Clearly, this has a weak correlation to reducing the current maximum DG value. Therefore, the scheduling option evaluation in FDS is significantly flawed.

2.6. Simulated Annealing

Simulated annealing (SA) is a probabilistic method for approximating the global minimum of a cost function. It is often used when the solution space is discrete. Various SA formulations are thus proposed for operation scheduling problem. Many optimization objectives

in operation scheduling can be solved by SA with the corresponding cost functions. We will introduce the SA formulation for general latency-constrained scheduling with a single-speed library in this section. Based on [24] [25], our literature review and experiments, we believe this SA formulation provides the best scheduling solutions with a reasonable runtime.

The main idea of SA is that it considers some neighboring state *P'* of the current state *P*, and probabilistically determines whether to move the system to the new state *P*' or stay in state *P*. These moves ultimately lead the system to a state of near-optimal cost. The steps are summarized as follows:

1. Generate an initial state and configure the initial temperature T_0 and the freezing temperature T_f.
2. Generate a new state in the neighborhood of the current state.
3. Change to the new state with a probability $min\{1, e^{-\frac{c_{new}-c}{T}}\}$, where c_{new} (c) is the cost of the new (current) solution and *T* is the current temperature.
4. Repeat steps 2 and 3 until the cost of new solutions does not change much, then decrease *T* base on a cooling scheme.
5. Repeat steps 2 to 4 until $T \leq T_f$.

There are several performance-related configurations that can be applied to SA. Each of the configurations that we choose is detailed in the following subsections. The pseudo code of our SA is presented in Figure 7.

2.6.1. Initial Solution

The initial solution of SA determines the optimization starting point. It is supposed to be generated fast and has a not-too-bad cost. As discussed in [25], ASAP and ALAP scheduling do not yield good initial solutions since they tend to cluster operations in earlier or later cc's, which

is bad for minimizing resource usage. Further, ALAP scheduling is even worse in energy minimization since the latency of the solution is not minimized and it also makes SA moves harder to change the initial solution towards a low-cost direction. LS also suffers from the same limitation of ASAP and ALAP scheduling. FDS is too slow for large DFGs due to its high time complexity. Therefore, we construct an initial solution by randomly picking an operation, scheduling it to a random cc to its MR, updating the MRs of unscheduled operations and iterating the random scheduling process until all operations are scheduled. The runtime complexity of the randomized scheduling process is $O(n^2)$, since there are n iterations and at most n options (in the first iteration) for the randomization algorithm to choose.

2.6.2. Temperature

Similar to the SA introduced in [25], we use the well-known geometric cooling scheme that is also used in [34]. The temperature decreasing factor is set to 0.9. When the current temperature reaches the freezing temperature, the best solution is reported.

However, it is not clear how the initial and freezing temperatures are determined in [25]. We adopt the idea in [35] that is also used in [24]: set the initial temperature so that the statistical probability to accept a cost-increasing move is high (e.g., 80%), and set the freezing temperature so that the statistical probability to accept such a move is very low (e.g., 1%). To be specific, before the annealing process, we generate a number of moves (e.g., 2n) to the initial solution and get the average cost of new solutions that have increased costs. We can then set the initial and freezing temperatures based on the average cost and the Boltzmann probability $e^{-\frac{c_{new}-c}{T}}$ for accepting cost-increasing moves. This allows the SA to fully explore the global solution space at the beginning and the local minimum at the end.

2.6.3. Local Search and Move Set

The local search is to generate a number (e.g., 2*n* as in [25]) of successful moves in the current temperature.

Though the move sets in [24] [25] can guarantee that the entire solution space can be reached by the moves, every move makes only a little modification (e.g., reschedule an operation to the neighboring later cc in its MR and push the scheduled cc's of affected successors to their neighboring later cc's) the current solution. This is hard for SA to explore the solution space that is very different from the current solution and hence is less likely to fully explore the solution space for searching the global minimum. Further, the moves in [25] are very likely to fail since a move would be rejected if rescheduling an operation would violate the dependency constraint in its predecessor or successors. Checking dependency constraints for this situation is also time-consuming.

Our move set has 3 moves: up-move, down-move and random-move. The first two moves cover all four moves in [24], and the last move enables broader and more efficient solution space searching. Each of the moves are defined as follows:

- Up-move: Randomly choose an operation to reschedule it to a neighboring earlier cc in its MR and reschedule affected predecessors to their neighboring earlier cc's. The move fails if the randomly chosen operation was scheduled in the earliest cc in its MR.
- Down-move: Randomly choose an operation to reschedule it to a neighboring later cc in its MR and reschedule affected successors to their neighboring later cc's. The move fails if the randomly chosen operation was scheduled in the latest cc in its MR;
- Random-move: Randomly choose an operation and randomly reschedule it to another cc in its MR. The move fails if the randomly chosen operation has a MR = 1.

Algorithm SA(DFG *G(V, E)*, a single-speed library *K*, a legal latency constraint L_c)
1. ASAP and ALAP scheduling to determine initial MRs
2. Generate an initial scheduling solution *S* by randomization
3. Adaptively calculate the initial temperature $t = t_0$ and the freezing temperature t_e
4. **While** ($t > t_e$)
5. **For** (*i* from 0 to *2n*)
6. Randomly generate a successful move and hence a new solution S_n
7. $S = S_n$ according to the probability $min\left\{1, e^{-\frac{S_n.cost - S.cost}{T}}\right\}$
8. $i = i + 1$
9. **End for**
10. $t = 0.9t$
11. **End While**
12. **Return** the scheduling solution

Figure 7. The pseudo code of simulated annealing.

2.7. Integer Linear Programming

The integer linear programming (ILP) formulation proposed in [20] [21] yields optimal solutions for MR-LCS problems with a single-speed library. Although it has a very high runtime compared to other heuristic and stochastic algorithms and is not applicable to large problems due to excessive memory usage, we can compare the solutions of our algorithms to ILP solutions to see how close we are from the optimum. With minor algorithmic improvement, we will briefly introduce and summarize the ILP formulation in [20] [21] that is used in our experiments.

The objective function to be minimized is:

$$\sum_{all\ k \in K} c_k \times M_k \tag{2.17}$$

where c_k is the cost of FT k and M_k is the number of FUs of FT k allocated in the scheduling solution. For MR-LCS problems with a single-speed library, the cost of each FT is simply 1. The weighted sum can be applied if area minimization is desired, where the cost of each FT is the area of FUs of the FT.

There are a number of constraints that must be satisfied as follows:

- One operation can only be scheduled in one cc of its MR:

$$\sum_{i \in MR_u} t_{u,i} = 1, for\ all\ 1 \le i \le n \tag{2.18}$$

where $t_{u,i}$ is a 0/1 integer variable associated with operation u: $t_{u,i}$ = 1 if u is scheduled in cc i in its MR; $t_{u,i}$ = 0 otherwise.

- Dependency constraints:

$$\sum_{i \in MR_v} i \times t_{v,i} - \sum_{i \in MR_u} i \times t_{u,i} \ge d_u, for\ all\ (u, v) \tag{2.19}$$

where d_u is the delay of operation u.

- The number of operations of the same FT executed in the same cc can never exceed the number of allocated FUs of the FT:

$$\sum_{u \in each\ operation\ of\ FT\ k} t_{u,i} \le M_k, for\ all\ 1 \le i \le L_c, k \in K \tag{2.20}$$

Using this ILP formulation, the solution returned by any ILP solver like IBM CPLEX includes the lowest/optimal cost and the values of all $t_{u,i}$'s, which is the scheduling solution. The number of $t_{u,i}$'s in [20] [21] is $n \times L_c$. In our implementation, we decreased this number to $\sum_{u \in V} MR_u$, that is, we only define $t_{u,i}$'s in the MR of each operation in V. Meanwhile, we have to modified Equation (2.20) to:

$$\sum_{u \in each\ operation\ of\ FT\ k\ and\ i \in MR_u} t_{u,i} \le M_k, for\ all\ 1 \le i \le L_c, k \in K \tag{2.21}$$

This modification significantly reduces the number of variables and the complexity of constraints in the ILP formulation and hence improves its efficiency. However, it is still much slower than other heuristic and stochastic algorithms and fails to tackle larger input DFGs due to excessive memory usage that will be shown in our experiments.

3. PSA: NEW POWER MINIMIZATION ALGORITHM

The work in this chapter has been presented in [1] [2].

In this chapter, we propose a SA-based module selection scheduling algorithm power simulated annealing (PSA) for total power minimization (the MP-LCS problem) with a multi-speed library. The algorithm is built from a SA-based hierarchical framework that effectively and rapidly explores the speed options of each operation, and an internal fast scheduler that is modified from the well-known list scheduling algorithm. PSA has the following properties:

- Stochastic- and constructive-heuristics in PSA have complementary characteristics. The stochastic-heuristic, represented by the SA-based hierarchical framework, obtains the speed of each operation (module selection) as the partial solution. Then, the constructive-heuristics, represented by the modified list scheduling, efficiently generates reasonably good complete solutions to evaluate the partial solutions of the stochastic-heuristic. The evaluation results are then fed to the stochastic-heuristic for further solution space exploration. The combination of both yields good solutions while keeping the runtime low.
- The constructive-heuristic in PSA is very flexible: it can be any scheduler that can handle scheduling operations with known speeds (power and delay characteristics). This makes PSA easier to adapt to different optimization objectives and provides more options in tradeoff between optimization effort and runtime.

The experimental results show that PSA provides an average of 8.8% leakage power improvement over the state-of-the-art approximate algorithms. Further, compared to an optimal 0/1-ILP formulation for total (dynamic and leakage) power optimization under latency

constraints, it has an average optimality gap of only 5.9%, and offers a significant runtime advantage by a factor of 193.

3.1. Power Model

To evaluate a scheduling solution for the objective of optimizing the total power, the DP and LP should be determined respectively. We define a configuration $c(u_i)$ of an operation u_i as $c(i) = (k_i, s_i)$, where $1 \leq i \leq n$ (n is the number of operations in the input DFG), k_i is the FT of u_i and s_i is the speed of u_i. For example, for a triple-speed FU library, s_i is defined as:

$$s_i = \begin{cases} 1 & if\ u_i\ is\ bound\ to\ the\ slowest\ FU \\ 2 & if\ u_i\ is\ bound\ to\ the\ faster\ FU \\ 3 & if\ u_i\ is\ bound\ to\ the\ fastest\ FU \end{cases} \quad (4.1)$$

Further, we define the speeds of all operations of a DFG as a speed vector $SV = [s_1, s_2, \ldots, s_n]$. The dynamic energy of the DFG is then determined as:

$$E_D = \sum_{i=1}^{n} dp\big(c(u_i)\big) \times d\big(c(u_i)\big) \quad (4.2)$$

where $dp(c(u_i))$ and $d(c(u_i))$ is the dynamic energy consumed per cc and the delay of u_i which is bound to an FU of configuration $c(u_i)$, respectively. Note that the dynamic energy is irrespective to the scheduling solution.

Different from dynamic energy, the leakage energy of the DFG is correlated to the scheduling solution. Given a scheduling solution, the leakage energy of the solution is:

$$E_L = \sum_{i \in all\ FTs} \sum_{j \in all\ speeds\ of\ FT\ i} lp(i,j) \times num(i,j) \times L \quad (4.3)$$

where $lp(i, j)$ is the leakage energy consumed per cc of FUs of FT i and speed j, $num(i, j)$ is the number of FUs of FT i and speed j and L is the achieved latency of the solution.

The total energy consumption is simply the sum of E_D and E_L. For power minimization, the transformation from total energy to total power P_T is:

$$P_T = \frac{E_D + E_L}{L_c} \tag{4.4}$$

By using this power model, total power minimization is equivalent to energy minimization, since the latency constraint is a constant.

3.2. Simulated Annealing Framework

The SA framework in PSA has the same good temperature and local search scheme to the basic SA algorithm introduced in Section 2.6. However, the initial solution generation, the local search including the move set and the post-move processing are different due to the difference between the problems.

3.2.1. Initial Solution

Different from the SA in Section 2.6, the added dimension of speed for each operation makes generating initial solutions randomly more difficult. We thus need to determine the speed of each operation first, since it is important to make every individual solution generation fast, and separating the speed determination and the scheduling process is a key for this purpose.

There are 3 initial speed configurations: all operations are at the slowest speed (all executed by the slowest FUs), at the fastest speed and at random speeds. Based on our experiments, we choose the first configuration: the slowest FUs usually have the lowest overall power consumption, since the PDP ratios of FUs of faster speeds are practically hard to below 1.

After making the initial speed configuration, we can then generate the initial solution for the SA framework of PSA by using the modified list scheduling to schedule all operations. As will be shown in Section 3.3, the scheduler has very similar or even lower runtime than the randomized scheduling. Further, its solution quality is reasonably good and more consistent: the randomized scheduling sometimes generates very sub-optimal solutions, and may hamper the subsequent solution space exploration and the quality of the final scheduling solution.

3.2.2. Move Set

The moves in PSA only work on the speed vector, the module selection part of the solution. Based on the speed vector, the internal scheduler determines the scheduling and the complete solution. There are four types of moves in PSA as follows:

- Move 1: Randomly choose an FU is not the fastest speed, then increase the operations that are bound to the FUs to a neighboring faster speed;
- Move 2: Randomly choose an FU is not the slowest speed, then increase the operations that are bound to the FUs to a neighboring slower speed;
- Move 3: Randomly choose two FUs that are of the same FT but not of the same speed, randomly choose the same number of operations from the two FUs, then exchange the speeds between the operations of the two FUs;
- Move 4: Randomly choose a starting point m ($1 \leq m \leq n$, where n is the number of operations) and an endpoint m' ($m \leq m' \leq n$), then randomly reconfigure the speed of each operations in the range [m, m'].

For example, let a dual speed vector SV_{dual} = [0, 1, 0, 0, 1] with operation 1, 3, 4 binding to the slow FU and operation 2, 5 binding to the fast FU. Assuming all operations are of the same FT. The new speed vector SV'_{dual} after Move 1 and 2 is [**1**, 1, **1**, **1**, 1] and [0, **0**, 0, 0, **0**], respectively. There are several possible speed vectors after applying Move 3: if only one operation in each of the FUs are chosen, a possible speed vector SV'_{dual} is [**1**, **0**, 0, 0, 1], where operation 1 and 2 are chosen in the slow and fast FU, respectively; if two operations in each of the FUs are chosen, a possible speed vector SV'_{dual} is [**1**, **0**, **1**, 0, **0**], where operation 1, 3 and operation 2, 5 are chosen in the slow and fast FU, respectively. Similarly, a possible speed vector SV'_{dual} is [0, **1**, **1**, **1**, 1] after applying Move 4, where m = 2 and m' = 4.

Among all the moves, the first three make FU-centric speed modifications to the speed vector and generally lead to a solution with a small power change. Move 4, on the other side, is not FU-centric and may lead to large "jumps" in the solution space. The apparent randomness of the distribution of a subsequence of operations among different FUs is an interesting point for Move 4. This randomness can make Move 4 to generate significant gaps (either better or worse) from previous solutions. Based on Move 4, it is possible to obtain any speed vector from the current module selection configuration, and thus, using all the moves, the solution space exploration is continuous in the speed or module selection dimension, which is a desirable property of SA algorithms.

3.2.3. Post-Move Processing

The post-move processing is the steps executed every time a move is generated and the speed vector is modified. Different from the SA in Section 2.6, the new speed vector generated by the moves is not a complete solution. Therefore, the new speed vector is passed into the internal scheduler, where a new scheduling solution is generated. The total power consumption of the new solution is calculated by our power model, followed by the normal SA steps.

3.3. Internal Scheduler

The internal scheduler is to generate a scheduling solution according to a speed vector. It does not handle module selection directly but should be able to handle operations with different speeds. Since a speed vector is evaluated by its corresponding scheduling solution and the final solution quality is ultimately determined by the scheduling quality, the scheduler should be effective in power minimization. Besides, due to the SA nature, the runtime of the scheduler should also be low. These requirements above make the scheduler important and non-trivial.

For these purposes, we slightly modified the efficient list scheduling (LS) introduced in Section 2.4 and significantly improved its effectiveness in FU minimization, which is indirectly related to power minimization. We observed that many FUs are under-utilized in LS and many operations are schedule in late cc's due to the lack of available FUs in earlier cc's. Such operations could be scheduled earlier by allocating new FUs earlier, which leads to an average higher FU utilization and ultimately reduces the number of FUs in the final scheduling solution.

First, we define the *utilization rate* $ur_w(k, s)$ of an FU *w* of FT *k* and speed *s* as:

$$ur(k,s) = \frac{A_w \times d(k,s)}{L} \tag{4.5}$$

where A_w is the number of operations that bind to FU *w* and *d(k, s)* is the delay of FUs of FT *k* and speed *s* and *L* is the achieved latency. Thus, the utilization rate sum *UR(k, s)* of new FUs (allocated during scheduling process rather than allocated initially before any operation is scheduled) of FT *k* and speed *s* is:

$$UR(k,s) = \sum_{i=1}^{num_{new}(k,s)} ur_i(k,s) \tag{4.6}$$

where $num_{new}(k, s)$ is the number of new FUs of FT *k* and speed *s*. The ceiling of the utilization rate sum is an optimistic estimate of the additional number of FUs to be allocated before any operation is scheduled. We can thus iteratively perform LS to schedule with an incrementally updated initial FU allocation for each FT and speed using the utilization rate sum as a guideline, until the initial FU allocation is all the same to the final allocation, i.e., no new FUs are allocated during the scheduling process.

The pseudo code of PSA is presented in Figure 8.

Algorithm PSA(DFG $G(V, E)$, a multi-speed library K, a legal latency constraint L_c)
1. Initialize the speed vector SV
2. Generate an initial scheduling solution S = MLS(G, K, L_c, SV)
3. Adaptively calculate the initial temperature $t = t_0$ and the freezing temperature t_e
4. **While** ($t > t_e$)
5. **For** (i from 0 to $2n$)
6. Randomly generate a successful move and hence a new speed vector SV_n
7. Generate a new scheduling solution S_n = MLS(G, K, L_c, SV_n)
8. $S = S_n$ according to the probability $min\left\{1, e^{-\frac{S_n.cost - S.cost}{T}}\right\}$
9. $i = i + 1$
10. **End for**
11. $t = 0.9t$
12. **End While**
13. **Return** the scheduling solution

Algorithm MLS(G, K, L_c, speed vector SV)
1. $\boldsymbol{r} = (1, 1, ..., 1)$ //allocate one FU per FT and speed before scheduling
2. Compute the ALAP times t^{ALAP} per L_c
3. **Repeat**
4. Unschedule all operations
5. **While** (there are unscheduled operations)
6. **For** (each FT k)
7. **For** (each speed s)
8. Determine the available unscheduled operation set $U_{t,k}$
9. Compute slack s_u for all $u \in U_{t,k}$ by Equation (2.6)
10. Schedule 0-slack operations in $U_{t,k}$ to t, allocate new FUs if needed, update r_k if new FUs are allocated
11. Schedule non-0-slack operations in $U_{t,k}$ to t in slack-increasing order and bind them to remaining available FUs
12. **End For**
13. **End For**
14. $t = t + 1$
15. **End while**
16. **If** (there are new FUs allocated)
17. Calculate utilization rate sum $UR_{k,s}$ for each FT k and speed s for which there are new FU allocated and update $r_{k,s}$ as $r_{k,s} = r_{k,s} + UR_{k,s}$
18. **End if**
19. **Until** (there are no new FUs allocated)
20. **Return** the scheduling solution

Figure 8. The pseudo code of PSA with the modified list scheduling.

3.4. Experimental Results

We implemented our PSA algorithm in C++. Experiments were conducted on a machine with Core i7-4710HQ (3.5GHz) and 16GB RAM in Windows 10. We constructed 2-speed and 4-speed FU libraries by [36] with 65nm technology. TABLE I shows the power-delay characteristics of our 4-speed libraries for the main FTs: addition, multiplication and division. The ratio of dynamic to leakage power for all designs is 3:1. The 2-speed library only includes the slowest and the fastest speeds of the 4-speed library as in TABLE I. For non-arithmetic FTs, like shift registers and SRAM, there is only one speed in the libraries, since the power and delay of these FTs are far less than the arithmetic FTs and it is hard to obtain design variations for them, as discussed in more details in Section 4.4. The latency constraint L_c was defined as a fraction of the ASAP latency using FUs of all slowest speeds: $L_c = \alpha \times L_{asap}(slowest)$, where α is the fraction controlling the latency constraint and $L_{asap}(slowest)$ is the ASAP latency using FUs of all slowest speeds. The input DFGs are from [37].

We also implemented an extended version of SA in Section 2.6 considering module selection. The moves set was modified to randomly shift a randomly chosen operation to a new configuration, that is, changing either its scheduled cc, speed or both to generate a new solution from the current solution. To make a fair comparison to PSA, the initial solution generation, temperature scheme and acceptance criteria of this SA is all the same to PSA. For the local search configuration at each temperature, the number of moves M generated is $h \times n$, where h is a constant control variable and n is the number of operations. The total power and runtime results of the modified SA and PSA-LS using the 2-speed FU library with a latency parameter of 1.2 is presented in TABLE II. We can see that PSA with LS (PSA-LS) reduces the total power by an average 6.5% to 10.7% compared to the extended SA meanwhile the runtime of PSA-LS is

95.2% to 97.9% less than that of the SA, across all the DFGs and local search configurations. We then choose h = 4 for the local search part of the SA-based algorithms in the remaining experiments in this section.

We then compare the LP and runtime among competing algorithms ILP, MWIS [38], Min-Cut [39], and PSA-MLS with 2-speed library in TABLE III, since MWIS and Min-Cut techniques are for LP minimization and can only handle scheduling with 2-speed FU libraries. Both techniques must be significantly extended to utilize libraries with more than 2 speeds. Besides, we implemented MWIS and experimented with two initial solution configurations: the maximum resource sharing and the least resource sharing by duplicating all module instances as in [38]. The best solution out of the two is reported. Min-Cut requires to assign the slowest speed to all operations before the scheduling such that the ASAP latency is greater than the latency constraint. To adapt to the limitation of the algorithm, we choose α = 0.6. From the results in TABLE III, we can see that PSA-MLS has an average LP improvement of 12.4% and 5.2% compared to MWIS and Min-Cut while keeping the runtime close to them. Further, compared to the optimal ILP results, PSA-MLS only has an 13.4% optimality gap in average and is 84 times faster.

Finally, we present the results of total power minimization for ILP, the extended SA and our PSA with various internal schedulers with 2/4-speed libraries in TABLE IV. The latency constraint is 2 times of that used in TABLE III. To demonstrate the flexibility of the proposed SA framework and the effectiveness of our MLS scheduler, we nested LS (detailed in Section 2.4), FDS (detailed in Section 2.5) and MLS. “*” signs in the table means the solution of ILP was still not available after running 24 hours, hence we took the latest solution after 24 hours and record the runtime when the solution was available for the first time. From the results, we can

see that PSA-MLS with the 2-speed library only has an average optimality gap of 5.9% and is 193 times faster. The optimality gap is even closer when the 4-speed library is used: it is only 4.8%. Besides, the various PSA formulations are significantly better than the extended SA by at least 23.7% in average for the 2 libraries. The results also show that PSA with MLS consistently achieves better average results than PSA with FDS with significantly reduced runtime (179 times faster).

The results above demonstrate the efficacy and efficiency of PSA in power optimization with module selection.

TABLE I

POWER-DELAY CHARACTERISTICS OF ARITHMETIC FTS OF THE 4-SPEED FU LIBRARY

FT	Design	Delay (cc)	Power (mW)
Addition	Kogge-Stone	1	0.673
	Carry Select (Brent-Kung)	2	0.612
	Carry Select (Ripple-Carry)	3	0.612
	Ripple-Carry Adder	4	0.556
Multiplication	Tree-Multiplier-Add	4	9.725
	Dadda (with Kogge-Stone)	6	9.007
	Wallace Tree (with Brent-Kung)	8	9.007
	CSvA-Based-Multi-Add	10	5.004
Division	Radix-8 (Kongge-Stone based)	8	14.68
	Radix-4 (Kongge-Stone based)	12	11.65
	Radix-8 (Brent-Kung based)	16	11.65
	Radix-4 (Brent-Kung based)	24	9.174

TABLE II

TOTAL POWER AND RUNTIME COMPARISON BETWEEN PSA-LS AND THE EXTENDED SA

DFG	L_c	Total Power (mW·cc/GHz)										Runtime (s)									
		SA					PSA-LS					SA					PSA-LS				
		h = 1	*h = 2*	*h = 4*	*h = 8*	*h = n*	*h = 1*	*h = 2*	*h = 4*	*h = 8*	*h = n*	*h = 1*	*h = 2*	*h = 4*	*h = 8*	*h = n*	*h = 1*	*h = 2*	*h = 4*	*h = 8*	*h = n*
hal	26	36.1	35.6	35.6	35.0	35.6	34.5	34.5	34.5	34.5	34.5	2.8	4.9	8.8	19.0	25.3	0.1	0.2	0.3	0.7	0.9
arf	60	132.3	127.7	126.6	124.3	121.5	115.4	115.4	115.4	115.4	115.4	9.5	19.7	42.5	98.0	575.0	0.5	1.2	1.6	2.9	9.2
motion	28	111.8	111.5	109.4	108.4	106.5	106.0	106.0	103.3	103.3	103.3	11.5	22.2	48.9	121.0	958.4	0.4	0.7	1.0	2.0	8.2
ewf	88	83.5	83.0	82.5	80.1	81.0	69.0	69.0	69.0	69.0	69.0	11.3	21.9	52.8	129.0	261.0	0.7	1.2	2.4	4.7	19.5
Avg	51	90.9	89.5	88.5	87.0	86.2	81.2	81.2	80.6	80.6	80.6	8.8	17.2	38.3	91.8	454.9	0.4	0.8	1.3	2.6	9.5

TABLE III

LEAKAGE POWER AND RUNTIME COMPARISON AMONG THE COMPETING ALGORITHMS

DFG	L_c	Size[a]	LP (mW·cc/GHz)				Runtime (s)			
			ILP	MWIS	Min-Cut	PSA-MLS	ILP	MWIS	Min-Cut	PSA-MLS
hal	13	11	18.0	18.0	18.0	18.0	0.2	0.2	0.2	0.4
arf	30	28	49.0	49.0	49.0	49.0	0.3	1.5	1.5	1.5
motion	14	32	37.3	37.3	37.3	37.6	0.5	1.8	1.5	1.1
ewf	44	34	34.1	37.4	35.7	34.1	7.3	4.0	3.5	2.1
feedback	18	53	53.2	58.4	56.1	54.2	9.3	6.2	6.0	2.3
collapse	18	56	40.8	49.2	46.3	45.0	11.3	8.3	7.6	2.8
write	17	106	25.4	32.6	30.8	29.6	16.7	4.1	3.1	3.4
aux	21	108	117.9	141.0	132.8	125.1	25.2	7.1	6.4	7.6
matmul	25	109	109.1	150.2	135.8	130.5	104.9	6.7	6.8	8.7
idctcol	40	114	73.0	114.7	90.4	84.4	1212.5	8.7	7.3	16.2
jpeg	36	134	93.3	138.3	119.6	112.6	2975.1	12.1	10.3	21.2
smooth	36	197	185.7	257.7	249.1	228.8	3354.1	25.2	23.3	24.6
Avg	26	81.8	69.7	90.3	83.4	79.1	643.1	7.2	6.5	7.7

[a] DFG sizes are indicated by the number of operations.

TABLE IV

TOTAL POWER AND RUNTIME COMPARISON AMONG ILP, SA AND VARIOUS VERSIONS OF PSA WITH 2/4-SPEED LIBRARIES

DFG	L_c	2-Speed Library										4-Speed Library				
		Total Power (mW·cc/GHz)					Runtime (s)					Total Power (mW·cc/GHz)				
		ILP	SA	PSA-LS	PSA-FDS	PSA-MLS	ILP	SA	PSA-LS	PSA-FDS	PSA-MLS	ILP	SA	PSA-LS	PSA-FDS	PSA-MLS
hal	26	34.5	35.6	34.5	34.5	34.5	0.3	8.8	0.3	1.9	0.3	34.5	36.6	34.5	34.5	34.5
arf	60	99.2	126.6	115.4	99.2	106.8	1.9	42.5	1.6	52.7	1.1	98.6	127.7	109.6	99.2	101.7
motion	28	89.4	109.4	103.3	91.0	94.2	2.1	48.9	1.0	19.1	1.1	88.4	111.5	103.9	90.5	91.9
ewf	88	69.0	82.5	69.0	81.5	69.0	16.4	52.8	2.4	169.0	1.6	68.5	85.2	69.0	71.2	69.0
feedback	37	106.6	150.0	125.8	108.7	112.5	3.5	153.0	5.3	204.7	2.7	106.6	155.8	123.9	108.7	108.5
collapse	37	79.5	107.4	104.6	84.4	84.7	18.1	40.6	1.8	161.0	2.6	79.5	109.1	103.1	81.1	82.3
write	34	49.4	81.9	55.2	51.7	51.1	68.4	91.2	6.9	2468.0	9.7	49.4	81.9	53.6	51.7	50.4
aux	42	230.4	354.8	256.5	241.5	233.0	146.4	151.3	7.9	1423.0	11.0	230.4	359.6	253.7	243.4	233.0
matmul	50	242.3	352.0	288.4	263.6	256.6	593.6	227.1	7.6	1675.0	11.0	241.36*	362.4	286.7	262.5	254.9
idctcol	80	187.5	322.9	219.1	204.5	194.8	1417.0	194.4	11.1	3162.0	17.5	179.62*	322.9	232.2	204.1	190.1
jpeg	73	236.4	409.0	264.5	261.1	253.9	1986.0	381.1	14.4	4521.0	21.1	236.4*	409.0	258.1	260.3	244.9
smooth	73	457.5	693.9	520.3	482.3	502.5	18072	720.6	28.2	6916.0	36.2	445.3*	727.6	542.0	482.3	486.0
Avg	52.3	156.8	235.5	179.7	167.0	166.1	1860.5	176.0	7.4	1731.1	9.7	154.9	240.8	180.9	165.8	162.3

4. NEW CRITERIA FOR POWER-DRIVEN RESOURCE LIBRARY CONSTRUCTION

The work in this chapter has been partially presented in [1] and published in [3].

In the chapter, we propose that the degree of optimization achievable in HLS with module selection (essentially a multi-speed library) significantly depends on how the mix of FUs in the library are parameterized. In particular, for MP-LCS problem, after introducing module selection in HLS and showing our motivation examples, we propose that appreciably better power optimization is possible when:

- the FUs for each FT have a wide range of both power and delay metrics;
- their pair-wise power-delay product ratios are close to 1, say, in the range [0.8, 1.25], than when these criteria are not satisfied.

For this, we provide a probabilistic justification. We also briefly show that it is possible to achieve these parameter ranges for arithmetic FTs, which form the bulk of FUs in HLS problems, due to:

- the variety of designs that have been proposed for such functions;
- the flexibility of combining different design approaches in a hierarchical manner to yield hybrid designs that satisfy the aforementioned parameter constraints in case the original designs do not.

This is justified by an analysis of theoretical design examples.

The work in this chapter has been published in [3] and partially presented in [1].

4.1. Introduction of Module Selection

An additional dimension in the design space of HLS, module selection, can provide a significant space to improve the power cost in HLS. FUs with different designs (design diversity) potentially offer a much larger range of power and delay characteristics than is available with current approaches such as multi-V_{dd} and multi-V_{th} FUs that have been used in prior power-driven HLS works. We can judiciously select from a diverse set of FU designs to obtain a minimal power cost. Different FU designs have different power and delay characteristics, can execute significantly different numbers of operations with a given latency constraint and have different fragmentation (to be defined later) amount. For example, a ripple-carry adder (RCA) has a delay and power consumption (sum of DP and LP) of $\Theta(n)$, where n is the number of input bits. In this analysis, we realistically assume that the dynamic as well as LP are proportional to the total number of gate inputs across the circuit, which is $\Theta(n)$ for an RCA and $\Theta(2n)$ for a carry-lookahead adder (CLA). Thus, an RCA's power per throughput metric is $\Theta(n^2)$ (throughput $\propto$ 1/delay). On the other hand, a CLA has a delay of $\Theta(2\log n)$, and power consumption of $\Theta(2n)$, giving a power per throughput metric of $\Theta(4n\log n)$, which is much lower than that of an RCA. However, this does not necessarily mean that a CLA is always better than an RCA, since dynamic power, which is lower per computation for a CLA, is consumed only during operations, but total LP, which is more for a CLA than an RCA ($\Theta(2n)$ versus $\Theta(n)$), is consumed throughout the latency constraint of the design (or over periods in which the FU is on). At the same time, since fewer CLAs could be used to replace, say, k RCAs (by a factor roughly proportional to their delay ratio of $\Theta((2\log n) / n)$ if allowed by precedence constraints, it is not necessary that all the CLAs in the design will expend more LP than all the RCAs they have replaced.

Let us consider the example illustrated in Figure 9, where the RCAs (slow FUs) have a delay of 4 clock cycles (cc's) and CLAs (fast FUs) with a delay of 2 cc's. Figure 9(a) shows a scheduling and binding solution using only RCAs for four addition operations u, v, w and x. The precedence constraint $w \rightarrow x$ and the latency constraint requires an allocation of 3 RCAs. However, these operations can be scheduled and bound to CLAs so that only one CLA is required as shown in Figure 9(b). Thus, even if the power cost of one CLA is 2.5 times that of one RCA, the solution in Figure 9(b) still saves 0.5 time of the power consumption of an RCA.

In terms of total power, whether a set of RCAs or CLAs is better for a group of operations depends on the dependency constraints and available slacks for these operations, and the corresponding utilization (or conversely, fragmentation) they impose on these FUs. In general, considering a mix of CLAs and RCAs (and in general, slow/low-power and fast/high-power FUs) should provide more power-efficient solutions under given latency constraints.

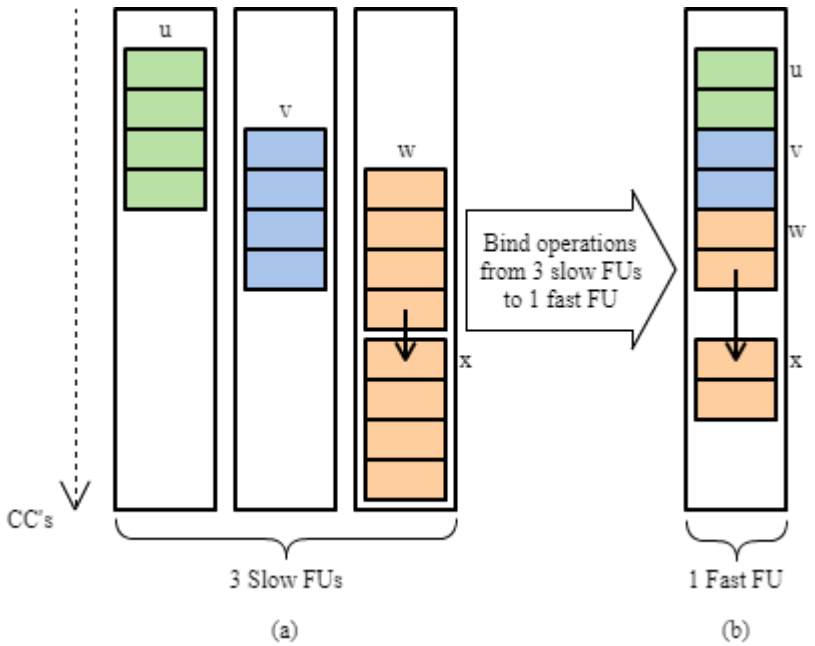


Figure 9. An example for power reduction with module selection. The colors represent different operations and they are bound to FUs represented by large rectangles.

4.2. Motivation of Resource Library Construction

Although module selection can effectively reduce power consumption as demonstrated in Figure 9, it significantly depends on the FUs in the library. If the power of a CLA is at least 3 times greater than that of a RCA, substituting the CLA for RCAs will increase the power consumption. On the other hand, if the delay of a CLA is increased to 3 cc's, two CLAs will be allocated for substituting the RCAs, since operations executed on one CLA will violate the latency constraint as shown in Figure 10. Such module selection results in even more power consumption.

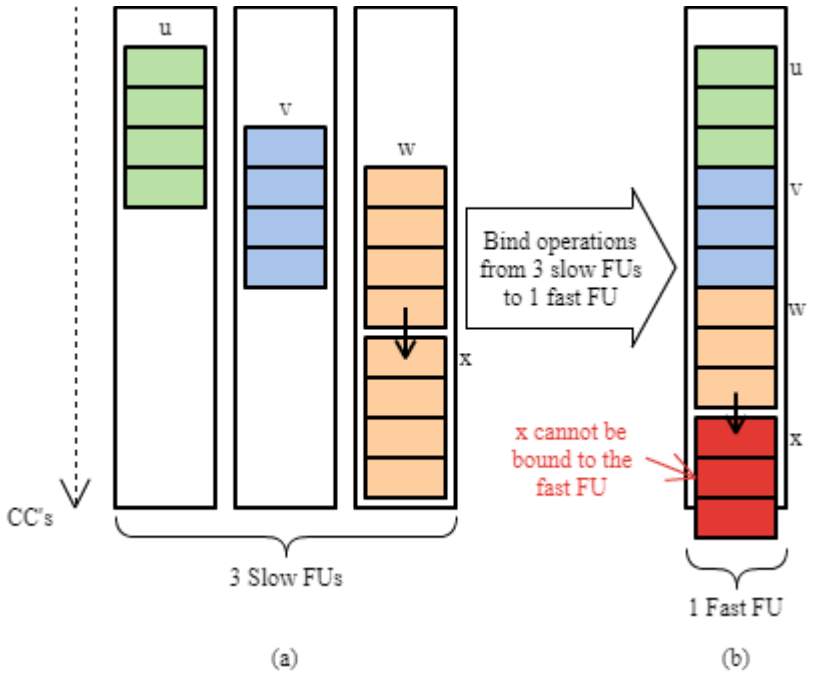


Figure 10. An example for unsuccessful power reduction with module selection. The colors represent different operations and they are bound to FUs represented by large rectangles.

Therefore, module selection does not help power minimization if the library it uses includes too many FUs that have undesired power and delay characteristics. Such FUs are called

bad FUs hereafter and should be avoided when constructing the library. However, as we have discussed in Section 1.4.4, no previous work noticed this issue and a lot of HLS works like [31] [32] [33] are still using these bad FUs. We will provide our hypotheses for better resource library construction and justify them in the following subsections.

4.3. Hypotheses for Effective Library Construction

Henceforth, unless otherwise specified, by power we mean the sum of dynamic and leakage energy consumed per clock cycle (cc). We first define the *power-delay-product* (PDP) of an FU as the product of the power and delay in cc's of the FU to execute an operation. Delay and power generally form a design trade-off, and it is desirable that as we move from a slow FU to a fast one, the ratio of the fast to slow PDPs (the *PDP ratio*) remains in the neighborhood of 1 (this can happen for arithmetic FUs like adder, multiplier and divider, as we have found via literature search and also by considering hybrid" designs—more on these issues later). A PDP ratio of 1 indicates that the speed increase of a fast FU has been obtained by a proportionate increase in total power to perform that operation in relation to the slower FU; a PDP ratio less than 1, indicates that the power increase is by a smaller factor than the speed increase (this also happens, as we show later, and means that the slower FU design is less power-efficient, i.e., its power-per-performance is more, than the faster one); a PDP ratio greater than 1 indicates the opposite trend.

4.3.1. The First Hypothesis

Our first hypothesis is:

HYPOTHESIS 1. *(a) A larger delay range along with a larger power range among a mix of FUs, and (b) a PDP ratio between the slowest FU and any of the faster FUs that is in the neighborhood of 1, say, in the range [0.8, 1.25], allows for better power optimization compared to smaller delay or power ranges or larger PDP ratios.*

Justification: We first illustrate this by our discussion in Section 4.2. If the delay range is smaller by increasing the delay of the CLA to 3 cc's, two CLA's will be needed and the power consumption will be increased significantly. Note that the PDP ratio of the CLA to the RCA here is increased from 1.25 to 1.875. On the other hand, if the delay range remains the same while the power range is increased by increasing the power of the CLA to more than 3 times of the RCA, substituting the CLA for the RCA also increases the power. Note that the PDP ratio of the CLA to the RCA here becomes greater than 1.5.

The desirability of a wide range of delay and power metrics is clear, since only FUs are appreciably distinguished by these metrics, will the module-selection search space be rich enough for proper exploration by a module-selection algorithm to arrive at a good tradeoff between the competing metrics. Empirical evidence for this aspect of Hypothesis 1 is provided in Section 4.4. We also note that a large delay or power range does not necessarily imply a large power or delay range, respectively, as is the case with FU libraries based solely on multi-V_{th} or multi-V_{dd} but with single designs. In these cases, the power ranges are greater than delay ranges, sometimes significantly, since: a) in the case of multi-V_{th} based FUs, the delay is roughly linearly proportional to V_{th} (it is inversely proportional to V_{dd} - V_{th}) but the LP is inversely proportional exponentially to V_{th}; and b) in the case of multi-V_{dd} based FUs, the delay is inversely proportional to V_{dd} but the DP is proportional quadratically to V_{dd} (the LP is proportional to V_{dd}). For example, in the 90nm dual-V_{th} ALU/adder and multiplier library of [40], the delay ranges are small: 1 to 2 cc's and 2 to 3 cc's, respectively, while the LP ranges are very high: 0.3 to 19.5μW and 2.3 to 124.6μW, respectively. Thus, the PDP ratios of the fast FUs are also very high, 32.5 and 36.1 for the adders/ALUs and multipliers, respectively. This implies that the fast FUs with low V_{th}'s are rendered almost useless and should not be selected by a

quality HLS algorithm for moderate-speed designs (those whose latency requirements are set to be greater than or equal to the critical path delay *Ds* of the DFG determined assuming operation delays correspond to the slowest FUs for each FT). In fact, our experiments with a state-of-the-art power-driven scheduling and module selection algorithm PSA [2] (details are in Chapter 4.5) on this library with latency constraints $L_c = (1 + \alpha)Ds$, for $\alpha \in [0, 1]$, showed exactly that: for all DFG's in the media benchmark suite [37] we use in all our experiments, the solutions did not use any fast FU. Thus, both analytical reasoning and empirical evidence, including our results in section 0, point to that only multi-V_{th} and multi-V_{dd} based FU designs do not satisfy Hypothesis 1 and are not good enough for low-power design, and that mainly design diversity based FUs (possibly coupled with multi-V_{th} and multi-V_{dd}) are needed for this purpose.

We now perform a probabilistic analysis to theoretically support Hypothesis 1 in terms of the PDP ratio criterion. Let p_1 = the probability that any cc t an operation is available for scheduling on an FU A at any stage during the HLS process. This means that if t is an available/idle cc in the FU in question, and there are $d - 1$ consecutive slots available after cc t on FU A, where d is the delay of an operation on the FU, then an operation can be scheduled on A with a probability of $p1$. Let p_2 = the probability that if an operation is available at an available cc t and t is available in FU A, then the $d - 1$ slots following t are available in A (we term this as the slot t on A being *fragmentation-free*). Since we can schedule at most L_c / d operations on A, where L_c is the latency constraint, we can schedule at most L_c / d cc's; thus, the average number of operations scheduled on FU A is $p_1 p_2 L_c / d$. Further, let u, $0 < u \leq 1$ be the average utilization factor of FUs (fraction of cc's which are used to execute operations) in the final synthesized design. This implies that the average number of operations scheduled on an FU with delay d is u

L_c / d. We thus have $p_1\ p_2\ L_c / d = u\ L_c / d \rightarrow p_1\ p_2 = u$. For simplicity, assuming $p_1 \approx p_2$, we get $p_1 \approx \sqrt{u}$.

Our extensive experiments using a strong power-driven simulated annealing based scheduler PSA [2] on several media benchmark DFGs with sizes of up to about 200 operations, and using 11 different latency constraints for each DFG, show that the average u is around 0.5 with a small standard deviation of 0.09. We thus get $p_1 \approx 0.7$.

Next, we consider the average number of operations that can be accommodated in a fast FU that replaces a slower one, including those that are transferred from the slower FU. Consider a slow FU A with delay d with m operations scheduled on it ($m = u\ L_c / d$ in average), and a faster FU B with delay d / s (a speedup of s) and with no operation on it. We want to determine the average number of operations that can be scheduled on B after the m operations of A are mapped to it. In average, A will have $(1 - u)L_c$ slots available. If we transfer the m operations from A to B without changing their scheduling times, we will have $(1 - u)L_c + m(d - d / s)$ slots available on B for scheduling. Of these:

(a) $m(d - d / s)$ slots are available due to the shrinking of the delay of m operations transferred from A by a factor of s (see Figure 11)—we call these the *freed time slots* on B;

(b) Of the remaining $(1 / u)L_c$ available slots on B, one category is the set of available slots transferred from A on which operations could not be scheduled due to *fragmentation* (see Figure 12); we will call the latter *A-fragmented* (A-f) time slots;

(c) The second category of the $(1 - u)L_c$ available slots on B are those that were fragmentation-free on A, but in which operations could not be scheduled due to unavailability of operations for A at those slots. Part of the reason for this is that more slow FUs were available at such fragmentation-free slots/cc's t than there were operations of that FT available at t. Thus,

some of these slow FUs like A have no operations scheduled in such fragmentation-free slots. However, in the scenario where we are using fast FUs to replace some slower ones, there will be approximately a factor of *s* fewer fast FUs than slow ones, and thus for a cc like *t* on a fast FU B, there is a higher likelihood of operations being available for scheduling. We can thus schedule some extra operations in B at such fragmentation-free slots.

We next analyze the increased number of operations that can be scheduled on B compared to A for the three scenarios above.

(a) Scheduling operations on freed time slots on a fast FU.

We first analyze the average number of operations that can be scheduled on a set of freed time slots corresponding to a single operation on A mapped to B. There are *(d – d / s)* such consecutive slots available on B; see Figure 11. Given *k* consecutive available slots/cc's on an FU with delay l, the average number of operations schedulable in the *k* slots is $p_1\ k\ /\ \lambda$, since operations are available to be scheduled in the $k\ /\ \lambda$ starting cc's among the *k* slots with a probability of p_1. Thus, the average number of operations that can be scheduled in the freed slots on B corresponding to a single mapped operation from A is:

$$\frac{p_1(d - \frac{d}{s})}{d/s} = p_1(s - 1) \tag{3.1}$$

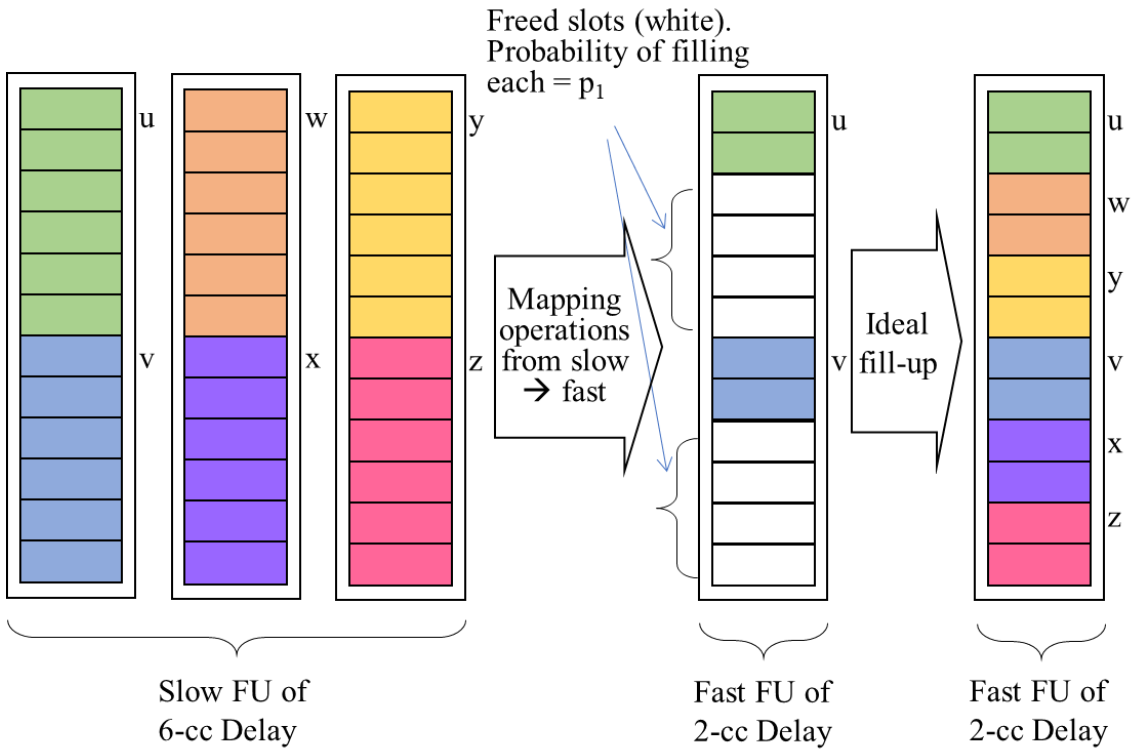


Figure 11. Mapping operations from slow FUs to a fast FU.

(b) Scheduling operations on slow FU fragmented time slots on a fast FU.

An available slot *t* on A is fragmented (A-f slot) if some operation *w* is scheduled on A in any time slot in the range [$t + 1$; $t + d - 1$]. As illustrated in Figure 12, the slot on B corresponding to the A-f slot in cc *t* is schedulable as long as *w* (which will also be mapped to B at the same slot on which it was scheduled on A) is not scheduled in the range [$t + 1$, $t + (d / s) - 1$] ($d / s = 2$ in Figure 12). In general, we may be able to schedule more than one extra operation (those not scheduled on A) on B in the range [$t + 1$, $t + d - 1 - (d / s)$], depending to the cc that *w* is scheduled in the range [$t + 1$, $t + d - 1$]. We can estimate the average number of such extra operations we can schedule on B corresponding to the A-f cc *t*, by dividing up the range [$t + 1$, $t + d - 1$] into *s* contiguous regions, with the first $s - 1$ of them having d / s cc's each, and the last one having $(d / s) - 1$ cc's. If *w* is scheduled in the *i*'th such region, $1 \leq i \leq s$, then we can

schedule $i - 1$ extra operations (if available) on B in the $i - 1$ preceding regions. The probability that w is scheduled in any of the s regions is approximately $1 / s$ (it would be exactly $1 / s$ if the last region also had d / s instead of $(d / s) - 1$ cc's). Thus, the average number of extra operations, if available, that can be scheduled on B corresponding to the A-f time slot t is:

$$\frac{1}{s} \times \sum_{i=1}^{s} i - 1 = \frac{s-1}{2} \tag{3.2}$$

Next, we need to determine the probability of operation availability to be scheduled in the $i - 1$ regions preceding the aforementioned i'th region in which w is scheduled, and multiply it with $i - 1$ in Equation (3.2) to obtain the average number of extra operations we can schedule on B corresponding to the A-f time slot in cc t. If w is scheduled at exactly the first cc in the i'th region, then to schedule $i - 1$ extra operations in the preceding regions, we need to have an operation available to schedule at exactly the first cc in each region. The probability for this is p_1 in each of the $i - 1$ regions, and thus the average number of extra operations that we can schedule is $p_1(i - 1)$. However, if w is scheduled in the 2nd cc of the i'th region, we have the flexibility to schedule one of the extra operations in the preceding $i - 1$ regions in either the 1st or 2nd cc of its region, and the overall probability of being able to schedule $i - 1$ extra operations in the preceding $i - 1$ regions is a little more than p_1:

$$\frac{1}{i-1}[(i-2)p_1 + 1 - (1-p_1)^2] \tag{3.3}$$

where $1 - (1 - p1)^2 \geq p_1$ is the probability for having an operation available to schedule in either the 1st or 2nd cc of its region. The probability expressions for operation availability become more complex as there is more flexibility to schedule operations in the preceding $i - 1$ regions based on w being scheduled anywhere from the 3rd to (d / s)'th cc in the i'th region. However, based on numerical examples, we can approximate the average probability to be $1.2\ p_1$, and thus

the number of extra operations that can be scheduled on B as $1.2\ p_1(s - 1) / 2$, where we introduce a small fudge factor of 1.2 to account for the probability of operation availability across an average of $(s - 1) / 2$ regions being greater than p_1. For example, for $d / s \geq 3$ and $s = 2$, if w is scheduled in the 2nd region, then the probability p_{avail} of having an operation available for scheduling in the 1st region when w is scheduled in the k'th cc in the 2nd region, $1 \leq k \leq 3$ are:

- if $k = 1$, $p_{avail} = p_1 = 0.7$;
- if $k = 2$, $p_{avail} = 1 - (1 - p_1)^2 = 0.91$;
- if $k = 3$, $p_{avail} = 1 - (1 - p_1)^3 = 0.973$.

Thus, the average $p_{avail} = 0.861 = 1.23\ p_1$, assuming w has an equal probability of being scheduled in any cc 1 to 3 in the 2nd region. Thus, the average number of extra operations we can schedule in FU B in the previously fragmented space in FU A corresponding to the A-f slot t is $1.23\ p_1(s - 1) / 2$ in this example.

Further, the position of potential A-f cc's that we need to consider for extra operation scheduling are right after the end of the execution of a scheduled operation on A (if we use any available cc after such positions in the same fragmented space, then we will be counting the extra operation schedules multiple times for each such maximal fragmented space between two consecutive operation schedules on A that are mapped to B). Of the $\mu L_c / d$ such positions at the end of each operation's execution on A, the A-f ones are those that are fragmented, the probability of which is $1 - p_2 \approx 1 - p_1$ (based on our initial assumption that $p_1 \approx p_2$). There are thus $(1 - p_1)\ \mu L_c / d$ such A-f slots, and we call these maximal A-f slots, since they are starting slots of maximal fragmented spaces. Thus, since there are $\mu L_c / d$ operations scheduled on A in average, and we can only schedule extra operations on B corresponding to its maximal A-f slots, the fractional increase of extra operations on B due to these schedulings is:

$$\frac{(1-p_1)1.2p_1(s-1)}{2} \tag{3.4}$$

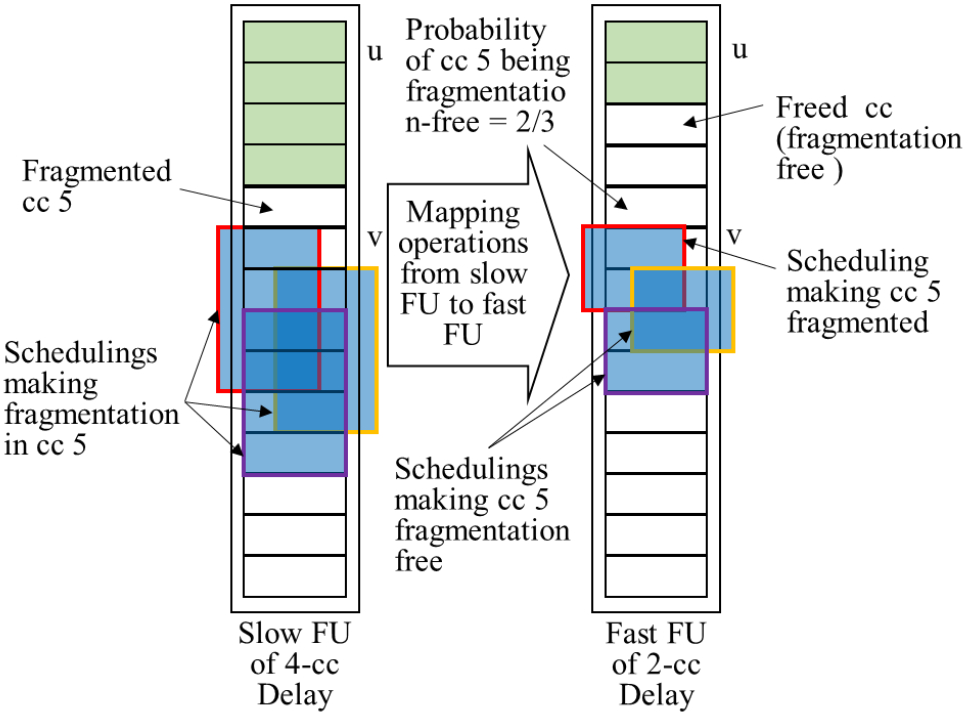


Figure 12. Mapping operations from a slow FU to a fast FU does not always cause fragmentation in the latter, allowing more scheduling flexibility on the latter than the former.

(c) Scheduling operations on slow FUs' fragmentation-free time slots on a fast FU.

There will be $p_2\ \mu L_c / d \approx p_1\ \mu L_c / d$ fragmentation-free available spaces in A occurring right after a $p_2 \approx p_1$ fraction of the average number of scheduled $\mu L_c / d$ operations on A. By definition, there are at least d cc's in these spaces between two consecutive operations scheduled on A, and are available for scheduling on B. Thus at least s operations of delay d / s can be scheduled in each such space. As mentioned earlier, while

there were no operations available for A in these fragmentation-free slots/cc's, there is a reasonable likelihood of operations being available for scheduling on B in the scenario of replacing some slow FUs with faster ones: there will be fewer fast FUs used than the slow ones

that they are replacing by about a factor of *s*, and thus in each of these cc's the ratio of the number of operations available for scheduling to the number of available FUs will be much higher than in the slow FUs case. We will assume that the probability of this availability is the general operation availability probability p_1. Thus, the number of operations that can be scheduled in the fragmentation-free available spaces of A mapped to B is $(p_1 \mu L_c / d)\, p_1\, s$, which represents a fractional increase in these number of operations compared to those on A of:

$$\frac{\frac{p_1 \mu L_c}{p_1 s}}{\frac{\mu L_c}{d}} = p_1^2 s \tag{3.5}$$

Adding the fractional increases of operations scheduled on B compared to those in A from Equations (3.1), (3.4) and (3.5), we have a total fractional increase of:

$$p_1(s-1) + \frac{1.2 p_1 (1-p_1)(s-1)}{2} + p_1^2 s \tag{3.6}$$

which for $p_1 = 0.7$ is 1.32s – 0.826. Thus, the average total number of operations scheduled on B is the following factor of the number of operations of A:

$$1 + 1.32s - 0.826 = 1.32s + 0.174 \approx 1.32s \tag{3.7}$$

Thus, if B's PDP ratio is around 1.32 (i.e., B's power factor over A is 1.32*s*), we about break even in power by utilizing fast FUs B to replace some slow FU's A. Hence, for there to be some reasonable power reduction of at least 5% by choosing fast FUs to replace some slow ones, we choose a more beneficial PDP ratio of 1.25, which will afford us a (1.32 – 1.25) / 1.25 = 5.6% power reduction.

The lower PDP ratio range of 0.8 in Hypothesis 1 is based on our design experiences, as well as on FU designs available in literature like [41] [42] [43], that show that it is possible to get a PDP ratio of a fast FU to be less than 1, and a pragmatic lower limit is 0.8.

4.3.2. The Second Hypothesis

Our second hypothesis is based on our exploration of hybrid designs of various arithmetic FTs as well as our literature search of a variety of designs for these FTs, and concerns the attainability of the PDP ratio range mentioned in Hypothesis 1.

HYPOTHESIS 2. *It is possible to design a mix of slow to fast FUs for arithmetic functions that satisfy the criteria of Hypothesis 1.*

Justification: The availability of a wide range of delay and power metrics across different FU designs for various arithmetic FTs is well-known. The designs range from completely sequential processing (e.g., RCA for addition, add/subtract and shift for multiplication and division) resulting in linear to quadratic delays, to highly parallel processing resulting in $\Theta(log\ n)$ delays (e.g., CLA, Wallace-tree multiplier) with delay functions in between (e.g., $\Theta(\sqrt{n})$ for a carry-select adder [CSA]), where n is the input size. This wide range of delays come with the attendant wide range of power consumption.

The more interesting issue is that of attaining a PDP ratio of a fast FU to the slowest FU in the library that lies in the range mentioned in Hypothesis 1. Firstly, we have looked at numerous arithmetic designs like [41] [42] [43] to determine if the original designs are such that they fit this criterion. As shown in TABLE V, a carefully chosen set of dual FU designs from these works easily satisfies this criterion.

Further, irrespective of the variety of available designs (that may or may not fit the PDP ratio criterion) across more than two FUs for each FT, we contend that it is possible to obtain hybrid designs that hierarchically combine different well-known designs (e.g., RCA, CLA and CSA for addition, array and Wallace-tree multiplier for multiplication) to satisfy the PDP ratio criterion. Let us take the example of an *n*-bit RCA and an *n*-bit CLA with 4-bit basic carry-and-

P/G generation units. We will consider gate cost (i.e., number of basic gates) in units of 2-input gates, and critical path delay in terms of number of inputs encountered in the path with the maximum number of this parameter. We assume for simplicity of exposition, that the dynamic as well as leakage power is linearly proportional to the gate cost—the total switching activity is linearly proportional to the number of gates (assuming a more or less uniform switching probability among the gates in a design), as is the leakage power. Thus, we will simply use total gate cost as a measure of total power expended by gates.

An n-bit RCA has gate cost = $7n$, and a critical path delay of $5n$. For n = 64, these numbers are 448 units and 320 units, respectively. A CLA has a gate cost = $36 \times (n / 4) + 16(n - 1)$, and a critical path delay of $8\log_4 n + 2\log_4 n = 10\log_4 n$ (the 1st term is for P, G generation, and the 2nd is for the subsequent carry generation). For n = 64, these numbers are 1584 and 30 units, respectively. Further, we need to consider interconnect delay and interconnect dynamic power consumption. The layout of an RCA is easily accomplished by almost “unit-length" interconnects between adjacent full adders (FAs). However, the average interconnect length will be much longer in a tree-structured circuit such as a CLA, and we conservatively assume that the average interconnect in a CLA is 2.5 times as long as in an RCA. We also assume that an RCA interconnect will expend as much dynamic power as a 2-input gate and 1.5 times of the delay (delay on an interconnect will be approximately quadratic in its length), and that dynamic and leakage power are roughly equal for a 2-input gate. Thus, the total power units in a 64-bit RCA is 448 + 448 / 2 = 672 (the first term is dynamic + leakage power in gates, and the second term is the dynamic power on interconnects), and its delay is 320 + 1.5 × 320 = 800 units. The corresponding numbers for a 64-bit CLA will be 1584+2.5 × 800 = 3564 units of power, and 30 + $1.5 \times 2.5^2 \times 30$ = 311 delay units. Thus, the PDP ratio of a 64-bit CLA to a 64-bit RCA is 3564

× 311 / 672 × 800 = 2.06, which does not fall in the desired range. However, if we obtain a hybrid design for a fast adder, in which we have four 16-bit CLAs, with the carries rippling between them, then the gate cost is 4 × 384 = 1536, and the delay is $8\log_4 16 + 4 \times 2\log_4 16 = 32$ units (the P, G computations take place in parallel in the four 16-bit CLAs: $8\log_4 16$, and then the carry generation is $2\log_4 16$ in each of them, and is sequential between them, i.e., the carry ripples between the 16-bit CLAs). The interconnect length ration between this hybrid design and an RCA will be less than 2.5, and we assume it is 2. Then, the total power units of the 64-bit hybrid design is 1536+2 × 768 = 3072, and delay is $32 + 1.5 \times 2^2 \times 32 = 224$ units. The PDP ratio of this design to an RCA is 3072 × 224 / 672 × 800 = 1:28, which falls almost in the desired range.

Similarly, since multipliers and dividers can be constructed hierarchically using a divide-and-conquer approach, we can use different designs at different levels to yield a hybrid design. For example, the multiplication $P = X \times Y$, where X, Y are n-bit numbers, can be broken up into:

$$P = \left(2^{\frac{n}{2}} \times X_h + X_l\right) \times \left(2^{\frac{n}{2}} \times Y_h + Y_l\right)$$

$$= 2^n (X_h \times Y_h) + 2^{\frac{n}{2}} (X_h \times Y_l) + 2^{\frac{n}{2}} (X_l \times Y_h) + (X_l \times Y_l) \quad (3.8)$$

Here X_h(X_l) is the higher (lower) n / 2 bits of X. Each of the above 4 sub-products is an (n / 2)-bit multiplication; the 2^i place value multiplications of these sub-products only represent proper place-value alignment during their summation. To obtain a hybrid Wallace-tree and array multiplication, we can use Wallace-Tree for each of the 4 sub-products to yield 2 summands for each, and then use the array multiplication structure (sequence of 8 carry-save additions) on the total of 8 summands. The final 2 summands can be summed by an adder to yield P. Thus, we believe that there is reasonable flexibility available to develop hybrid designs for various arithmetic functions in order to attain the desired PDP ratio given in Hypothesis 1. Furthermore,

if even hybrid design does not satisfy the desired PDP ratio constraint, the different designs, hybrid or original, can also be combined with careful gate sizing and gate V_{th} selection to yield the desired PDP ratios. Not that this cannot be achieved just by gate sizing and/or V_{th} selection, as these physical synthesis transforms do not offer enough delay and power variation to reduce say a 2.06 PDP ratio of a CLA to fall in the desired range, but combined with different design structures, the final FUs can be fine-tuned to have the desired PDP ratios.

The following is a corollary to Hypothesis 2.

COROLLARY 1. *Slow and fast FUs derived via design variation can much more easily meet desired speed and power ranges, and PDP ratios mentioned in Hypothesis 1, compared to applying multi-V_{dd} and multi-V_{th} to a single design.*

Justification: In current technology, V_{dd}'s as well as V_{th}'s have very small ranges, and thus they can affect the delay and power ranges and the PDP ratio (compared to a slow base design) of a given design in only a small way. This is empirically established in the following section.

4.4. Resource Library Construction Examples

We have identified many FU design works like [41] [42] [43] [44] [45] [46] to obtain for several FTs, FUs with different designs that have power and delay characteristics that satisfy the criteria of Hypothesis 1. The power and delay characteristics of FUs with different designs are presented in TABLE V. All of them are based on 0.18μm technology and a V_{dd} of 1.8V. For non-arithmetic FTs such as "SRAM(Write)", "SRAM(Read)", "Shift Register" and "AND", since either design variety does not lead to enough delay range (e.g., in SRAMs that we could find in [46]) or there is almost no design variety available (in "Shift Register" and "AND), in order to obtain dual-speed FUs, we just scale the V_{dd} up to 2.5V for these FTs to get a faster power-delay characterized speed with reasonable delay ranges (see the delay ratio column in TABLE V).

These non-arithmetic FUs, however, has a miniscule effect on power, as: they consume very little power compared to the arithmetic FUs, and about 70% of operations use the first 3 arithmetic FTs in TABLE V. For arithmetic FTs, we provide two different published designs that satisfy the criteria of Hypothesis 1. These FU designs thus empirically prove Hypothesis 2.

We also determined dual-V_{dd} FUs for all the FTs in TABLE VI to demonstrate Corollary 1. For an apples-to-apples comparison of PSA results obtained using dual-design and dual-V_{dd} FUs, we scale V_{dd} so that we achieve similar delay ratios to those for dual-design FUs (as in TABLE V) via a least mean-square error (LMSE) fitting of the delay ratios. The V_{dd}-scaling is based on the facts that the delay ratio is the inverse of the V_{dd} ratio, leakage power change is linearly proportional to V_{dd} change, and dynamic power is quadratically proportional to V_{dd}. Further, we assume that the percentage of dynamic and leakage powers are 60% and 40%, respectively, though the specific percentages assumed do not affect the verification of our hypotheses. From our calculations, V_{dd} needs to be scaled to 2.8V to obtain LMSE fitting of delay ratios to those in TABLE V. The resulting power and delay characteristics of FUs with V_{dd} of 1.8V and 2.8V are shown in TABLE VI. Comparing Tables 1 and 2, we can see that the FUs with different designs have desirable PDP ratios, and FUs with different V_{dd}'s that achieve similar delay ratios to FUs with different designs, have larger PDP ratios. This empirically proves Corollary 1.

TABLE V

POWER-DELAY CHARACTERISTICS OF FUS WITH DUAL DESIGNS

FT	Design	DP (mW)	LP (mW)	Delay (ns)	PDP (pJ)	Delay ratio[c]	PDP ratio[c]
Addition	Slow (RCA) [41]	0.51	0.34	5.86	4.98	0.38	0.96
	Fast (CLA) [41]	1.31	0.87	2.2	4.80		
Multiplication	Slow (PR3a) [42]	17.83	11.89	11	326.92	0.65	0.99
	Fast (tree9to4) [42]	27.07	18.05	7.18	323.96		
Division	Slow (Aqa-SM[a]) [43]	58.8	39.24	55	5392.20	0.65	0.99
	Fast (Aqa-FM[b]) [43]	89.33	59.56	35.9	5345.15		
SRAM (Write)	Slow (6T) [46]	2.10E-08	1.40E-08	0.1	3.50E-09	0.72	1.21
	Fast (2.5v)	4.00E-08	1.90E-08	0.072	4.25E-09		
SRAM (Read)	Slow (6T) [46]	5.50E-07	3.70E-07	0.1	9.20E-08	0.72	1.26
	Fast (2.5v)	1.10E-06	5.10E-07	0.072	1.16E-07		
Shift Register	Slow (MFF) [45]	0.31	0.2	0.23	0.12	0.74	1.26
	Fast (2.5v)	0.59	0.28	0.17	0.15		
And	Slow [44]	3.40E-09	2.20E-09	0.1	5.60E-10	0.72	1.23
	Fast (2.5v)	6.50E-09	3.10E-09	0.072	6.91E-10		

[a] SM = slow multiplier. It is used in the slow Aqa divider.

[b] FM = fast multiplier. It is used in the fast Aqa divider.

[c] The ratio is the corresponding metrics of the fast FU to the slow FU.

TABLE VI

POWER-DELAY CHARACTERISTICS OF FUS WITH DUAL VDDS

FT	V_{dd} (V)	DP (mW)	LP (mW)	Delay (ns)	PDP (pJ)	Delay ratio[a]	PDP ratio[a]
Addition	1.8	0.51	0.34	5.86	4.98	0.64	1.33
	2.8	1.23	0.53	3.77	6.64		
Multiplication	1.8	17.83	11.89	11.00	326.92	0.64	1.33
	2.8	43.14	18.50	7.07	435.88		
Division	1.8	58.8	39.24	55.00	5392.20	0.64	1.33
	2.8	142.28	61.04	35.36	7188.87		
SRAM (Write)	1.8	2.10E-08	1.40E-08	0.10	3.50E-09	0.64	1.33
	2.8	5.08E-08	2.18E-08	0.06	4.67E-09		
SRAM (Read)	1.8	5.50E-07	3.70E-07	0.10	9.20E-08	0.64	1.33
	2.8	1.33E-06	5.76E-07	0.06	1.23E-07		
Shift Register	1.8	0.31	0.2	0.23	0.12	0.64	1.34
	2.8	0.75	0.31	0.15	0.16		
And	1.8	3.40E-09	2.20E-09	0.10	5.60E-10	0.64	1.34
	2.8	8.23E-09	3.42E-09	0.06	7.49E-10		

[a] The ratio is the corresponding metrics of the fast/2.8V FU to the slow/1.8V FU.

4.5. Hypotheses Verification Using PSA

In Section 3.4, we have demonstrated the effectiveness and efficiency of PSA in power optimization with module selection. Thus, it is an appropriate platform for empirically testing our library construction hypotheses proposed in Section 4.3.

For our subsequent experiments on the power results of different libraries, for each DFG, we used a reasonably high-speed latency constraint of $L_c = L_{ASAP}(fastest) + 0.1(L_{ASAP}(slowest) - L_{ASAP}(fastest))$, where $L_{ASAP}(slowest)$ ($L_{ASAP}(fastest)$) is the ASAP latency of the DFG assuming the slowest (fastest) speed for all operations. TABLE VII shows results of 14 media benchmark DFGs (from [37]) of PSA using dual-V_{dd} FUs (see TABLE V) and PSA using dual-design FUs (see TABLE VI). The results show that PSA using dual-design FUs achieves up to 29.19% and an average 16.83% energy improvement (equivalently, power improvement) over PSA using dual-V_{dd} FUs. These results especially show that FUs with different designs that typically have larger power and delay ranges than that of different-V_{dd} FUs, can provide us much better power optimization, thus proving the delay and power range aspect of Hypothesis 1 in Section 4.3.1.

We also constructed more extensive 3- and 4-speed design-based libraries based on the extrapolation of the power-delay characteristics of the arithmetic circuits listed in TABLE V—as we argued earlier and demonstrated in TABLE V, it is possible to design arithmetic functions with a wide range of power and delay, as well as with desired PDP ratios via hierarchically constructed hybrid designs using different known design approaches at different levels of the hierarchy. Our intention here was to test our hypotheses not only between multi-design and multi-V_{dd} libraries, but also between different multi-design libraries that either satisfy or do not satisfy the PDP ratio aspect of Hypothesis 1. To that end, we constructed a 4-speed good" library (the PDP ratios of all the three faster FUs to the slowest FU for each FT are in the range of [1,

1.25]) that satisfies Hypothesis 1 and a 4-speed library with one "bad" FU (the PDP ratio of a faster FU to the slowest FU for each FT is in the range of [1.5, 2]) per FT that does not satisfy Hypothesis 1. Thus, the latter library does not satisfy Hypothesis 1 in only a minor way for only 25% of the speeds, but as we will see in the results, this causes a significant increase in power of the corresponding designs compared to using the 4-speed good library. We stressed Hypothesis 1 verification further by constructing a 3-speed library with the same delay and power ranges as the 4-speed libraries, for an apples-to-apples comparison, in which all FUs satisfy Hypothesis 1 and are a proper subset of the FUs in the 4-speed good library. The motivation is to check if our hypothesis holds up for an all-good but smaller design space (3-speed library compared to 4-speed library) than that of the 4-speed 1-bad library. TABLE VIII shows the results of the comparisons between the three libraries. The experiments show that PSA using either the good 4-speed or good 3-speed library gives significant power improvements of 17.26% and 10.57% in average, respectively, over the 4-speed library with only one bad speed per FT. Another interesting phenomenon can be seen in the "Bad FU" column, which gives the number of bad FUs chosen for each DFG by PSA for the 4-speed 1-bad-FU library. For six DFG's, PSA correctly does not choose any bad FU, but still suffers a power deterioration, though almost always in single-digit percentages (in 9 out of 12 comparisons across these 6 DFG's), compared to the other two libraries. This most probably is due to the reduction of available delay and power ranges in some cases, and the general loss of solution space, resulting from ignoring the bad FUs. However, when PSA chooses one or more bad FUs, for many DFG's it has a 20+% to 30+% power deterioration compared to the 4-speed all-good library. This bolsters Hypothesis 1 further:

When a module-selection algorithm is smart enough not to choose any bad FU of a library not satisfying Hypothesis 1, it still has power deterioration compared to a library satisfying Hypothesis 1 due to the relative loss of solution space, and when the algorithm is not so smart and chooses a few bad FUs, it suffers even greater power increase. Further, for reasonably high-speed (but not the highest possible speed) designs with latency constraints that are a little above the smallest critical path delay of the DFG (critical path delay assuming operations delays are those of the fastest FUs in the library), it is almost always better for power optimization to have a smaller library with FUs that satisfy Hypothesis 1 versus a larger library where some FUs (that can include the fastest FUs) do not satisfy the hypothesis.

TABLE VII

TOTAL ENERGY (EQUIVALENTLY, POWER) RESULTS FOR PSA USING DUAL-VDD FUS AND PSA USING DUAL-DESIGN FUS

DFG	size[a]	L_c	Dual-V_{dd}			Dual-Design		
			Energy (pJ)	Slow FUs	Fast FUs	Energy (pJ)	Slow FUs	Fast FUs
hal	11, 8	34	2322	4	2	2252	4	1
horner	18, 16	73	3674	1	4	2966	2	2
arf	28, 30	85	7096	1	2	5779	1	2
motion	32, 29	23	7177	10	9	5809	9	7
ewf	34, 47	130	2933	2	1	2794	2	1
h2v2	51, 52	109	1484	3	3	1155	5	1
feedback	53, 50	28	8615	19	5	6284	13	4
epic	56, 73	92	10750	4	4	9320	5	3
bmp	106, 88	89	8068	8	8	6670	6	3
aux	108, 104	32	17969	20	16	13935	21	8
mul	109, 116	71	16678	12	8	15186	14	2
idcot	114, 164	115	13731	10	13	11967	16	4
jpeg	134, 169	58	18711	31	17	13250	24	5
smooth	197, 196	54	41653	39	29	32836	31	16
Avg	75.1, 81.6	70.9	11490.1	11.7	8.6	9300.2	10.9	4.2

[a] DFG sizes are indicated by the number of operations and the number of dependency arcs.

TABLE VIII

POWER RESULTS AND FU STATISTICS FOR PSA WITH 4-SPEED-ALL-GOOD LIBRARY, PSA WITH 3-SPEED-ALL-GOOD LIBRARY AND PSA WITH 4-SPEED-ONE-BAD LIBRARY

DFG	L_c	4-Speed-All-Good		3-Speed-All-Good		4-Speed-1-Bad	
		Total Power (10^5 units)	Improv. vs 4-Sp-1-Bad	Total Power (10^5 units)	Improv. vs 4-Sp-1-Bad	Total Power (10^5 units)	Bad FUs
hal	71	171.2	23.37%	202.1	9.53%	223.4	1
horner	153	205.5	16.16%	219.9	10.28%	245.1	1
arf	181	430.1	9.26%	463.2	2.28%	474.0	0
motion	88	398.3	22.59%	453.1	11.93%	514.5	0
ewf	283	239.4	17.53%	269.3	7.23%	290.3	0
h2v2	243	105.5	0.19%	105.7	0.00%	105.7	0
feedback	106	453.2	9.70%	494.5	1.47%	501.9	0
epic	112	472.0	30.65%	526.6	22.63%	680.6	1
bmp	107	272.0	36.55%	291.9	31.91%	428.7	1
aux	123	1030.8	10.33%	1094.4	4.80%	1149.6	1
mul	155	1093.2	11.14%	1165.1	5.29%	1230.2	4
idcot	244	786.2	6.79%	845.0	-0.18%	843.5	0
jpeg	223	986.2	12.02%	1058.1	5.61%	1121.0	1
smooth	204	1810.5	24.86%	1949.1	19.10%	2409.4	4
Avg	163.8	603.9	17.26%	652.7	10.57%	729.9	1.0

5. LPR-GPS: NEW DIRECTION FOR ENERGY MINIMIZATION

The work in this chapter has been published in [4].

In this chapter, we propose a scheduling algorithm LPR-GPS with a single-speed library to minimize the total leakage energy (LE) of a computation (the ME-LCS problem), which is the most important power/energy metric to minimize in systems that do not operate continuously. This problem is motivated further in Section 5.2 after the energy model is formulated in Section 5.1.

Our LPR-GPS algorithm is still able to obtain significant total LE reduction without considering orthogonal power/energy reduction techniques like module selection of different power-speed characterized FUs for each function [8] [9] [10], bus switching probability reduction [11] and power gating [12]. It thus provides a good optimization starting point for other low-power techniques (which are all orthogonal to our approach) to achieve better results than they do otherwise. For example, techniques [8] [9] [10] that switch low-V_{th} and/or high-V_{dd} FUs on non-critical paths to high-V_{th} and low-V_{dd} FUs, respectively, to reduce power but increase the path latencies close to the latency constraint, can be combined with latency space exploration similar to that in our algorithm, so that a good sweet spot of less than the most aggressive low-power V_{th}/V_{dd} reassignment and a latency smaller than the latency constraint is reached where the total LE is minimized.

We then give the detailed formulation of our LPR-GPS algorithm in Section 5.3, where the following major improvements of LPR-GPS compared to FDS are detailed:

- An initial probabilistic distribution graph of the number of functional units used in each control step based on a first-pass non-uniform probability driven randomized scheduling that yields the final starting probabilities that are conducive to LE minimization;
- A root-mean-square based estimation of the maximum functional unit usage distributed across control steps that contributes to minimizing either the overall unit-time leakage power or total leakage energy;
- a fast and greedy noncommittal scheduling algorithm for the purpose of estimating the latency for scheduling output operations in order to minimize the product of estimated unit-time leakage power and estimated latency.

Experimental results on 11 media benchmarks demonstrate a total LE reduction of up to 64% and an average of 44% compared to conventional FDS with a power-driven modification that only minimizes unit-time LP (which is also what other low-power approaches do), and a total LE reduction of up to 39% and an average of 12% compared to a version of our algorithm that has the first two aforementioned improvements, but does not explore the latency space for total leakage energy minimization. This demonstrates the efficacy of co-exploring unit-time leakage power and latency spaces for leakage energy minimization.

5.1. Energy Model

We formulate the LE minimizing scheduling problem as follows.

First, we define *unit LP* LP_{unit} as the amount of leakage energy expended per cc by the HLS design:

$$LP_{unit} = \sum_{k \in all\ FTs} lp(k) \times num(k) \tag{5.1}$$

where lp(k) is the leakage energy consumption per cc of FUs of FT *k*, and *num(k)* is the number of allocated FUs of FT *k*. Note that while LP_{unit} is strictly speaking an energy metric and not a

power one in the conventional sense (i.e., it is not energy per second), we have preferred to use the term "power" in it for purposes of distinguishing it from the term "total LE", and the fact that it is still energy per some unit time (which is not necessarily a second).

Then, we define *total LE* LE_{total} is the total amount of leakage energy expended by the design per computation:

$$LE_{total} = LP_{unit} \times L \tag{5.2}$$

where $L \leq L_c$, the latency constraint, is the achieved latency of the scheduling solution. Total LE is the minimization objective of our LPR-GPS algorithm.

Finally, we note that the total dynamic energy DE_{total} expended by the design is:

$$DE_{total} = \sum_{u \in V} dp(u) \times d(u) \tag{5.3}$$

where *dp(u)* is the dynamic energy consumed per cc of FUs of the FT that can execute operation *u* and *d(u)* is the operation delay of *u*. Since operations are dependent on the DFG of the design and there is only one speed per FT, thus *d(u)* and *dp(u)* are constants for a given single-speed library and DE_{total} is independent of the scheduling solution, i.e., the dynamic energy consumed by an operation is identical for all scheduling solutions. Therefore, the only component of energy consumed per computation in a HLS design with a single-speed library, that can be minimized in a scheduling solution is the total LE.

5.2. Motivation of Leakage Energy Minimization

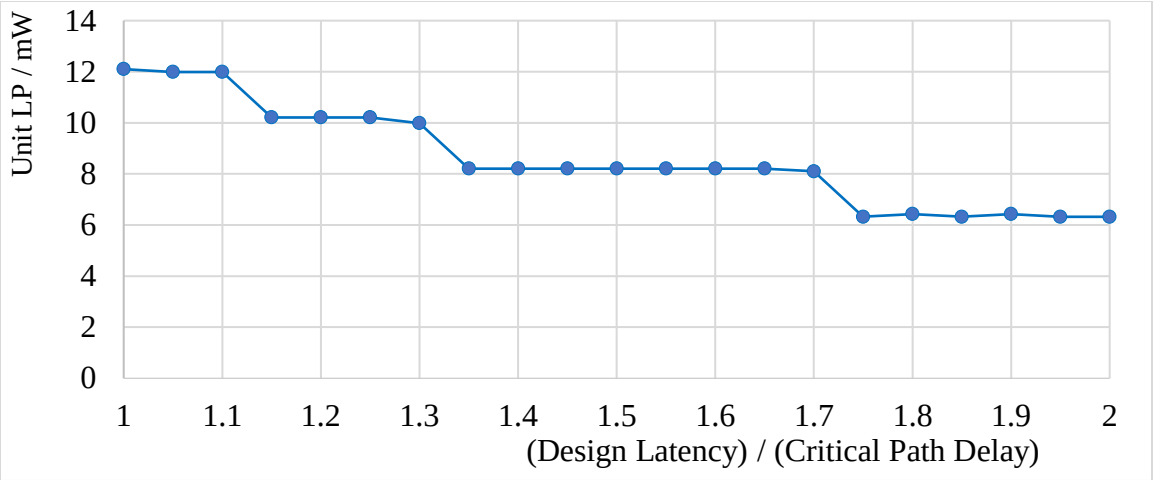
Current digital designs generally have a target speed determined by the overall application they are a part of. They need to operate by consuming as little energy as possible and not necessarily as fast as possible. Our work in this chapter that minimizes total LE of a computation processed by a digital system, either ASIC or FPGA-based, is especially applicable to systems that perform one or a stream of computations for a period of time and then become

idle for some time, for which point they can be powered off (power-gated) or put in sleep-mode (for example, a low V_{dd} state) before it "wakes up" at a later time to perform another stream of computations. Examples of such systems are embedded systems that are only active when controlling a larger system that works sporadically (for example, implanted drug delivery systems and cell phones) and hardware accelerators in a system-on-chip that are used occasionally [47]. In such an operating scenario, it is important to minimize the LE consumed per computation so that total energy consumption over the lifetime of the digital system is minimized, or, alternatively, for a battery power device, the battery-charge lifetime is maximized. Minimizing LE per computation is equivalent to minimizing the product of unit LP and the achieved latency *L* of the computation. These considerations apply to both non-pipelined and pipelined systems—in the latter, the latency comes into the picture as the fill-time of the pipeline, and can represent a significant portion of the total LE for short bursts or streams of computation. Conventional scheduling algorithms exploit the entire latency constraint for the computation to reduce the number of FUs needed in the corresponding HLS design, or to reduce power (examples of such algorithms include those that use multiple supply or threshold voltage techniques to replace fast but high power-consuming FUs with slow ones on non-critical paths). However, as we show below, minimizing power does not minimize computation energy, which, as we have discussed above, is the important objective to minimize in not-always-in-use but always powered-on systems and devices that contain the digital system whose energy consumption we aim to minimize. To minimize computation LE, the product of unit LP and latency needs to be minimized. However, latency is rarely considered as a part of an energy optimization objective. To the best of our knowledge, our work is the first one that considers this aspect explicitly for LE minimization.

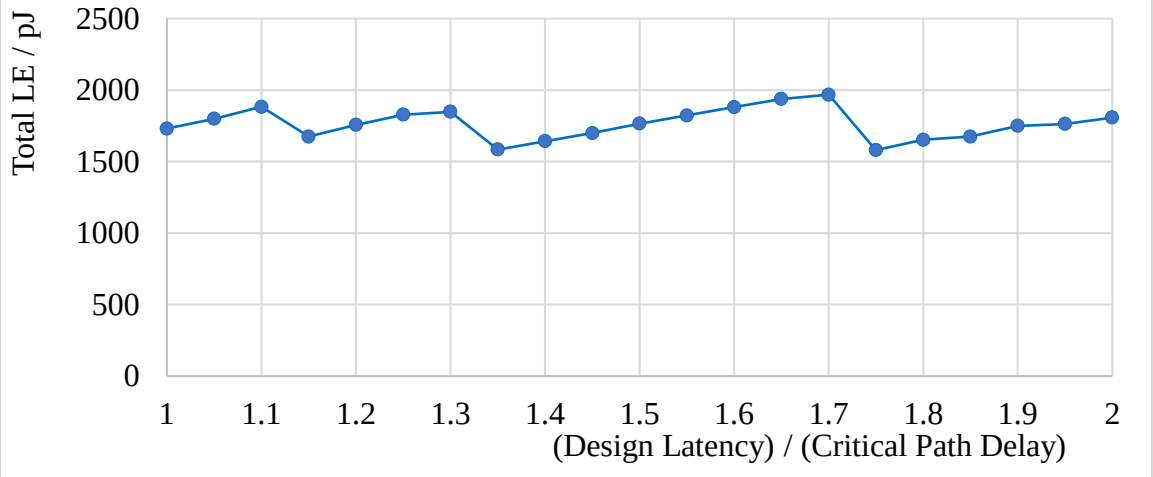
As we mentioned above, most digital systems do not need to perform their computations as fast as possible, but instead have a target speed and need to minimize power/energy as much as possible. This indicates that the latency constraints for such designs will be larger than the critical path delay of the DFG of the corresponding computation. Thus, in our experiments, we use a latency constraint that is somewhat larger, by factors ranging from 1.5 to 2 of the critical path delay of the DFG, which is the minimum achievable latency in a corresponding design.

In Figure 13, we illustrate the optimization space that is available for minimizing the total LE of a computation with a given latency constraint L_c. Using as an example of LP minimization algorithms, a sophisticated but latency-agnostic LP minimization (not total LE minimization) algorithm PR-GPS that we will discuss in Section 5.3.2, we plotted both unit LP and total LE of a DFG idctcol with different latency constraints in Figure 13(a) and (b), respectively. The latencies are given as factors of the critical path delay of the DFG, which is in the range [1, 2] with a granularity of 0.05. Similar plots were obtained for other DFGs, and we present the idctcol plots as representative of a typical DFG. Since PR-GPS only minimizes unit LP, as expected, the final latencies of the designs it synthesizes is exactly or very close to the given L_c's (as using as much the available latency as possible minimizes the number of FUs and thus LP). Thus, the plots in Figure 13 are essentially the unit LP and total LE of different designs for the same DFG but with different latencies. As expected, the unit LP is monotonically non-increasing as the design latency increases as the increased latencies provide opportunities to increase resource sharing and hence reduce the number of high power-consuming FUs. However, total LE fluctuates with design latency increasing, and the best results are frequently seen at somewhat lower latency designs. For example, for an L_c factor of 1.6, the minimum total LE is obtained for a latency factor of 1.35. Therefore, to achieve better total LE minimization, an algorithm should

be able to determine such smaller latencies, if they exist (they will not always exist, as the plot also shows—for example, for an L_c factor of 1.35, there does not exist a lower-latency design with lower LE), rather than exploiting the entire latency constraint to minimize the unit LP and expecting the total LE to be minimized as well.



(a)



(b)

Figure 13. (a) Unit LP results and (b) total LE results obtained by PR-GPS under different latency constraints for the DFG idctcol.

Finally, we note that once scheduling is completed, it is straightforward to bind operations to actual FUs if there are no other considerations except the optimization metric and the latency constraint that were considered during scheduling. This can be achieved by scanning the scheduled operations in increasing order of cc's, binding them to available FUs (at that point in the binding process) and allocating new FUs if all FUs of that FT are busy, so that the number of active FUs in each cc for each FT is exactly the number of FUs on which operations are executed as determined after scheduling. This is essentially the same as using the polynomial-time left-edge algorithm [48] for minimizing the number of colors in an interval graph that can be used to model operation execution in FUs across cc's.

5.3. Our Scheduling Algorithm for Leakage Energy Minimization

In this section, we present our leakage energy optimization algorithm that explores the dimension of minimizing the product of unit-time leakage power (unit LP) and latency. The algorithm can be coupled with any other leakage and dynamic power optimizing techniques [8] [9] [10] [11] [12] as it does not compete with these existing techniques but supplements them to achieve a better energy optimization than can be achieved by these techniques alone.

The main beneficial aspect of FDS, which we retain in our algorithm, is its global cognizance of the min-max optimizing metric's estimated value based on all scheduling options as captured by the *DG(k)* for each FT k. This min-max metric is the number of FUs in FDS and it is unit LP in our case. Using this global information, in each scheduling iteration, in which it schedules one unscheduled operation, FDS evaluates each scheduling option via the self and PS forces (see Section2.5.3 and 2.5.4) to choose a scheduling option that minimizes the total force and, according to FDS's rationale, the minimization of the corresponding expected total number of FUs across all FTs. However, as we analyzed in Section 2.5.5, there are a few drawbacks in

FDS for optimizing a min-max objective. We have made significant modifications to FDS to rectify these drawbacks, as well as to optimize a non-min-max objective such as total LE (which has as a component a max function, unit LP, for each *DG(k)*) as follows.

Firstly, instead of associating the *DG(k, i)* values, where *k* is the FT and *i* is the cc, as spring constants and the probability changes Δ*pu(i)* as displacements, thus bringing the concept of force = (spring constant) × (displacement) that needs to be minimized as in [13] [14], we consider the *DG(k, i)* values across the 2-dimensional cc and FT space as a global probability map (GPM) of all possible scheduling events, and we choose the best scheduling event to schedule so that a more direct estimate of Equation (5.2) than the forces is minimized. This and other issues of our new GPM based algorithm are explained in detail in the subsequent discussions.

1. The use by self and PS forces in FDS to minimize Equation (5.5) is somewhat flawed, and as one of our major improvements to FDS, we modify this formulation in Section 5.3.2 to get a more accurate estimate of the final value of the min-max objective of interest (number of FUs in FDS or, in our case, unit LP, which is part of minimizing total LE) for each FT across all cc's.
2. Another drawback in FDS is the assumption of uniform scheduling probabilities across all cc's in the mobility range (MR) of an operation. We rectify this in Section 5.3.1 with a more accurate scheduling probability determination and hence more accurate *DG*(*k, i*)'s that are conducive to the min-max optimization metric of interest.
3. Finally, we develop our most significant extension of an FDS-type evaluation of scheduling options for LP minimization, in which, along with unit LP estimation, we heuristically

estimate the latency of the final solution for each scheduling option, and thereby estimate the total LE.

We then choose the scheduling option that minimizes the product of estimated latency and estimated unit LP, the total LE. Note that earlier LP minimization HLS techniques [9] [10] [38] only minimized the unit LP assuming that the achieved latency will be the same as the latency constraint. In fact, empirically, they use up all or almost all the latency space available to get their final solution as this minimizes unit LP; however, this does not minimize total LE as discussed in Section 5.2. Furthermore, since the latency estimation part is somewhat time-consuming, we reasoned that:

1. We really need to do this estimation for scheduling only *output operations* (operations that have no successors), as the maximum latency only across all output operations is the final latency of the design.
2. We can also schedule all output operations first, thereby obtaining a pre-determined latency for the partial design in which only output operations are scheduled, and which probabilistically minimizes the total LE while satisfying the latency constraint. We can then schedule the rest of the operations without the latency estimation, that is, the design latency is now a pre-determined constant and we only minimize unit LP. This allows us to obtain the solution faster with little impact on the solution quality.

Our experiments have borne out the efficacy of the above approach. The three main steps of our algorithm are summarized as follows:

1. Determine more accurate low-energy oriented scheduling probabilities and thus *DG(k, i)*'s by an initial low unit LP driven randomized scheduling process coupled with a statistical analysis

of the results. Note that unit LP can be substituted by any other min-max metric of interest like number of FUs or total area to obtain accurate DGs for that metric using our approach.

2. Use a more accurate selective root-mean-square (RMS) based estimate of the final maximum DG values across FTs for each scheduling option. This gives us a highly accurate unit LP estimate for each scheduling option.
3. Scheduling output operations first by evaluating both the estimated unit LP and the estimated latency for their scheduling options using a greedy and quick noncommittal scheduling of unscheduled output operations, and choosing the scheduling option that minimizes their product, i.e., the estimated total LE of the design. After all output operations are scheduled and thus the probabilistically minimized total LE is determined, the remaining operations are scheduled based on minimizing only the estimated unit LP.

These ideas and the steps of our algorithm which we call LPR-GPS (for Latency times unit Power minimization via RMS-driven Global Probability map based Scheduling) are explained in subsequent subsections in greater detail. The pseudo code for LPR-GPS is given in Figure 14.

Algorithm LPR-GPS(DFG $G(V, E)$, latency constraint L_c, single-speed library K)
1. Determine low-energy oriented scheduling probabilities (refer to Section 5.3.1)
2. **While** there are unscheduled output operations **Do**
3. Update t^{ASAP} and t^{ALAP} of unscheduled output operations
4. **For** each scheduling option of output operations **Do**
5. Perform noncommittal greedy scheduling (refer to Section 5.3.3)
6. Calculate the unit LP estimate LP_{unit} and latency L estimates and the corresponding total LE estimate $LE_{est} = LP_{unit} \cdot L$ of the greedy scheduling solution
7. **End for**
8. Choose the scheduling option with the smallest LE_{est} to schedule
9. **End while**
10. **While** there are unscheduled operations **Do**
11. Update t^{ASAP} and t^{ALAP} and scheduling probabilities of unscheduled operations
12. **For** each scheduling option **Do**
13. Calculate RMS-based unit LP estimate LP_{unit} (refer to Section 5.3.2)
14. **End for**
15. Choose the scheduling option with the smallest LP_{unit} to schedule /* Note that the latency has already been determined after output node scheduling, and thus at this point LP_{unit} minimization is equivalent to total LE minimization */
16. **End while**

Figure 14. The pseudo code of LPR-GPS algorithm.

5.3.1. Energy-Driven Scheduling Probability Determination

We determine the energy-driven scheduling probabilities via first-pass unit LP based probability driven Monte Carlo simulation to obtain multiple randomized scheduling solutions. From these, the final energy-driven probabilities are determined to construct the DG. The main idea is that as a first-pass probability, we want to assign a low probability to a scheduling option that is more likely to increase the unit LP in the scheduling solution, to lower the chance that it is chosen to schedule. On the contrary, a good scheduling option that is more likely to lower or to keep the unit LP unchanged should be assigned with a high probability. The uniform scheduling probability assumed by FDS does not achieve this, and is a drawback of FDS. We then perform multiple randomized schedulings based on these first-pass probabilities and select those solutions that have low LE, and determine second-pass scheduling probabilities based on these solutions. These serve as the initial probabilities for DG construction, and our deterministic LE

minimization algorithm that co-explores unit LP and latency spaces for minimizing their product. According to our experiments, just this change of initial scheduling probabilities achieves an average of 4.2% total LE reduction compared to having uniform scheduling probabilities. Since the scheduling probability determination is orthogonal to the issue of force formulation, it can also be incorporated with other FDS-like algorithms for improving their solution quality.

An example to illustrate the rationale of using energy-driven scheduling probabilities based on several scheduling solutions is given in Figure 15. For scheduling the two addition operations with a delay of one ccc shown in Figure 15(a) into 3 cc's, there are three possible solutions as shown in Figure 15(b). Since the two additions are executed in sequence, all solutions need one adder and hence their unit LP's are the same. However, by Equation (5.2), their total LE is different due to the difference in the achieved latency, i.e., the two solutions on the right in Figure 15(b) have 1.5 times of the total LE than the one on the left, and hence should be avoided by the scheduler. Furthermore, according to Equation (2.8), the scheduling probabilities of op1 and op2 in cc 1 and cc 2 are all 0.5. Such uniform scheduling probability does not differentiate between scheduling options that correspond to good or bad scheduling solutions, and thus does not help to achieve the total LE minimization goal. A good scheduling probability formulation, as we have in our energy-driven scheduling probability formulation, assigns probabilities for op1 in cc 1 and op2 in cc 2 that are much higher than that for op1 in cc 2 and op2 in cc 3. This makes the scheduling option corresponding to low energy solutions more likely to be chosen in the final scheduling solution.

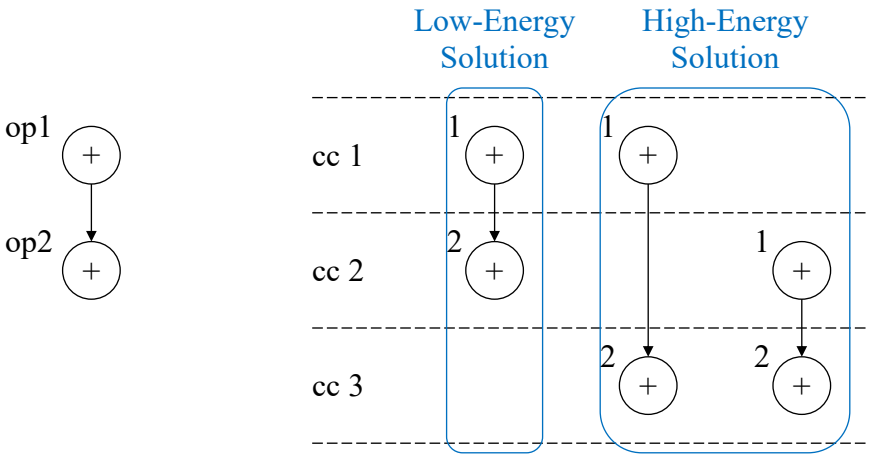


Figure 15. (a) An unscheduled DFG with two operations op1 and op2. (b) All three solutions to schedule the DFG subject to a latency constraint of 3 cc's.

An outline of our energy-driven scheduling probability formulation is as follows. We perform several energy-driven random schedulings and derive the energy-driven scheduling probabilities from the statistics of low-energy scheduling solutions. The randomization is guided by first-pass version of low unit LP oriented scheduling probabilities, called *modified probabilities*. The final probabilities determined from the statistics of low-energy randomized scheduling solutions are used for DG construction (refer to Section 2.5.2).

A DG with uniform scheduling probabilities is constructed first for each FT in the same way of FDS. For each unscheduled operation, the modified scheduling probability of a scheduling option is inversely proportional to the DG values of the corresponding FT in its MR, since the objective of FDS-like algorithms is to minimize the maximum value of a metric (i.e., for optimizing a min-max objective metric) for that FT. For a single-cc operation u of FT k, if its MR is MR_u, its modified scheduling probability $mp_u(i)$ in cc i is:

$$mp_u(i) = \frac{\frac{1}{DG(k,i)}}{\sum_{j \in MR_u} \frac{1}{DG(k,j)}} \tag{5.4}$$

This formulation is extended for multi cc operations as follows. For a multi-cc operation *u* of FT *k*, if its delay is d_u and mobility range is MR_u, its energy-driven random scheduling probability $mp_u(i)$ in cc *i* is:

$$mp_u(i) = \frac{\frac{1}{EDG(k,i)}}{\sum_{j \in MR_u} \frac{1}{EDG(k,j)}} \tag{5.5}$$

where *EDG(k, i)* is the effective DG (EDG) value of FT *k* in cc *i*. The EDG is meant to capture an average-type function of the DG values in the delay range $[i, i + d_u - 1]$ of an operation *u* with delay d_u scheduled in cc *i*, which gives more prominence to higher DG values, since we use it to determine scheduling probabilities that are inversely proportional to the high DG value(s) in its delay range. Once again, this is desirable so that we schedule to minimize the maximum DG value of the final scheduled solution. The EDG value is formally defined as:

$$EDG(k,i) = \underset{m \in [i, i+d_u-1]}{RMS}\big(DG(k,m)\big) \tag{5.6}$$

where *RMS* is the root-mean square function defined as:

$$\underset{m \in [1,n]}{RMS}(x_m) = \sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_n^2}{n}} \tag{5.7}$$

where x_m is a numerical value.

We prefer RMS over arithmetic average here and in the RMS-based power estimate formulation to be discussed later, because it assigns higher than a linear weight to larger values in the set. Since the unit LP is very likely determined by the current largest DG values among the set of considered DG values, RMS average allows us to focus on the maximum and near-

maximum values while also considering some effect of other lower values. Note that, any of the maximum and near-maximum current DG values have good likelihoods of becoming the final maximum DG values (across all FTs) at the end of scheduling, and thus determining the final unit LP. It is thus useful to not just take the maximum value but a higher-value prioritized average, such as RMS, of these DG values. Otherwise, for example, we will not be able to distinguish between DG value subsets that: a) have several near-max values and thus have a high likelihood of the breaching the current max DG value during subsequent schedulings and determining the final unit LP, and b) have very few or no near-max values, making it much less likely to lead to the final max DG value and thus determining the final unit LP. The RMS value of the latter subset is lower than the former, and the latter thus represents a more preferable subset of cc's to schedule an operation in to minimize the final unit LP.

After determining the modified scheduling probabilities, several energy-driven randomized scheduling solutions are generated based on these probabilities. Each solution is obtained by scheduling operations in a topological order, randomly choosing one cc *i* in their mobility ranges with a probability of $mp_u(i)$, until all operations are scheduled. In our experiments, the number of energy-driven random scheduling solutions we generate is the product of latency constraint L_c and the number of operations $|V|$ in order to obtain a reasonable sample size of randomized solutions for problems of different sizes, and thus more accurate energy-driven final scheduling probabilities with which to construct the DG.

Not all energy-driven random scheduling solutions are accepted for final scheduling probability determination: only the top 10% of low total LE results are considered to calculate the energy-driven scheduling probabilities. If there are *N* such low-energy random scheduling solutions, and the scheduling option of scheduling operation *u* in cc *i* appears $n(u, i)$ times among

all the results, we determine the final scheduling probability to schedule operation *u* in cc *i* as:

$$p_u(i) = \frac{n(u,i)}{N} \tag{5.8}$$

A caveat is that it is possible to miss some *m* scheduling options of an operation in the set of low-energy random scheduling solutions, in the case that such scheduling options only appears in high-energy solutions. For a DFG and a library that provide sufficient low-energy random scheduling solutions, a quick fix is to assign the lowest scheduling probability of *1/N* to each such scheduling options and proportionally scale the other scheduling probabilities of the same operation by a factor of *(1 – m/N)* so that the sum of scheduling probabilities for the operation is 1. This maintains a complete solution space while ensuring a low likelihood for a bad scheduling option to be chosen during the actual scheduling phase.

5.3.2. RMS-Based Power Estimation

The RMS-based power estimate minimizes the sum of unit LP's of all FTs during scheduling by keeping its increment as low as possible, or on the flip side, making its decrement as high as possible. Our RMS-based power estimation function gives maximum and near-maximum DG values higher prominence, since at least one of these values are most likely to be the maximum ones after all operations are scheduled. At the same time, in order to reduce "noise", it ignores DG values that are smaller than a threshold fraction of the current maximum DG value for the corresponding FT.

We first observe that the force formulation in FDS is not strongly correlated with minimizing the maximum DG values, the min-max goal. Based on the analysis of this drawback discussed in Section 2.5.5, we propose a root-mean-square (RMS) based power estimate that is proven to yield better results than the FDS force formulation. A conceptual example is given in Figure 16. In a particular scheduling iteration, consider two single-cc delay operations, *u* and *v*,

whose *MR's* are [i, i + 1] and [j, j + 1], respectively. Besides the four cc's in the MRs, there is another cc shown in Figure 16 between the two, whose DG value is 9. The current maximum DG value in the DFG is 10. Between *u* and *v*, there are four scheduling options available: *u* in *i*, *u* in *i* + 1, *v* in *j* and *v* in j + 1. For the sake of clarity, we only calculate self-forces in this example, as PS-forces have a similar effect. The FDS forces of the four scheduling options are:

$$SF_u(i) = 10(0.5) + 8(-0.5) = 1, \quad SF_u(i+1) = -1,$$

$$SF_v(j) = 3(0.5) + 6(-0.5) = -1.5, \quad SF_v(j+1) = 1.5.$$

Clearly, scheduling of *v* in cc *j* is chosen by FDS in the current iteration, since it has the smallest force.

Calculating the RMS-based power estimates of the four scheduling options with a threshold fraction of 0.9, i.e., only DG values ≥ 0.9 $DG_{max}(k)$ are included in the RMS function, by Equation (5.7), we get:

$$P_{unit}(u,i) = RMS(10.5) = 10.5, \quad P_{unit}(u,i+1) = RMS(9.5,9) = 9.25,$$

$$P_{unit}(v,j) = RMS(10,9) = 9.51, \quad P_{unit}(v,j+1) = RMS(10,9) = 9.51.$$

Based on the RMS formulation, our algorithm chooses to schedule *u* in cc i + 1 in the current iteration. For the min-max goal, the scheduling option chosen by our algorithm reduces the max DG value by 0.5 while FDS's choice does not change the max DG. As we discussed in Section 2.5.5, the problem with the FDS force formulation is that it essentially captures the difference between the DG value of the to-be-scheduled cc and the average DG value of all other cc's in the MR of each unscheduled operation being evaluated, while any estimate of the increase or decrease in the maximum DG value(s), post-scheduling, is not accounted for unless the maximum DG value lies in the MR of an operation. Further, as the example above illustrates, across the forces of all scheduling options, the least force will generally not correspond to the

largest decrease or smallest increase of the maximum DG value due to the aforementioned property of what the force measures. Thus, the scheduling decisions are actually not made based on this most important consideration and hence FDS cannot guarantee a good approximation of the min-max goal.

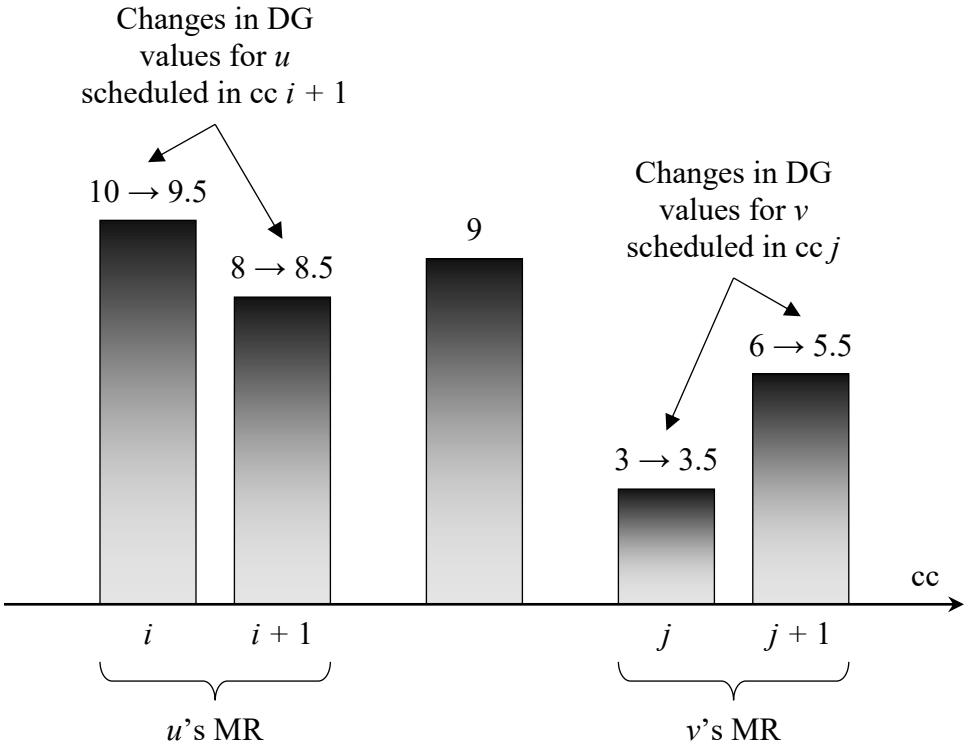


Figure 16. An example to illustrate that the RMS-based metric estimate is an improvement over FDS force for a min-max goal.

In our RMS-based power estimate, a power distribution graph (PDG) is introduced to estimate the unit LP distribution across cc's for each FT. As we mentioned in Section 2.5.2, the maximum DG value $DG_{max}(k)$ for FT k operations at any stage in the scheduling process gives us a probabilistic estimate of the number of FUs of that FT, since it has the highest likelihood to become the maximum final DG value, which determines the final number of FUs of FT k needed after all operations have been scheduled. For the sake of unit LP estimation, we transform the DG value $DG(k, i)$ of FT k in cc i to PDG value $PDG(k, i)$ as:

$$PDG(k,i) = lp(k) \times DG(k,i) \quad (5.9)$$

where *lp(k)* is the leakage energy of FT *k* per cc. Intuitively, the maximum PDG value $PDG_{max}(k)$ is the probabilistic estimate of the unit LP of FUs of FT *k*. A straightforward use of PDG values instead of DG values in FDS yields a straightforward power-driven FDS (PFDS) algorithm for unit LP minimization to which we compare our LPR-GPS algorithm.

In our selective RMS formulation for determining the goodness of a scheduling option for minimizing unit LP, we only consider a few PDG values. There are two sets of candidate PDG values to be included in the RMS function: a local set and a global set. The former includes some candidate PDG values that are directly affected by the scheduling option, while the latter includes certain values that are indirectly affected. It is desirable to include more candidate PDG values in the local set than in the global set of the RMS-based unit LP estimation, since the PDG value changes caused directly by a scheduling option are more deterministic as compared to the more probabilistic nature of the changes in the global PDG values due to shrinking of the MRs of predecessors and successors of the operation whose scheduling option is being evaluated.

The criteria to pick candidate PDG values when a scheduling option is evaluated are as follows. For a scheduling option to schedule operation *u* of FT *k*, we define the local range R_{local} of *u* to be PDG values of FT *k* in the range $MR_u \cup DR_u(t_u^{ALAP})$, which is $[t_u^{ASAP}, t_u^{ALAP} + d_u - 1]$, where recall that t_u^{ASAP} and t_u^{ALAP} are the earliest and latest cc's for scheduling *u*, respectively, and d_u is the delay of *u*. The local candidate PDG value set $S_{local}(u, i, k)$ is formulated as:

$$S_{local}(u,i,k) = \{PDG(k,i) | PDG(k,i) \geq \alpha \times PDG_{max}(k), PDG(k,i) \in R_{local}\} \quad (5.10)$$

where $0 \leq \alpha \leq 1$ is the local threshold fraction, and $PDG_{max}(k)$ is the maximum PDG value of FT *k*. For the same operation, we define the global range $R_{global}(k')$ to be PDG values of FT *k'* that is

in the range [1, L_c] if $k' \neq k$, and in the range [1, L_c] – R_{local} if $k' = k$. The global candidate PDG value set $S_{global}(u, i, k')$ of a FT k' is formulated as:

$$S_{global}(u,i,k') = \{PDG(k',i) | PDG(k',i) \geq \alpha' \times PDG_{max}(k'), PDG(k',i) \in R_{global}(k')\} \quad (5.11)$$

where $0 \leq \alpha' \leq 1$ is the global threshold fraction. To include more candidate PDG values from the local set, we set $\alpha < \alpha'$. Also, both local and global threshold fractions should gradually increase with more operations scheduled, as rationalized below. The dynamic threshold fraction formulation for α is given as:

$$\alpha = \alpha_c + \frac{N_{schd}^2}{N_{unschd}^2}(1 - \alpha_c) \quad (5.12)$$

where α_c is the initial local threshold fraction, N_{schd} and N_{unschd} are the numbers of scheduled operations and unscheduled operations, respectively, at any stage in the scheduling process. There is a similar dynamic fraction formulation for the global threshold α'. This dynamic fraction formulation matches the fact that with more operations scheduled, the PDG values become more deterministic and the likelihood for smaller PDG values to become the maximum value at the end of scheduling decreases. Therefore, the size of both S_{local} and S_{global} become smaller, and they eventually only contain the maximum PDG values after all operations are scheduled. Our experiments show that the best average results are obtained when $\alpha_c = 0.7$ and $\alpha_c' = 0.8$.

Finally, the RMS-based unit leakage power estimate $LP_{unit}(u, i, k)$ for scheduling operation u of FT k in cc i is:

$$LP_{unit}(u,i,k) = \underset{PDG(k,j) \in S_{local}(u,i,k) \cup S_{global}(u,i,k)}{RMS} PDG(k,j) + \sum_{k' \in K - \{k\}} \underset{PDG(k',j) \in S_{global}(u,i,k')}{RMS} PDG(k',j) \quad (5.13)$$

where *K* is the set of all FTs in the DFG.

We term the scheduling algorithm that incorporates the new scheduling probabilities discussed in Section 5.3.1 and RMS-based power estimate to schedule all operations discussed above, without any latency estimation and thus without minimizing the unit LP and latency product, as PR-GPS (for unit Power minimization via RMS-driven Global Probability map based Scheduling).

5.3.3. Greedy Scheduling for Latency Estimation

As mentioned earlier, we schedule all output operations first in a way that the product of the estimated unit LP and estimated latency is minimized. Unit LP estimation was described in Section 5.3.2. Latency estimation is relatively more complex. For this purpose, we use a noncommittal fast greedy algorithm that is geared to minimize latency with minimum increase of the unit LP, thereby approximately minimizing the product of estimated unit LP and estimated latency.

For evaluating a scheduling option of an output operation, we perform noncommittal greedy scheduling of all remaining unscheduled operations in topological order as follows. We determine the cc to schedule an unscheduled operation *u* by determining in its current MR if there exists at least one cc, where scheduling *u* does not require allocating a new FU (beyond the max number of currently allocated FUs for *u*'s FT), and if so, we schedule *u* in the earliest cc among these cc, thereby greedily minimizing both unit LP and latency for this local scheduling. If such a cc does not exist, we schedule *u* in the earliest cc in its current MR. This simple scheduling strategy is reasonably effective since it keeps FU allocation and thus unit LP growth as slowly increasing as possible while always trying to schedule as early as possible, which is correlated to the goal of minimizing the product of unit LP and latency in LPR-GPS.

5.4. Experimental Results

Our LPR-GPS algorithm is implemented in C++ and experiments were performed on a machine with Core i7-4710HQ (3.5GHz) and 16GB RAM in Windows 10. We use 11 DFG benchmarks from [37]. The sizes of the benchmarks range from 11 to 197 operations. We have identified many works such as [42] [43] [44] [45] [46] [49] on FU design to obtain power/delay characteristics for several FTs as presented in TABLE IX. All FUs are based on 180 nm CMOS technology and for 16-bit input data. The latency constraints for the DFGs are determined as a latency constraint factor times the critical path delay. In our experiments, the latency constraint factor is in the range of [1.5, 2], as we need to represent typical embedded system requirements here in which there is a target speed but it is not the fastest possible, and in which total leakage energy minimization is a significant goal. The detailed results for a latency constraint factor of 1.8 is presented in TABLE X, while results for other latency constraint factors are captured by a plot in Figure 17.

As we noted earlier, it is not meaningful to compare LPR-GPS to other power optimization techniques in HLS, since they use orthogonal approaches such as V_{dd} or V_{th} based module selection and interconnect switching minimization, and since they can be combined with LPR-GPS to get better results than they would get using those approaches alone. In other words, LPR-GPS's exploration is not competing with those of other known power minimization approaches, but in fact can supplement them. In TABLE X, we do the more meaningful comparisons of LPR-GPS to two versions of our internal techniques that have competing approaches to some of LPR-GPS's sub-techniques and do not consider the dimension of obtaining an "optimal" latency to minimize total LE: PFDS and PR-GPS (both discussed in Section 5.3.2). For PFDS results, the latency achieved always turns out to be equal to the latency

constraint, although the moderate latency constraint provides an appreciable optimization space. It is almost the same for PR-GPS; however, compared to PFDS, PR-GPS reduces the total number of FUs by an average of 40.19%, yielding an average total LE reduction of 37.05%. This demonstrates the efficacy of both random-scheduling based metric-conducive determination of scheduling probabilities, and the RMS-based leakage power estimation (or any min-max objective estimation) that are present in PR-GPS but not PFDS. After incorporating greedy-scheduling based latency estimation and scheduling based on minimizing estimated total LE, the total LE results of LPR-GPS are significantly better than both PFDS and PR-GPS: an average 44.86% (12.41%) of total LE reduction is achieved comparing to PFDS (PR-GPS). Although the FU usage, and thus unit LP, of LPR-GPS is slightly increased compared to PR-GPS, as expected (and as also demonstrated by the conceptual plot in Figure 13), the latency is decreased by 21.83%. This clearly establishes the benefit of considering estimates of both unit LP and latency during scheduling.

TABLE IX
CHARACTERISTICS OF FUS

Function Type	**Characteristics**	
	LP (mW)	**Delay (cc)**
Adder/Subtractor **[16]**	0.11	10
AND **[17]**	0.02	1
Multiplier **[18]**	1.76	22
ASR/LSR **[19]**	0.20	1
SRAM Read **[20]**	0.11	1
SRAM Write **[20]**	0.22	2
Divider **[21]**	5.28	88

It can be seen that PR-GPS fails to minimize total LE compared to PFDS for DFGs h2v2 and collapse due to extra FU usage in the latter. Besides, both algorithms have the same total LE for DFG “horner”. However, with latency considered, LPR-GPS can obtain superior scheduling solutions for these 3 DFGs compared to PR-GPS: the total LE of each of the three DFGs are decreased by more than 30%. This occurs due to the total number of FUs in the 3 DFGs not increasing or increasing very little in LPR-GPS compared to the other two algorithms, while the latency realized by LPR-GPS is decreased by about 38%. This observation further demonstrates the importance of considering latency as a part of total LE optimization in HLS.

It can also be noted that for four of the DFGs, LPR-GPS obtains less than 4% total LE improvement over PR-GPS. To identify the reason, we used a similar approach to that in Figure 13, i.e., ran PR-GPS with various latency constraints and plotted the total LE results across different latency constraint factors from 1 to 2. It turns out that the total LE differences between the given L_c’s and a smaller latency with minimum total LE are all lower than 10% and an average of 5% for the four DFGs. Thus, the fact that LPR-GPS does not obtain significant total LE improvements compared to PR-GPS for these DFG’s is mainly due to the inherent

characteristics of the DFG's and not due to a deficiency in LPR-GPS. The overall results in TABLE X thus demonstrate that LPR-GPS can search the solution space that is neglected by PR-GPS (and in general, by algorithms minimizing unit LP only), efficiently and effectively to yield latencies $\leqslant$ the given L_c so that close to the lowest possible total LE for the given DFG is obtained.

To further demonstrate the efficacy of LPR-GPS with various moderate latency constraints, we also run PR-GPS and LPR-GPS under six different latency constraint factors in the range [1.5, 2], with a step size of 0.1. The average total LE of the two algorithms under different latency constraints are plotted in Figure 17. The average LPR-GPS results are more stable than the average PR-GPS results, showing that LPR-GPS is capable of locating the sweet spot latency $L \leqslant L_c$, where the unit LP and latency product is minimized.

Finally, we present the runtime comparison among the initial energy-driven scheduling probability determination step (for initial DG construction), PFDS, PR-GPS and LPR-GPS in Figure 18. The runtime of scheduling probability determination step is about half of that of PR-GPS and LPR-GPS. PFDS is very fast, but has a much lower solution quality than our algorithms. Also, while much slower than PFDS, our algorithms are not slow in an absolute sense: they take only about 900 seconds to obtain results for a DFG with 197 operations, which we believe is fast enough from a practical point of view.

TABLE X

TOTAL LE COMPARISON AMONG PFDS, PR-GPS AND LPR-GPS

DFG	# of operations	Latency Constraint	PFDS			PR-GPS				LPR-GPS				
			Latency	# of Fus	LE_{total} (pJ)	Latency	# of FUs	LE_{total} (pJ)	PRDS to PFDS	Latency	# of FUs	LE_{total} (pJ)	LPRDS to PFDS	LPRDS to PRDS
hal	11	86	86	6	497.286	86	5	344.28	-30.77%	70	5	280.22	-43.65%	-18.61%
horner	18	178	178	6	732.363	178	6	732.36	0.00%	122	6	501.96	-31.46%	-31.46%
arf	28	208	208	6	1526.55	208	5	1156.48	-24.24%	138	6	1012.81	-33.65%	-12.42%
h2v2	51	262	262	6	665.27	255	7	675.85	1.59%	148	8	408.72	-38.56%	-39.53%
collapse	56	167	167	10	1668.23	167	12	1705.37	2.23%	100	11	1165.74	-30.12%	-31.64%
write	106	178	178	23	1765.91	178	10	1444.45	-18.20%	99	17	891.44	-49.52%	-38.29%
interpolate	108	140	140	24	2317.06	140	17	2161.38	-6.72%	134	18	2083.65	-10.07%	-3.60%
matmul	109	174	174	24	4024.55	174	14	2321.86	-42.31%	169	13	2236.34	-44.43%	-3.68%
idctcol	114	257	257	15	2272.03	257	11	1652.77	-27.26%	254	10	1605.23	-29.35%	-2.88%
jpeg	134	235	235	45	6219.51	235	16	2452.04	-60.58%	185	16	2218.33	-64.33%	-9.53%
smooth	197	243	243	34	7457.96	243	16	3701.96	-50.36%	239	17	3667.6	-50.82%	-0.93%
Average	84.73	193.45	193.45	18.09	2649.70	192.82	10.82	1668.07	-37.05%	150.73	11.55	1461.09	-44.86%	-12.41%

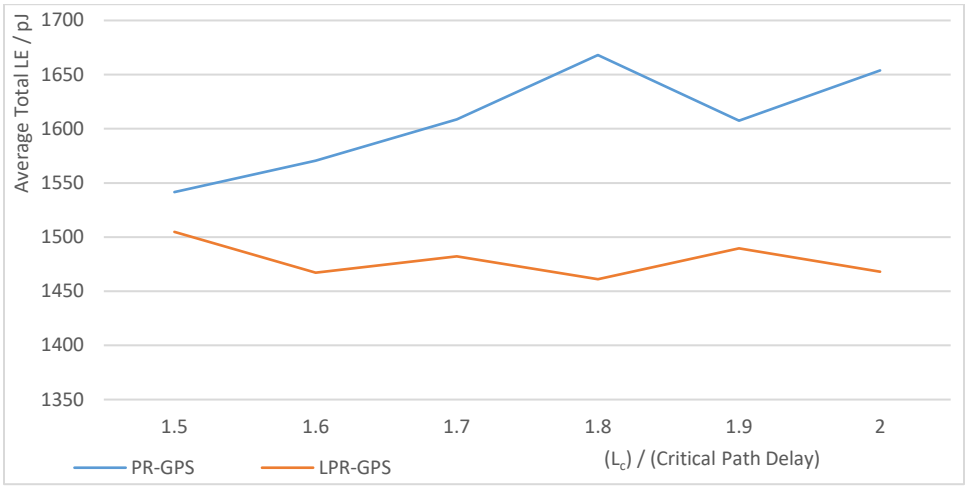


Figure 17. Average total LE comparison between PR-GPS and LPR-GPS subject to different latency constraints.

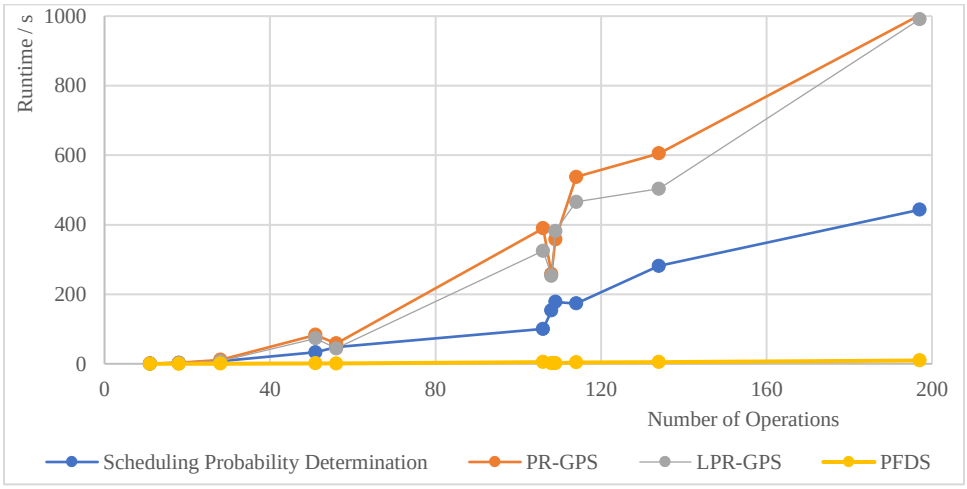


Figure 18. Runtime comparisons among initial scheduling probability determination, PFDS, PR-GPS and LPR-GPS.

6. FALLS: FAST AND EFFICIENT FU MINIMIZATION

The work in this chapter has been presented in [5] and accepted for publication in [6].

In this chapter, we propose a latency-constrained scheduling algorithm FrActional search and Lookahead based List Scheduling (FALLS) with a single-speed library to minimize the total number of FUs (the classical MR-LCS problem), and thus the total area, in HLS designs. Based on LS and inherits its efficiency, we have made the following two major innovations:

- A novel lookahead technique to selectively schedule available operations by allocating the needed FUs earlier or reserving available FUs for scheduling more timing-urgent operations later, such that no additional FU is needed and higher FU utilization is obtained;
- A fractional search framework is developed to iteratively estimate the number of FUs of each function type required in the final design based on the current scheduling solution and FU utilization, and reiterate the lookahead-based list scheduling with the new FU allocation estimate to further increase FU utilization.

Extensive experiments conducted over several DFGs and a wide range of latency constraints demonstrate that FALLS is much more effective than other approximate state-of-the-art algorithms in both number of FUs and total FU area, and has a much smaller runtime. Results also show that FALLS has only an average 5.5% optimality gap compared to an optimal integer linear programming (ILP) formulation, but is 278k times faster. FALLS also performs much better in architectural (FU + mux/demux + register) area, interconnect congestion and number of interconnects than the competing approximate algorithms, and is at most 6% worse in them than the ILP method.

6.1. Formulation of the Optimization Objective

The objective, minimizing the total number of FUs of a scheduling solution, is:

$$\sum_{k \in K} w(k) \times num(k) \tag{6.1}$$

where $w(k)$ is the weight of FT k and is equal to 1 for FU minimization, $num(k)$ is the number of FUs of FT k and K is the set of all FTs. For area minimization, $w(k)$ is the area of FUs of FT k. As we can see from the experimental results, minimizing the total number of FUs has a close indirect relationship with area minimization.

6.2. Our Scheduling Algorithm for FU Minimization

We present our FALLS algorithm in this section. FALLS schedules in chronological order of cc's and utilizes slack to determine the timing-urgency of available unscheduled operations, which are the beneficial aspects of the classical LS algorithm. However, to rectify the drawback of LS, we have made significant extensions as follows:

- We schedule non-0-slack operations following a novel lookahead technique that allocates new FUs earlier than they would be in LS or reserves available FUs in the current cc for scheduling future 0-slack operations, such that the average FU utilization is increased.
- An estimate based extension of binary search, which we call fractional search, is proposed to incrementally estimate the number of FUs required for the design and finally accurately pre-allocate FUs at the last scheduling iteration to further increase FU utilization.
- We use FU utilization rate as a general guideline to dynamically adjust pre-allocation, pre-allocation expansion technique for conservatively pre-allocating more FUs to increase FU utilization and pre-allocation pruning technique for eliminating redundant FUs.

A general view of FALLS is given first: the pseudo code is presented in Figure 21. The internal scheduler of FALLS, Lookahead, is based on LS but improved by our lookahead

technique. Nesting the enhanced scheduler (in line 4 and 9), fractional search iteratively determines a more accurate pre-allocation by the pre-allocation expansion (line 5) and pruning (line 6 to line 16) technique, which are based on the pre-allocation and post-allocation of the previous iteration (call to Algorithm Lookahead). The final solution is the latest solution after the last iteration where there is no improvement to the current solution after FU expansion and pruning techniques.

Algorithm FALLS (DFG $G(V, E)$, L_c, FU library R)
1. $soln.\boldsymbol{r}^{pre}$ = (1,1, ..., 1) *//pre-allocate one FU per FT*
2. Compute the ALAP times t^L for L_c
3. **Repeat** *//fractional search begins*
4. $soln$ = Lookahead (t^L, $soln.\boldsymbol{r}^{pre}$)
5. For each FT k where $soln.r_k^{post} > soln.r_k^{pre}$, increase $soln.r_k^{pre}$ by Equation (6.7) ($r_k^{pre/post}$ is the k'th element of vector $\boldsymbol{r}^{pre/post}$)
6. **For** each FT k where $soln.r_k^{post} \leq soln.r_k^{pre}$ **Do**
7. Get $r\_major_k^{pre}$ by major pruning (see section 6.2.4) of $soln.r_k^{pre}$
8. Temporarily update $soln.\boldsymbol{r}^{pre}$ with $r\_major_k^{pre}$
9. Get a new solution = Lookahead (t^L, $soln.\boldsymbol{r}^{pre}$)
10. **If** the cost of the new solution is improved **Do**
11. Linear search the range [$r_k^{pre-min}$, $r\_major_k^{pre}$] to determine a better $r\_minor_k^{pre}$, where $r_k^{pre-min}$ is the previous largest unsuccessful $soln.r_k^{pre}$ that was tried
12. **Else**
13. Binary search the range ($r\_major_k^{pre}$, $soln.r_k^{pre}$] to determine a better $r\_minor_k^{pre}$
14. **End If**
15. Update $soln.\boldsymbol{r}^{pre}$ with the best $r\_minor_k^{pre}$ or $r\_major_k^{pre}$
16. **End For**
17. **Until** no improvement in $soln.\boldsymbol{r}^{post}$
18. **Return** the latest scheduling solution

Algorithm Lookahead (ALAP times t^L, pre-allocation vector $\boldsymbol{r}^{pre}$)
1. $\boldsymbol{r}^{post} = \boldsymbol{r}^{pre}$, $t = 1$
2. Unschedule all operations if they are scheduled
3. **While** there are unscheduled operations **Do**
4. **For** each FT k **Do**
5. Determine the available unscheduled operation set $U_{t,k}$
6. Compute slack s_u for all $u \in U_{t,k}$ by Equation (2.6)
7. Schedule 0-slack operations in $U_{t,k}$ to t, allocate new FUs if needed, update r_k^{post} if an FU is used for the first time
8. Apply the lookahead technique (see section 6.2.1) to schedule non-0-slack operations in $U_{t,k}$ to t, allocate new FUs if needed and update r_k^{post} if an FU is used for the first time
9. **End For**
10. $t = t + 1$
11. **End while**
12. **Return** $\boldsymbol{r}^{pre}$, $\boldsymbol{r}^{post}$ and the scheduling solution as $soln$

Figure 19. The pseudo code of our FALLS algorithm.

6.2.1. Lookahead Scheduling

The lookahead technique makes better scheduling decisions than LS for non-0-slack operations. In the current scheduling cc *t*, it detects operations that are currently unavailable and will become 0-slack in some near-future cc's. To allow these operations to be executed on currently available FUs when they are available and 0-slack, some available FUs are reserved for this purpose in the current cc *t*. Moreover, it aggressively allocates new FUs in cc *t* to schedule certain non-0-slack operations under the condition that if the operations are not scheduled in cc *t*, the same number of new FUs are still needed to be allocated for scheduling them in later cc's. By preventing allocating avoidable new FUs in later cc's and allocating new FUs earlier that are unavoidable later, the average FU utilization is increased and hence the number of FUs needed is minimized.

The advantage of reserving FUs for later use is illustrated by the example in Figure 20. The DFG in Figure 20(a) has only two FTs: addition of 1-cc delay and multiplication of 2-cc delay. The latency constraint is 5 cc's. In Figure 20(b), LS schedules op5, whose slack is 3, in cc 1, since it is the only available multiplication operation in cc 1 and there is an available multiplier. The overlapping of execution time of op2 and op5 results in a new multiplier being allocated in cc 2. On the other hand, in cc 1, our lookahead scheduling detects that op2 will become 0-slack in cc 2 and hence reserves the multiplier for scheduling op2 in cc 2 to avoid the new multiplier being allocated, as in Figure 20(c). As the scheduling proceeds, op5 eventually becomes 0-slack in cc 4, and the multiplier being busy in cc's 2-3 becomes available for op5. Therefore, by reserving the multiplier in cc 1 and scheduling op5 later, one multiplier is saved.

The other aspect of the lookahead scheduling, aggressive early new FU allocation, is illustrated by the example in Figure 21 with the same set of FTs and latency constraint as in

Figure 20. The scheduling quality here solely depends on the allocation of adders. In Figure 21(b), after op4 is scheduled in cc 1, LS schedules op5 in cc 3, since op5 is non-0-slack in cc 2 and there is no available multiplier then. Although a new multiplier must be allocated no matter where op5 is scheduled, LS fails to detect this situation due to the limited information provided by slack alone. This forces op6 to be scheduled in cc 5, where op3 is concurrently scheduled. This leads to a new adder to be allocated. Different from LS, our lookahead scheduling in Figure 21(c) realizes that a new multiplier is unavoidable for scheduling op5, hence allocates it when op5 is first available in cc 2 and schedules op5 there. Such scheduling decision makes no change in multiplier allocation, but reduces the number of adders by one: op6 can be scheduled one cc earlier and hence avoid being executed concurrently with op3.

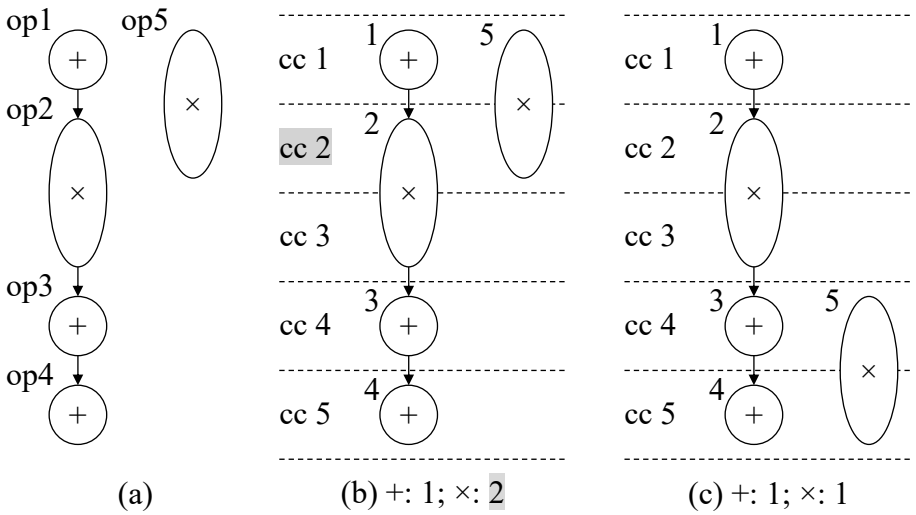


Figure 20. Illustration of the advantage of reserving FUs for later use in the lookahead scheduling of FALLS. "op*i*" denotes operation *i*. FU allocation results shown below solutions. (a) An unscheduled DFG; (b) The solution of LS; (c) The solution of lookahead scheduling.

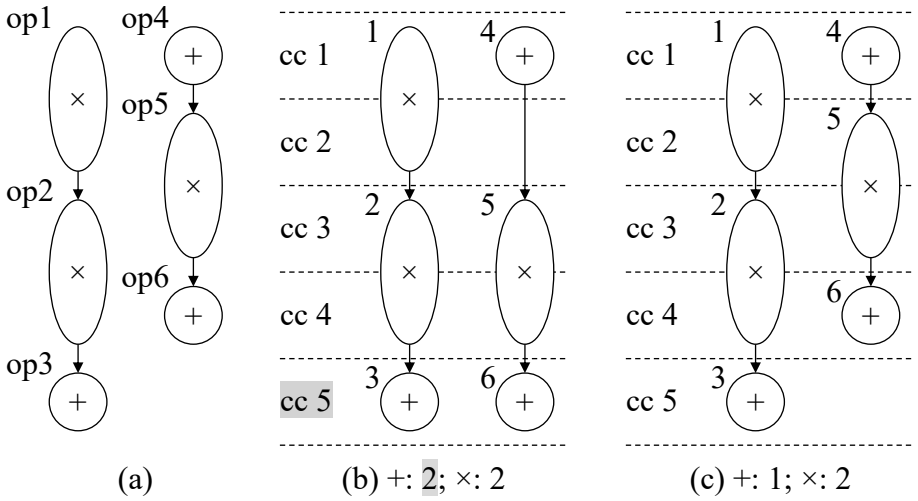


Figure 21. Illustration of the benefit of early allocation of new FUs in the lookahead scheduling of FALLS. FU allocation results shown below solutions. (a) An unscheduled DFG; (b) The solution of LS; (c) The solution of lookahead scheduling.

Now we formulate our lookahead technique as follows. In any cc t during the scheduling process, after scheduling 0-slack operations of FT k, we explore the cc's in the cc range $R(t) = [t + 1, t + d_k - 1]$ of FT k, where d_k is the delay of FT k and $d_k > 1$. We explore this cc range because this is the maximum range for which any executing operation of FT k scheduled in a cc $\leqslant t$ will finish its execution in some cc in this range, and we will thus know exactly how many FT k FUs will be available in these cc's. This information, along with which operations will become 0-slack in these cc's, is needed to determine FU reservation and early new FU allocation in cc t. For cc $i \in [t, t + d_k - 1]$, we define: $a(i)$ to be the number of FUs that will be busy in $i - 1$ and become available in i; $z(i)$ to be the total number of 0-slack operations in i; $z'(i)$ to be the number of 0-slack operations in i that are available in cc $t < i$. Therefore, $z(i) - z'(i)$ is the number of 0-slack operations in i that are unavailable in t. Each of these parameters in the cc range can be determined in cc t. With these parameters, we can recursively compute $Avail(i)$, the number of available FUs in cc i after scheduling $z(i) - z'(i)$ operations in cc i, as:

$$Avail(i) = max\{0, Avail(i-1) + a(i) - [z(i) - z'(i)]\} \quad (6.2)$$

At the recursion boundary of cc *t*, *Avail*(*t*) is the number of available FUs after scheduling all 0-slack operations in t. Based on *Avail*(*i*), we can determine *new*(*i*), the number of new FUs needed in *i* for scheduling the *z*'(*i*) 0-slack operations of *i*, as:

$$new(i) = max\{0, z'(i) - Avail(i)\} \quad (6.3)$$

This needs to be followed by an update of *Avail*(*i*) in order to compute *Avail*(*i* + 1) by Equation (6.2): *Avail*(*i*) = 0 if *new*(*i*) > 0, otherwise *Avail*(*i*) = *Avail*(*i*) – *z*'(*i*). As is hopefully clear from the formulation, new FUs are only allocated for scheduling *z*'(*i*) 0-slack operations when there are not enough available FUs after scheduling the *z*(*i*) – *z*'(*i*) 0-slack operations. The updated *Avail*(*i*) that accounts for scheduling *z*'(*i*) operations becomes the number of available FUs in *i* after scheduling all its *z*(*i*) 0-slack operations.

After all cc's in the cc range *R*(*t*) are explored, we can determine *S*(*t*), the maximum number of available non-0-slack operations to be scheduled in cc *t* by:

$$S(t) = max\left\{0, \min_{j \in [t, t+d_k-1]} Surplus(j)\right\} + \sum_{j=t+1}^{t+d_k-1} new(j) \quad (6.4)$$

where

$$Surplus(i) = Avail(t) + \sum_{j=t+1}^{t+d_k-1} \{a(j) - [z(i) - z'(i)]\} \quad (6.5)$$

Surplus(*i*) is thus the number of available FUs in cc *i* after scheduling *z*(*j*) – *z*'(*j*) 0-slack operations in each cc *j* in [*t* + 1, *i*] without allocating any new FUs in any of these cc's; it can thus be negative. Equation (6.4) incorporates both aspects of the lookahead technique that are illustrated in Figure 20 and Figure 21. The first term with *Surplus*(*i*) allows use of only $max\left\{0, \min_{j \in [t, t+d_k-1]} Surplus(j)\right\}$ of the available FUs in cc *t* for available non-0 slack operations in it and reserves the rest for later use in *R*(*t*). The second term with *new*(*i*) is for early

allocation and use of the appropriate number of new FUs in cc *t*. The idea of *Surplus*(*i*) is that if it is positive, and for the sake of argument we ignore other *Surplus*(*j*) values, then we can schedule at most *min*(*Surplus*(*i*), *z*'(*i*)) available non-0-slack operations in cc *t* of the *z*'(*i*) 0-slack operations of cc *i*, on the already available FUs in *t* (after its 0-slack operations are scheduled), without incurring any extra new FU in *R*(*t*) compared to scheduling these operations in cc *i*. However, for this to be true for all cc's in *R*(*t*), we can only schedule $\min_{j \in [t, t+d_k-1]} Surplus(j)$ (if it is positive) available non-0-slack operations in cc t (in slack increasing order—these are the operations that become 0-slack earliest among all the *z*'(*i*) operations in *R*(*t*)) without allocating any extra new FUs in *R*(*t*). If any more are scheduled in cc *t*, then the minimum positive *Surplus* point *r* in *R*(*t*) will become negative, meaning that extra new FU(s) will be needed to schedule some of the *z*(*r*) – *z*'(*r*) 0-slack operations in cc *r*.

Thus, accounting for both the minimum Surplus(*i*) and early allocation of new FUs in cc *t*, we schedule *S*(*t*) available non-0 slack operations in slack increasing order in cc *t*.

Now we illustrate the lookahead formulation by the example in Figure 20 for scheduling multiplication op5 in cc *t* = 1. The FT assumed below is multiplication. Since the delay of multiplication is 2 cc's, from cc 1 we lookahead up to cc 2. Since we have an available multiplier in cc 1 (*Avail*(1) = 1) and there is one 0-slack operation in cc 2 (*z*(2) = 1), but none of the two multiplications in the DFG is a *z'*(2) operation (*z'*(2) = 0) as op2 is not available in cc 1 and op5, which is available in cc 1, becomes 0-slack in cc 4. *Avail*(2) = 0 and *Surplus*(2) = 0 according to Equation (6.2) and (6.5), respectively, meaning that no FU allocated till cc 1 is available after scheduling *z*(2) – *z'*(2) = 1 operations in cc 2. Since *z'*(2) = 0, which means no *z'*(2) operations needs to be scheduled in cc 1, *new*(2) = 0. Therefore, from Equation (6.4), *S*(t) = 0 and hence no operation is scheduled in cc 1, though there is an available multiplier and an available operation

(op5). We are thus reserving this multiplier for use in cc 2 by another operation (op2) that only becomes available and 0-slack in cc 2. This gives us the best scheduling solution in Figure 20(c). Similar calculation can be done for the other example in Figure 21 that illustrates the early new FU allocation aspect of our lookahead technique.

We present the following formal result of the lookahead technique, where we assume for simplicity of exposition that $\min_{j \in [t,t+d_k-1]} Surplus(j)$ is positive.

Theorem 1: (*a*) *By reserving* $Avail(t) - \min_{j \in [t,t+d_k-1]} Surplus(j)$ *available FUs of cc t for later use in the cc range* [$t + 1$, $t + d_k - 1$], *given the configuration of cc t, the lookahead technique minimizes the number of new FUs allocated in this cc range. (b) Additionally, by early allocation of* $\sum_{j=t+1}^{t+d_k-1} new(j)$ *new FUs in cc t, the lookahead technique does not allocate any extra new FUs for scheduling 0-slack operations in the cc range, compared to not doing this early allocation.*

Proof Outline: (a) Let *Surplus(r)* = $\min_{j \in [t,t+d_k-1]} Surplus(j) = x$. Thus, x available non-0-slack operations of cc t that become 0-slack in the *cc range* [$t + 1$, $t + d_k - 1$] are scheduled in increasing slack order in cc t. These are the first x *z'(i)* operations in this cc range, and assume they are the *z'(i)* operations in the range [$t + 1$, $t + j$], $j \leq d_k - 1$. This means that the "new" surplus of every cc in the cc range reduces by x, but none becomes negative. For the first j cc's in this range, no new FUs are needed to schedule any of their 0-slack operations, as, by definition of *Surplus*(i), all their $z(i) - z'(i)$ 0-slack operations are to be scheduled on available FUs in each of these cc's, and the remaining $z'(i)$ 0-slack operations are scheduled on available FUs in cc t. For the remaining cc's in the above range, after the reduction by x of their surpluses, the number of FUs available to schedule their $z'(i)$ operations is exactly the same as under the scheduling scenario where each of the $z'(i)$ operations of cc's in [$t + 1$, $t + j$], are scheduled in exactly those

cc's (where they need to be scheduled at the latest), instead of earlier in cc t. Thus, any new FUs that need to be allocated in the range $[j + 1, t + d_k - 1]$ to schedule their $z'(i)$ 0-slack operations are also unchanged, and is a minimum number, since they are based on only scheduling 0-slack operations of earlier cc's (that are mandatory to schedule). On the other hand, had we scheduled $x + m$, $m > 0$, non-0-slack operations of cc t, then the new surpluses of some cc's q in the range $[t + 1, t + d_k - 1]$ (there will be at least one, cc r) will become a negative value $-y_q$, meaning that in each of them $y_q > 0$ more new FUs will be needed to schedule their 0-slack operations (the $z'(i)$ ones and/or the $z(i) - z'(i)$ ones) than needed in either of the above two scheduling scenarios.

(b) It follows from the definition of $new(i)$ that these are the minimum number of new FUs needed in cc i in the range $[t + 1, t + d_k - 1]$ to schedule their $z'(i)$ 0-slack operations irrespective of how the available FUs in cc t are used to schedule their non-0-slack available operations (for our lookahead based scheduling, this will also be the exact number of new FUs needed). Thus, by allocating a portion of them earlier in cc t for the purpose of scheduling exactly the same $z'(i)$ operations in cc t that would be scheduled on them if they had not been allocated in cc t, but in each later cc i in the above range, we do not need any extra new FUs compared to the latter scheduling scenario. Further, by allocating these new FUs earlier in cc t and scheduling the respective operations on them at that time, we will make them available earlier (after cc $t + d_k - 1$), thus increasing the likelihood that fewer new FUs will be needed after cc $t + d_k - 1$ due to earlier allocated FUs being available. ■

Finally, for operations of single-cc delay, a different lookahead approach to aggressively schedule non-0-slack operations in cc t is as follows. For any cc t, the number of available FUs of FT k in cc $t + 1$ is number of all FUs of k that are either available or busy in cc t. If in cc $t + 1$, the number of 0-slack operations that are not available in cc t is greater than the number of

available FUs in cc $t + 1$, then let m be the difference. We allocate m new FUs in cc t on them and schedule available operations in minimal slack order in cc t. In this way, we allocate new FUs that are mandatory one cc earlier to maximize their utilization potential.

6.2.2. Fractional Search

Our fractional search framework contains two sub-techniques: pre-allocation expansion and pre-allocation pruning. Both techniques rely on an indicator of FU utilization, utilization rate, to determine the number of FUs to be adjusted in the pre-allocation. The utilization rate (ur) of an FU is the fraction of cc's in which the FU is busy executing operations over the entire scheduling latency. For the p'th FU of FT k with $n_{k,p}$ operations bound to it, its utilization rate $ur_{k,p}$ is:

$$ur_{k,p} = \frac{n_{k,p} \times d_k}{L} \tag{6.6}$$

where d_k is the delay of FT k and L is the achieved latency of the current scheduling solution. Intuitively, FUs allocated in earlier cc's have a greater potential to have high utilization rates compared to those allocated in later cc's.

We first conceptually illustrate fractional search in Figure 22. For any FT k of a solution, we determine the new pre-allocation by the pre-allocation and post-allocation of the previous iteration. If the former is smaller than the later, we expand the pre-allocation by adding the sum of utilization rates of new FUs (an optimistic estimate) to it. Otherwise, we attempt to prune pre-allocated FUs by a utilization rate based major pruning followed by minor binary or linear pruning steps to gradually approach the accurate pre-allocation. Unlike binary search, which iteratively eliminates half of the search space, fractional search makes the new estimate based on utilization rate to more efficiently locate the target value. Fractional search terminates when the

latest round of prunings for each FT that satisfies the pruning condition, no further solution improvement can be obtained.

An example of applying fractional search to our lookahead scheduling to improve the scheduling solution is given in Figure 23. We consider another DFG in Figure 23(a) and an FU library which is the same as the one in Figure 20. Here, the latency constraint is 3 cc's and the scheduling quality solely depends on the scheduling of addition operations. In cc 1 of Figure 23(b), since adder has a single-cc delay, our lookahead technique without considering single-cycle operations (but we do handle these operations by a different lookahead approach at the end of Section 6.2.1) cannot search beyond cc 1, the behavior of our lookahead scheduling is same to LS. As a result, op2, op4 and op6 are pushed into cc 3 and become 0-slack operations, which makes two new adders to be allocated in cc 3. Therefore, a total of 3 adders are needed in the final solution. The two adders newly allocated in cc 3 are significantly under-utilized as they are idle in the first two cc's. On the other hand, the solution of our lookahead scheduling with fractional search in Figure 23(c) that allocates only two adders, is derived as follows. The solution of the first scheduling iteration is the same to the solution in Figure 23(b). Since there are two new FUs allocated in the last cc, both have a utilization rate of 1/3, and since $\lceil 1/3 \times 2 \rceil = 1$, the pre-allocation expansion technique estimates that only one additional adder, i.e., a total of 2 adders is needed in the pre-allocation, compared to only 1 adder in pre-allocation in the current iteration. Note that the pre-allocation pruning technique is not able to prune any FU in this example as there are new adders allocated and there is only one multiplier. In the second scheduling iteration, with two adders in the pre-allocation, both non-0-slack op1 and op5 can to be scheduled in cc 1, which makes operation 2 to be available in cc 2 rather than the last cc as in the first iteration. This avoids op2, op4 and op6 all being scheduled in cc 3 as occurs in Figure

23(b) and increases the utilization rate of the newly pre-allocated FU from 1/3 to 2/3 and the overall FU utilization rate from 5/9 to 5/6. We thus obtain the solution of Figure 23(c) with reduced cost that requires only 2 adders. This example demonstrates the benefits of fractional search to minimize the number of FUs.

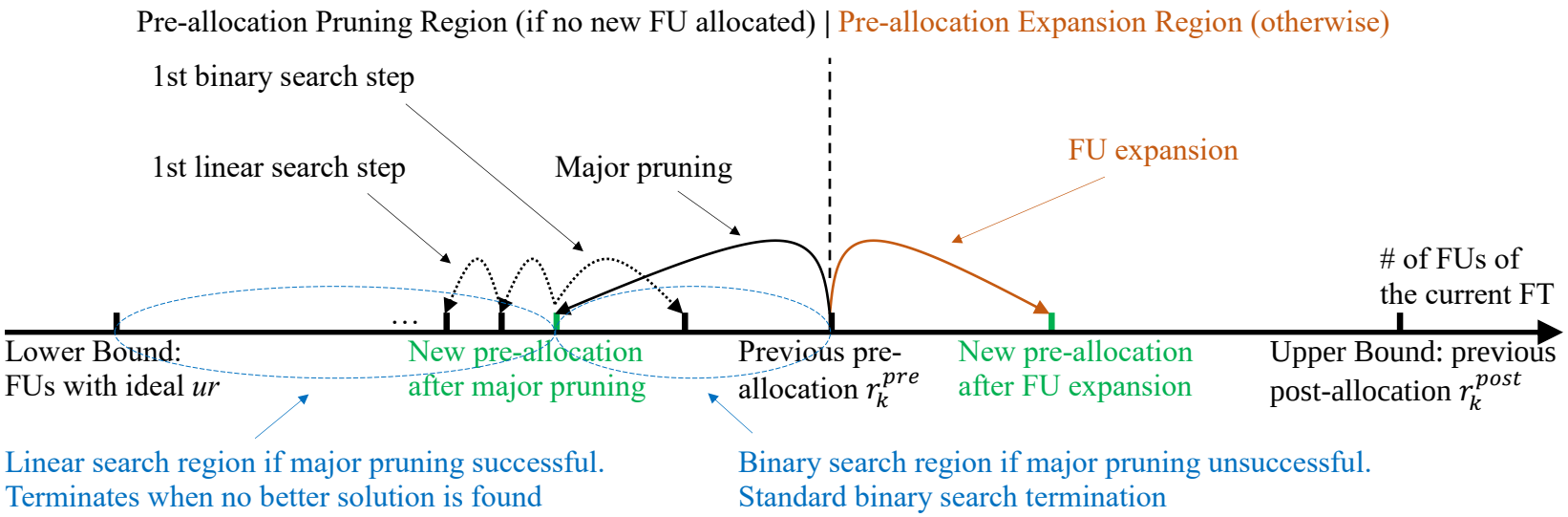


Figure 22. Graphical illustration of a single iteration of fractional search for a FT.

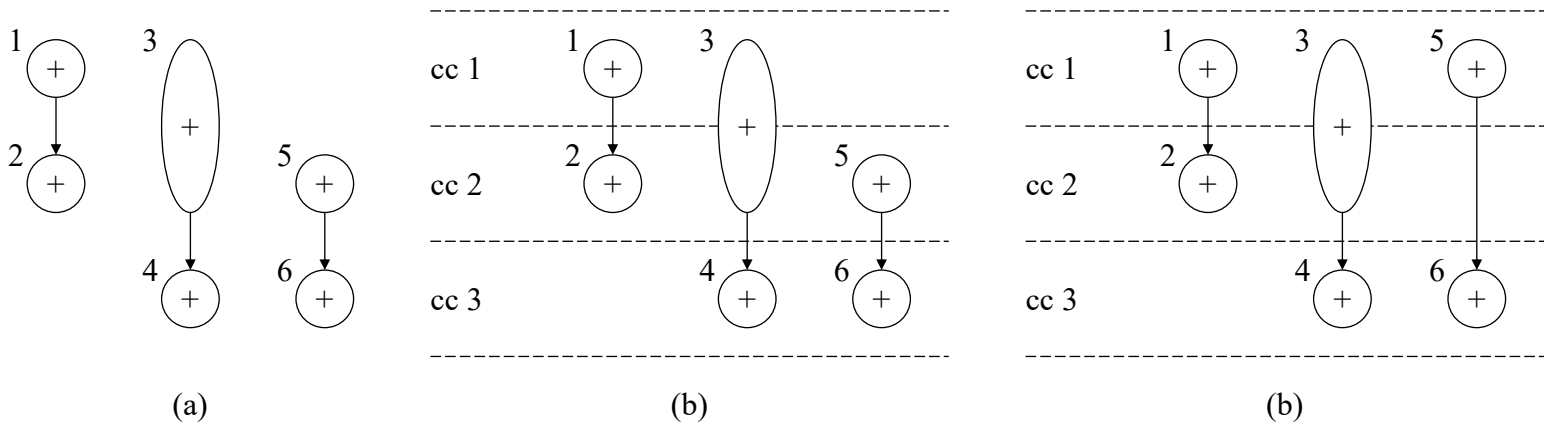


Figure 23. Illustration of the benefit of FU pre-allocation estimate in fractional search of FALLS. (a) An unscheduled DFG; (b) The solution of the lookahead scheduling; (c) The solution of the lookahead scheduling with fractional search.

6.2.3. Pre-allocation Expansion Technique

Given a solution of an iteration, for any FT *k*, if its pre-allocation r_k^{pre} is smaller than its post-allocation r_k^{post}, the pre-allocation expansion technique is applied. The number of FUs to be increased in the pre-allocation r_k^{new} of FT *k* is:

$$r_k^{new} = \left\lceil \sum_{p \in FU_{new(k)}} ur_{k,p} \right\rceil \tag{6.7}$$

where *FU_new*(*k*) is the set of new FUs of FT *k* allocated in the current scheduling iteration. The pre-allocation expansion is performed in a conservative way in which it only allocates the minimum number of FUs which can handle all operations bound to the new FUs with an ideal utilization rate of 100%. This allows fractional search to gradually approach the minimum number of required FUs.

6.2.4. Pre-allocation Pruning Technique

Given a solution of an iteration, for any FT *k*, if its pre-allocation r_k^{pre} is not smaller than its post-allocation r_k^{post}, the pre-allocation pruning technique is applied. The pruning is performed when $r_k^{pre} > r_k^{post}$, since the unutilized $r_k^{pre} - r_k^{post}$ pre-allocated FUs that have no operations bound to should obviously be pruned. Further, when $r_k^{pre} = r_k^{post}$, it is potentially beneficial to prune some of the utilized pre-allocated FUs, since these over-allocated FUs may be used to over-schedule less timing-urgent non-0-slack operations that have slacks greater or equal to the delay, making the FUs sparsely utilized in later cc's. This pruning includes a major pruning followed by minor prunings that are either based on binary or linear search.

Besides pruning the unused FUs, if any, the idea of major pruning is to adaptively increase the utilization rates of the most under-utilized FUs. Let the maximum and minimum utilization rate among all the used FUs in the current solution be ur_{max} and ur_{min}, respectively.

We can evenly partition the range [ur_{min}, ur_{max}] into α ($\alpha \geqslant 2$; $\alpha = 4$ in our experiments) partitions. The most under-utilized FUs are in the 1st partition, which is the range $urr_1 = [ur_{min}, ur_{min} + \frac{ur_{max} - ur_{min}}{\alpha})$. We attempt to increase the utilization rates of these FUs to the adjacent partition with a higher average utilization rate range uur_2 so that fewer FUs are in the pre-allocation and this is expected to translate to fewer FUs in the post-allocation by reducing over scheduling. To execute the same number of operations that were bound to the most under-utilized FUs whose utilization rates are in urr_1 with fewer but fully-utilized FUs whose utilization rates are in urr_2, the least number of FUs required m is determined as:

$$m = \frac{\sum_{p \in FU(k, urr_1)} ur_{k,p}}{ur_{avg}(k, urr_2)} \tag{6.8}$$

where $FU(k, urr_1)$ is the set of FUs of FT k whose utilization rates are in uur_1 and $ur_{avg}(k, urr_2)$ is the average utilization rate of FUs whose utilization rates are in uur_2. The number of pruned FU is thus $|FU(k, urr_1)| - m$.

As illustrated in Figure 22, after major pruning, there are a series of minor prunings using either linear or binary search. If major pruning leads to a lower-cost solution, we perform linear search to further prune the number of FUs of FT k by the smallest granularity of one and re-schedule, and iterate until no better solution is found. On the other hand, if the new solution is worse than the previous one, we perform binary search in the range of the current pre-allocation and the previous pre-allocation until the best lower-cost solution is found or no lower-cost solution can be found. We apply linear and binary searches on the two situations respectively since the target value in the former is much closer than that in the later.

6.2.5. Time Complexity

The time complexity of the lookahead technique is $O(n \log n + nd_{max})$, where d_{max} is the maximum delay among all FTs. The $n \log n$ term comes from LS's time complexity, and the nd_{max} term from the fact that an FU executing an operation of FT k with delay d_k cc's, will be accessed d_k times to determine $a(i)$ and related parameters for lookahead processing. Further, if n_k is the number of operations of FT k and there are q FTs, fractional search will determine at most $O(\log n_k)$ new pre-allocations (and thus calls to lookahead scheduling) for FT k, and thus it overall makes $O(q \log n) \sim O(\log n)$ (q being a small constant compared to n) calls to lookahead scheduling. Thus, the total time complexity of FALLS is $O(n \log^2 n + (n \log n)d_{max}) \sim O(max(n \log^2 n, (n \log n)d_{max}) \sim O(n \log^2 n)$ if d_{max} is a small constant which is most likely to be.

6.3. Experimental Results

We first present results of FALLS comparing to several state-of-the-art algorithms in this section. Then we discuss our implementation and experimental results of ACO [25], since it is the most recent work on FU minimization in operation scheduling and our implementation according to [25] yields lower quality results than that reported in [25].

6.3.1. Results of FALLS

We implemented FALLS in C++. Experiments were conducted on a machine with Core i7-4710HQ (3.5GHz) and 16GB RAM in Windows 10. First, we make a direct comparison between FALLS and ant colony optimization (ACO) in [25]. The trivial FU library in [25] has only two FTs of FUs: multiplier of 2-cc delay, denoted by "*", for multiplication and division, and ALU of 1-cc delay, denoted by "+", for the remaining arithmetic and non-arithmetic FTs. For each DFG, the latency constraints are set to be a factor, called L_c factor, of the critical path delay. Due to the stochastic nature of ACO, it is hard to obtain consistent good results for all

DFGs. To make a fair comparison, we compare FALLS to ACO in TABLE XI for two large DFGs which are the only ones for which FU allocation results (nFU) of ACO for specified latency constraints are presented in [25]. The DFG sizes are indicated by the number of operations and data dependencies pair. The results show that across 11 different latency constraints, for the two DFGs, FALLS allocates an average of 8.1% and 9.8% fewer FUs than ACO, respectively. Further, the average FU area of FALLS is 10.7% and 7.4% smaller than that of ACO for the two DFGs, respectively, due to overall fewer FU allocation.

TABLE XI

FU ALLOCATION AND AREA COMPARISON BETWEEN ACO AND FALLS USING THE TRIVIAL LIBRARY

L_c Factor	idctcol (114, 164)								invert (333, 354)							
	ACO			FALLS			FU % Improv.	Area % Improv.	ACO			FALLS			FU % Improv.	Area % Improv.
	nFU	+, *	FU Area	nFU	+, *	FU Area			nFU	+, *	FU Area	nFU	+, *	FU Area		
1.0	11	5, 6	861.76	11	6, 5	852.48	0.0%	1.1%	47	25, 22	3648.32	46	22, 24	3593.6	2.1%	1.5%
1.1	10	4, 6	788.48	10	6, 4	769.92	0.0%	2.4%	42	23, 19	3254.08	42	18, 24	3300.48	0.0%	-1.4%
1.2	9	4, 5	705.92	9	5, 4	696.64	0.0%	1.3%	36	20, 16	2786.56	34	14, 20	2677.12	5.6%	3.9%
1.3	8	3, 5	632.64	8	5, 3	614.08	0.0%	2.9%	34	19, 15	2630.72	30	12, 18	2365.44	11.8%	10.1%
1.4	8	3, 5	632.64	7	4, 3	540.8	12.5%	14.5%	30	17, 13	2319.04	26	11, 15	2044.48	13.3%	11.8%
1.5	7	3, 4	550.08	7	4, 3	540.8	0.0%	1.7%	28	16, 12	2163.2	25	10, 15	1971.2	10.7%	8.9%
1.6	7	3, 4	550.08	6	4, 2	458.24	14.3%	16.7%	26	15, 11	2007.36	22	9, 13	1732.8	15.4%	13.7%
1.7	7	3, 4	550.08	6	4, 2	458.24	14.3%	16.7%	25	14, 11	1934.08	21	9, 12	1650.24	16.0%	14.7%
1.8	7	3, 4	550.08	5	3, 2	384.96	28.6%	30.0%	23	13, 10	1778.24	20	9, 11	1567.68	13.0%	11.8%
1.9	6	3, 3	467.52	5	3, 2	384.96	16.7%	17.7%	23	13, 10	1778.24	19	8, 11	1494.4	17.4%	16.0%
2.0	6	3, 3	467.52	5	3, 2	384.96	16.7%	17.7%	22	13, 9	1695.68	18	7, 11	1421.12	18.2%	16.2%
Avg	7.8	3.4, 4.5	620.67	7.2	4.3, 2.9	554.53	8.1%	10.7%	30.5	17.1, 13.5	2367.65	27.5	11.7, 15.8	2192.99	9.8%	7.4%

In TABLE XII, we compare FALLS to LS, FDS [13] [14], SA [25] and ILP [20] [21] (implemented in CPLEX) with 15 DFG benchmarks from [37], to show their effectiveness in FU minimization and the indirectly minimized total area. For this, we constructed a 16-bit non-trivial FU library that has eight FTs: adder/subtractor of 4-cc delay, multiplier of 10-cc delay, divider of 24-cc delay, arithmetic and logical shift register, memory read and write, and logical AND, all 1-cc delay. The delay and area of the FU designs from [36] are theoretically derived based on the number of gate inputs along the critical path and the total number of transistors (T), respectively, which are close enough practical approximations. Each number of FU (nFU) and FU area result in TABLE XII is the average for a DFG for 6 latency constraints with L_c factors in the range [1, 2] with a granularity of 0.2. The results show that FALLS reduces the total number of FUs by an average of 22.4% to 59.4% compared to LS, FDS and SA with similar area reductions. It also shows that FALLS has only 3.6% optimality gap in the number of FUs and merely 2.4% greater FU area compared to the optimal ILP for the 15 DFGs. Further, though the following are not our optimization objectives, we also determined the *architectural area* as follows. We use the linear-time binding techniques developed in [50] that uses simple augmentations of the well-known left-edge algorithm [48] to optimally bind operations to FUs while also using heuristics to share as many interconnects between FUs as possible for multiple data communication (and thereby heuristically minimize mux and demux sizes, and interconnects needed). We also use the algorithm implemented in [50] for optimal post-binding allocation of registers. The *architectural area* is then determined as the sum of the areas of FUs, mux's/demux's and registers. According to [50], while we do not perform binding explicitly with scheduling and module selection, the optimal binding of operations to FUs post the above functions uses the exact number of FUs as determined during the scheduling and module selection stage. There is thus no disadvantage in

number of FUs of different FTs and thus FU power in not performing binding simultaneously with the above stage. The results presented in TABLE XII and TABLE XIII show that FALLS has 18.8% to 53.0% average architectural area reduction compared to the competing approximate algorithms. Further, the average *maximum congestion*, defined as the maximum in- and out-degree of an FU, of FALLS is 3.5% to 14.7% smaller than these algorithms, and its average number of interconnects is 9.6% to 37.2% fewer. Also, FALLS has very close results to ILP: the average architectural area and number of interconnects of FALLS is only 2.7% and 4.0% greater than that of ILP, while the average max congestion is even slightly fewer.

We also performed the experiments with the trivial library used by [25]: the corresponding results are presented in TABLE XIV and TABLE XV. The results are consistently (comparable to the experiments with our non-trivial library) good compared to LS, FDS and SA. Further, the solution quality of FALLS is even closer to that of ILP: the optimality gap of FALLS in FU minimization is merely 1.2% compared to ILP. The abundant experiments using two distinct libraries demonstrate that FALLS is widely applicable to various FU libraries.

Finally, the average runtimes in milliseconds for the experiments in TABLE XII are presented in TABLE XVI. The results show that FALLS is extremely fast that taking only 0.62ms for the smallest DFG and 69.85ms for the largest using the non-trivial 8-speed library. The runtimes of SA and ILP are very high, preventing them from solving practical large-size problems. In fact, for the largest DFG rand-1300, CPLEX runs out of memory for even the smallest latency constraint (and thus the smallest solution space) imposed. This is indicated by "*" symbols in the tables. On the contrary, FALLS is merely about 3 times slower than the extremely fast but extremely sub-optimal LS as shown in the runtime plots of LS and FALLS in Figure 24. The linear curve fitting function for the average runtime of LS and FALLS is

0.01466869 n + 0.4143267 and 0.05515652 n – 0.03750594, respectively, where n is the number of operations. The polynomial curve fitting function for them is – 0.5733374 + 0.02756854 n – 0.000009815745 n^2 and –2.095263 + 0.08203282 n – 0.00002045069 n^2, respectively. Note that only the linear terms in both fitting functions matter most: the square term in the polynomial fitting function has too small constants considering most DFGs have far fewer than 200 operations. For both curve fitting functions, the ratio of the constants of the linear terms of LS and FALLS is about 1: 4, which is somewhat close to the theoretical complexity ratio of $log\ n$, if n is not a large value. This thus empirically demonstrate the time complexity of FALLS. The sharp runtime slope of FALLS on DFG size 200 to 300 operations is due to the extra FTs in the DFG (not all DFGs have the same # of the FTs: the more FTs a DFG has, the more time needed in FALLS' lookahead and hence longer runtime) and fluctuation on runtime measurement. FALLS is also 68, 873 and 278k times faster than FDS, SA and ILP, respectively, as indicated in TABLE XVI. Considering that FALLS obtains solutions to the largest DFG with 1300 operations in a miniscule 69.85 milliseconds, and that it has an average optimality gap of only 5.5%, one can conclude that it has very good runtime and solution quality scalability.

TABLE XII

AVERAGE NUMBER OF FUS AND AREA (INCLUDING FU AREA AND ARCHITECTURAL AREA) COMPARISONS AMONG COMPETING ALGORITHMS USING THE NON-TRIVIAL LIBRARY

DFG		# of ops	LS					FDS					SA					ILP					FALLS				
			nFU	FU Area	Mux+Demux Area	Register Area	Arch. Area	nFU	FU Area	Mux+Demux Area	Register Area	Arch. Area	nFU	FU Area	Mux+Demux Area	Register Area	Arch. Area	nFU	FU Area	Mux+Demux Area	Register Area	Arch. Area	nFU	FU Area	Mux+Demux Area	Register Area	Arch. Area
1	hal	11	6.2	388.8	8.0	31.7	428.5	5.7	347.5	9.4	16.4	373.4	5.8	361.3	9.8	16.4	387.5	5.7	347.5	9.1	14.7	371.3	5.7	347.5	10.7	14.7	372.9
2	horner	18	6.3	350.5	10.6	130.2	491.3	5.7	308.5	11.0	16.4	335.9	5.3	320.0	10.7	18.8	349.5	5.0	292.5	9.4	17.0	318.9	5.0	292.5	7.8	15.8	316.2
3	arf	28	9.3	588.4	16.6	194.2	799.2	6.0	339.2	15.2	32.9	387.3	6.0	339.2	18.9	37.5	395.6	4.7	268.2	10.2	29.3	307.7	4.7	281.2	7.4	27.0	315.5
4	motion	32	21.0	1258.9	17.6	68.6	1345.1	13.5	785.0	21.1	38.7	844.8	14.0	820.3	22.7	36.4	879.4	12.7	749.2	18.4	32.9	800.5	12.7	736.2	21.1	30.5	787.8
5	ewf	34	4.2	174.8	6.1	271.0	451.9	4.5	202.3	11.4	22.9	236.6	4.3	188.6	14.2	24.1	226.9	3.3	145.1	6.4	15.3	166.7	3.3	145.1	6.2	14.7	166.0
6	h2v2	51	10.7	373.5	20.8	31.7	426.0	7.2	287.9	14.1	28.7	330.7	7.5	224.3	19.4	38.1	281.8	6.3	219.1	14.2	34.6	267.9	6.5	232.9	13.8	37.5	284.2
7	feedback	53	21.2	1114.5	41.8	129.1	1285.3	13.8	733.4	43.8	60.4	837.7	14.5	746.3	51.5	59.8	857.7	11.3	595.3	38.7	55.1	689.2	11.5	605.5	42.2	51.6	699.4
8	collapse	56	29.2	1571.1	48.2	172.5	1791.7	13.5	809.6	40.0	71.6	921.2	14.0	786.9	48.2	73.9	909.0	11.5	695.1	39.8	63.4	798.3	11.5	695.1	38.6	59.8	793.5
9	write	106	69.2	2035.0	48.6	52.8	2136.4	14.3	557.8	44.6	65.1	667.5	15.5	545.5	58.6	75.1	679.2	11.2	442.9	35.4	56.9	535.1	12.0	484.0	43.5	63.9	591.5
10	interpolate	108	41.8	2018.0	110.4	245.2	2373.7	25.5	1298.5	91.0	100.3	1489.8	27.7	1533.8	105.0	99.7	1738.5	20.5	1089.1	93.1	82.1	1264.3	20.7	1095.8	87.2	83.9	1266.9
11	matmul	109	37.2	2182.0	126.2	28.7	2337.0	21.0	1204.7	90.4	112.1	1407.1	21.7	1271.4	106.7	127.3	1505.4	16.3	976.3	80.6	98.6	1155.5	17.3	1015.3	85.9	102.7	1203.8
12	idctcol	114	41.8	1781.2	97.3	126.1	2004.6	22.8	998.3	75.4	183.6	1257.3	16.5	757.9	81.6	176.0	1015.5	11.7	533.5	48.8	149.6	731.9	12.2	535.4	54.6	156.1	746.0
13	jpeg	134	36.8	1625.3	139.7	167.2	1932.2	26.2	1151.1	105.1	159.0	1415.3	23.2	1162.8	132.6	172.5	1467.9	15.8	759.4	92.2	143.7	995.3	16.8	768.3	96.0	147.3	1011.6
14	smooth	197	76.2	3963.1	217.8	53.4	4234.2	31.3	1785.2	170.2	136.1	2091.5	33.7	1854.8	203.7	177.8	2236.3	26.2	1516.6	161.3	141.4	1819.3	26.5	1533.5	176.5	147.8	1857.9
15	invert	333	100.5	5971.1	437.9	237.0	6646.0	58.5	3398.9	338.1	292.2	4029.2	57.8	3547.3	398.9	306.8	4253.0	38.3	2353.4	333.6	224.1	2911.1	41.3	2476.6	342.6	259.3	3078.5
Average		92	34.1	1693.1	89.8	129.3	1912.2	18.0	947.2	72.1	89.1	1108.4	17.8	964.0	85.5	96.0	1145.5	13.4	732.2	66.1	77.2	875.5	13.8	749.7	68.9	80.8	899.4
FALLS % Improv.			59.4%	55.7%	23.3%	37.5%	53.0%	22.9%	20.9%	4.3%	9.3%	18.8%	22.4%	22.2%	19.4%	15.8%	21.5%	-3.6%	-2.4%	-4.3%	-4.7%	-2.7%	0.0%	0.0%	0.0%	0.0%	0.0%

TABLE XIII

AVERAGE ARCHITECTURE AREA, MAX CONGESTION AND INTERCONNECTION COMPARISON AMONG COMPETING ALGORITHMS USING THE NON-TRIVIAL LIBRARY

DFG		LS			FDS			SA			ILP			FALLS		
		Arch. Area	Max Conges.	# of Intercon.	Arch. Area	Max Conges.	# of Intercon.	Arch. Area	Max Conges.	# of Intercon.	Arch. Area	Max Conges.	# of Intercon.	Arch. Area	Max Conges.	# of Intercon.
1	hal	428.5	5.0	10.7	373.4	5.3	10.8	387.5	5.2	11.2	371.3	5.3	10.5	372.9	5.5	11.2
2	horner	491.3	7.3	12.2	335.9	7.7	11.7	349.5	8.3	11.5	318.9	7.5	10.5	316.2	7.2	9.5
3	arf	799.2	12.2	20.0	387.3	9.8	15.5	395.6	10.8	18.2	307.7	8.2	11.7	315.5	7.7	9.8
4	motion	1345.1	8.2	32.8	844.8	9.7	26.8	879.4	9.8	28.8	800.5	9.2	24.3	787.8	9.7	25.2
5	ewf	451.9	6.5	8.0	236.6	8.7	11.7	226.9	9.3	13.7	166.7	6.3	7.7	166.0	6.3	7.5
6	h2v2	426.0	10.0	25.0	330.7	10.0	16.3	281.8	10.3	20.7	267.9	9.3	16.3	284.2	9.3	16.7
7	feedback	1285.3	10.3	45.3	837.7	13.0	39.8	857.7	13.0	45.8	689.2	11.5	35.0	699.4	12.3	36.5
8	collapse	1791.7	14.5	60.2	921.2	14.7	39.5	909.0	14.7	44.8	798.3	14.3	36.7	793.5	14.2	35.7
9	write	2136.4	15.7	94.7	667.5	11.5	41.0	679.2	13.3	51.2	535.1	11.2	32.7	591.5	12.0	38.3
10	interpolate	2373.7	15.5	106.2	1489.8	18.3	78.3	1738.5	19.7	90.3	1264.3	17.8	74.0	1266.9	17.0	71.3
11	matmul	2337.0	17.0	111.7	1407.1	18.3	74.2	1505.4	17.8	85.3	1155.5	16.3	64.0	1203.8	16.0	67.8
12	idctcol	2004.6	22.7	110.8	1257.3	22.2	79.5	1015.5	21.8	77.0	731.9	18.2	52.2	746.0	18.7	56.7
13	jpeg	1932.2	23.5	124.8	1415.3	23.2	93.2	1467.9	25.7	108.2	995.3	21.7	76.5	1011.6	21.8	79.3
14	smooth	4234.2	19.0	203.3	2091.5	21.0	127.8	2236.3	25.7	151.7	1819.3	18.0	119.2	1857.9	17.5	127.3
15	invert	6646.0	22.2	355.0	4029.2	30.0	251.2	4253.0	31.5	287.8	2911.1	28.5	226.5	3078.5	27.0	236.5
Average		1912.2	14.0	88.0	1108.4	14.9	61.2	1145.5	15.8	69.7	875.5	13.6	53.2	899.4	13.5	55.3
FALLS Improv.		53.0%	3.5%	37.2%	18.8%	9.5%	9.6%	21.5%	14.7%	20.7%	-2.7%	0.6%	-4.0%	0.0%	0.0%	0.0%

TABLE XIV

AVERAGE NUMBER OF FUS AND AREA (INCLUDING FU AREA AND ARCHITECTURAL AREA) COMPARISONS AMONG COMPETING ALGORITHMS USING THE TRIVIAL LIBRARY

DFG		# of ops	LS					FDS					SA					ILP					FALLS				
			nFU	FU Area	Mux+Demux Area	Register Area	Arch. Area	nFU	Area	Mux+Demux Area	Register Area	Arch. Area	nFU	Area	Mux+Demux Area	Register Area	Arch. Area	nFU	Area	Mux+Demux Area	Register Area	Arch. Area	nFU	Area	Mux+Demux Area	Register Area	Arch. Area
1	hal	11	4.8	380.5	4.0	14.7	399.1	3.5	276.6	3.7	14.1	294.3	3.5	278.1	5.4	13.5	297.1	3.5	278.1	4.5	12.3	294.9	3.5	276.6	3.7	13.5	293.8
2	horner	18	3.8	301.0	6.2	15.8	323.1	3.2	250.6	5.8	15.3	271.6	3.0	236.9	6.7	17.6	261.2	2.8	223.1	5.3	15.8	244.2	2.8	223.1	4.5	14.7	242.2
3	arf	28	9.7	771.8	14.9	34.6	821.3	5.3	421.8	11.0	30.5	463.3	4.8	383.6	13.8	29.9	427.3	4.5	357.6	7.4	27.6	392.5	4.5	357.6	6.6	24.6	388.8
4	motion	32	12.5	985.6	19.7	31.7	1037.0	7.7	605.1	15.4	20.5	641.0	8.2	644.9	18.2	21.7	684.8	7.3	579.1	15.2	19.9	614.3	7.5	591.4	10.9	21.1	623.4
5	ewf	34	5.0	386.5	6.1	15.3	407.8	4.3	337.7	8.6	21.1	367.4	3.7	282.6	10.1	22.9	315.6	3.3	258.2	6.6	15.8	280.6	3.3	258.2	4.2	17.0	279.4
6	h2v2	51	9.0	678.1	34.4	61.0	773.5	5.2	397.2	13.4	31.1	441.7	4.7	351.3	17.3	35.2	403.7	4.2	314.6	13.9	30.5	359.0	4.2	314.6	8.0	34.0	356.6
7	feedback	53	12.7	982.3	50.9	58.1	1091.3	8.2	635.6	35.4	45.8	716.7	8.0	621.8	41.1	46.3	709.3	7.3	573.0	38.2	44.0	655.2	7.3	573.0	20.3	41.1	634.3
8	collapse	56	15.5	1196.2	58.6	81.0	1335.7	9.5	734.8	50.2	58.1	843.1	9.8	760.8	54.6	62.2	877.5	9.0	695.1	52.0	56.3	803.4	9.0	695.1	26.1	62.2	783.4
9	write	106	43.0	3160.3	111.2	139.6	3411.1	10.8	803.1	103.7	72.2	979.0	11.0	815.4	105.1	71.0	991.5	10.8	803.1	102.2	69.2	974.6	10.8	803.1	50.2	72.2	925.5
10	interpolate	108	25.5	1993.9	121.6	101.5	2217.0	14.8	1164.3	89.3	58.1	1311.7	16.2	1269.8	99.8	72.2	1441.8	13.3	1048.2	86.9	67.5	1202.6	13.3	1048.2	46.7	65.1	1160.1
11	matmul	109	26.5	2057.9	135.5	153.7	2347.1	13.0	1020.7	105.4	82.7	1208.9	14.5	1138.3	120.2	95.0	1353.5	12.7	994.7	110.2	85.1	1190.0	12.7	994.7	55.7	85.1	1135.5
12	idctcol	114	19.5	1520.2	84.6	142.0	1746.8	9.3	725.7	38.1	113.8	877.6	9.3	721.1	58.7	129.7	909.4	7.2	553.0	39.8	114.4	707.3	7.2	553.0	13.9	112.6	679.6
13	jpeg	134	19.8	1527.6	163.2	160.7	1851.6	14.3	1112.2	90.1	112.6	1314.9	14.0	1090.9	115.2	129.1	1335.1	10.7	828.1	83.8	114.4	1026.3	10.8	838.7	36.3	117.3	992.4
14	smooth	197	44.3	3511.7	282.7	196.5	3990.9	22.0	1729.7	205.9	117.9	2053.5	25.0	1955.7	244.0	148.4	2348.2	20.3	1596.7	209.1	115.0	1920.9	21.3	1679.3	114.2	126.1	1919.7
15	invert	333	51.0	4000.2	486.1	323.8	4810.1	35.2	2745.6	426.9	227.0	3399.5	35.8	2819.2	461.4	242.3	3522.9	27.8	2183.5	427.8	193.0	2804.3	28.2	2212.5	209.3	195.4	2617.2
Average		92	20.2	1563.6	105.3	102.0	1770.9	11.1	864.0	80.2	68.1	1012.3	11.4	891.3	91.4	75.8	1058.6	9.7	752.4	80.2	65.4	898.0	9.8	761.3	40.7	66.8	868.8
FALLS % Improv.			51.6%	51.3%	61.3%	34.5%	50.9%	11.9%	11.9%	49.2%	1.8%	14.2%	14.6%	14.6%	55.5%	11.9%	17.9%	-1.2%	-1.2%	49.2%	-2.2%	3.3%	0.0%	0.0%	0.0%	0.0%	0.0%

TABLE XV

AVERAGE ARCHITECTURE AREA, MAX CONGESTION AND INTERCONNECTION COMPARISON AMONG COMPETING ALGORITHMS USING THE TRIVIAL LIBRARY

DFG		LS			FDS			SA			ILP			FALLS		
		Arch. Area	Max Conges.	# of Intercon.	Arch. Area	Max Conges.	# of Intercon.	Arch. Area	Max Conges.	# of Intercon.	Arch. Area	Max Conges.	# of Intercon.	Arch. Area	Max Conges.	# of Intercon.
1	hal	399.1	4.7	7.5	294.3	4.7	6.2	297.1	5.0	6.8	294.9	4.8	6.3	293.8	5.0	6.5
2	horner	323.1	6.7	7.7	271.6	6.2	6.8	261.2	6.2	7.5	244.2	5.8	6.3	242.2	5.8	6.3
3	arf	821.3	11.3	19.2	463.3	8.5	12.8	427.3	8.8	13.8	392.5	7.2	9.7	388.8	7.3	10.7
4	motion	1037.0	9.8	24.5	641.0	9.0	16.8	684.8	9.0	18.8	614.3	8.2	16.3	623.4	8.8	17.3
5	ewf	407.8	6.7	9.0	367.4	8.2	9.8	315.6	7.5	10.5	280.6	6.5	7.5	279.4	6.7	8.2
6	h2v2	773.5	11.2	31.3	441.7	9.2	14.5	403.7	9.2	16.0	359.0	8.2	13.7	356.6	9.7	15.8
7	feedback	1091.3	13.7	42.7	716.7	13.7	29.3	709.3	13.8	32.7	655.2	13.3	30.2	634.3	12.5	29.0
8	collapse	1335.7	16.7	52.2	843.1	15.0	39.8	877.5	15.7	43.0	803.4	15.0	40.2	783.4	14.8	41.7
9	write	3411.1	16.7	108.0	979.0	17.5	69.3	991.5	16.5	70.2	974.6	16.3	68.0	925.5	16.2	69.8
10	interpolate	2217.0	18.2	95.2	1311.7	18.5	64.7	1441.8	19.2	73.3	1202.6	17.8	63.0	1160.1	17.8	63.0
11	matmul	2347.1	24.0	105.7	1208.9	20.7	73.0	1353.5	22.2	83.5	1190.0	21.3	75.7	1135.5	19.7	73.8
12	idctcol	1746.8	18.5	78.2	877.6	16.2	40.8	909.4	17.7	53.7	707.3	15.3	40.3	679.6	15.5	42.2
13	jpeg	1851.6	26.0	117.3	1314.9	19.8	71.0	1335.1	21.8	85.8	1026.3	18.8	65.5	992.4	18.7	66.2
14	smooth	3990.9	21.7	203.3	2053.5	22.8	137.0	2348.2	26.2	162.0	1920.9	23.8	136.7	1919.7	24.7	145.3
15	invert	4810.1	29.0	325.0	3399.5	34.2	271.8	3522.9	34.3	293.8	2804.3	33.8	263.8	2617.2	33.2	265.0
Average		1770.9	15.6	81.8	1012.3	14.9	57.6	1058.6	15.5	64.8	898.0	14.4	56.2	868.8	14.4	57.4
FALLS Improv.		50.9%	7.8%	29.8%	14.2%	3.4%	0.3%	17.9%	7.2%	11.4%	3.3%	0.0%	-2.1%	0.0%	0.0%	0.0%

TABLE XVI

AVERAGE RUNTIME COMPARISON AMONG THE COMPETING ALGORITHMS USING THE NON-TRIVIAL LIBRARY

DFG		Size (# of operations, # of arcs)	LS	FDS	SA	ILP	FALLS
1	hal	11, 8	0.07	11.9	294.5	147	0.62
2	horner	18, 16	0.14	38.9	340.0	337	0.58
3	arf	28, 30	0.22	72.8	471.8	787	0.98
4	motion	32, 29	0.19	43.0	413.6	325	2.02
5	ewf	34, 47	0.20	96.8	640.9	1635	0.98
6	h2v2	51, 52	0.34	158.4	505.5	1625	1.24
7	feedback	53, 50	0.27	76.7	433.6	788	2.29
8	collapse	56, 73	0.36	74.9	433.6	819	1.95
9	write	106, 88	1.28	223.9	645.5	8654	2.88
10	interpolate	108, 104	0.95	199.1	853.6	3031	4.91
11	matmul	109, 116	1.68	228.5	857.3	4522	5.16
12	idctcol	114, 164	4.09	389.8	1033.6	390114	6.80
13	jpeg	134, 169	5.15	501.5	1181.8	20255043	10.22
14	smooth	197, 196	6.32	734.7	2040.9	100033	10.56
15	invert	333, 354	6.02	2391.8	3998.2	849116	26.40
16	rand_1300	1300, 1300	18.72	676558.8	114615	*	69.85
DFG 1-15 Avg		92.3, 99.7	1.82	350	943	1441132	5.17
DFG 1-16 Avg		167.8, 174.8	2.87	42613	8047	*	9.22

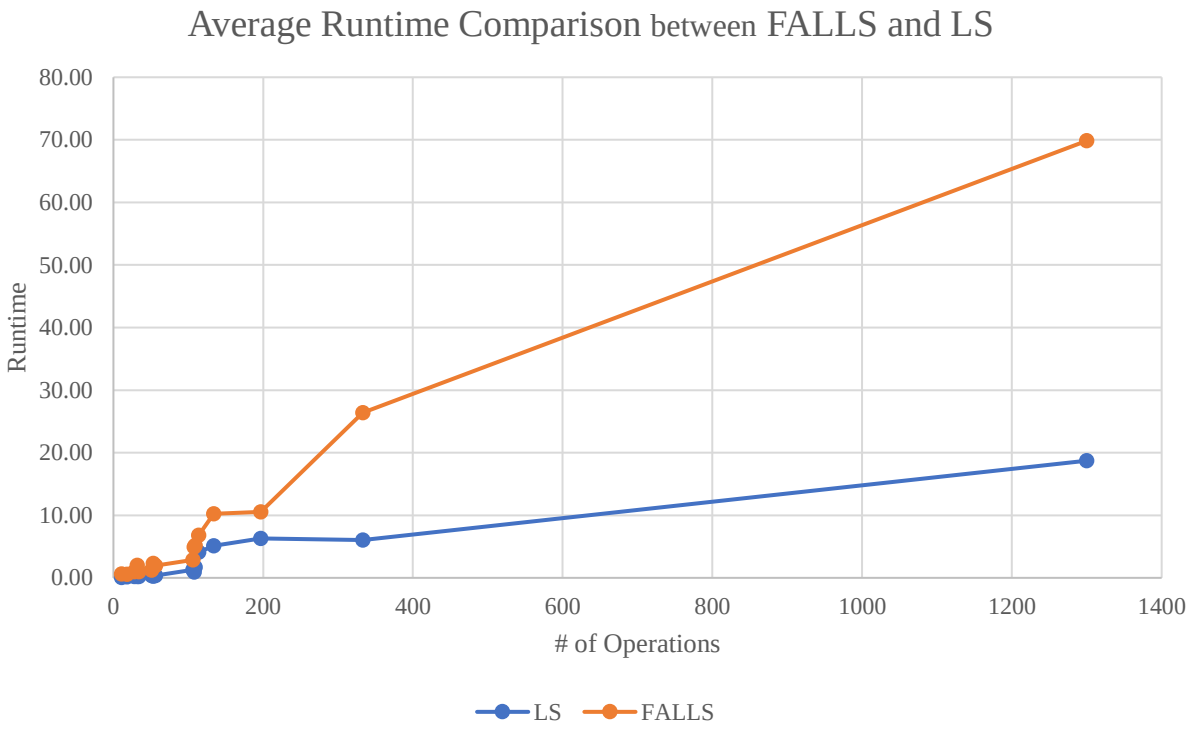


Figure 24: Average runtime comparison in milliseconds between FALLS and LS.

6.3.2. Our ACO Implementation

Since ACO proposed in [25] is the most recent operation scheduling algorithm for FU minimization to the best of our knowledge, it would be more convincing to compare FALLS to ACO to demonstrate the former's efficacy and efficiency. However, our ACO implementation referring to all the details provided in [25] fails to achieve the same or close results explicitly presented in [25][1]. Thus, to be fair, we do not present the results of our ACO implementation across all DFGs in our benchmark suite in the general comparison tables of Section 6.3.1. In TABLE XI, we compare FALLS to only published ACO results for only two DFGs for which [25] has explicitly published absolute results. However, we present our ACO implementation

[1] We contacted the senior author of [19] with our ACO results and asked for ACO parameter values they used (especially for parameter *avg* discussed later in the section) in their ACO runs that are not specified in [19]. The main response we got was ACO is stochastic but should produce stable results, and that our implementation must be different from theirs. We did not get a response to our parameter value query nor an offer of their ACO code.

results and compare these to FALLS' results in subsequent tables. More details regarding our ACO implementation and our ACO experimental results are as follows.

For the implementation, we initially set the two main parameters, the number of ants and the number of iterations to be 10 and 150, respectively, as reported in [19]. The other constant parameters were also all set according to the paper. However, the parameter *avg*, defined as the average size of the decision choices over all solution construction iterations, is not explicitly configured. We make an optimistic configuration as follows. For each ant in any stage of a scheduling solution construction, it probabilistically chooses an unscheduled operation and an cc in its mobility range. Therefore, the product of the average number of unscheduled operation and the average mobility range of an unscheduled operation is *avg* according to the definition. It is thus:

$$avg = \frac{n+1}{2} \times \frac{\sum_{u \in V} MR_u}{n}$$

where n is the number of operations in V and MR_u is the mobility range of operation u. We tried other more complex and accurate *avg* formulations, but the results hardly changed (change was an average of 0.4% improvement).

In TABLE XVII, we compare the results of our ACO implementation (all settings other than for *avg* are the same to [25] to the best of our knowledge) to the explicitly published ACO results. The published results are 17.31% and 26.96% better than our results for the two DFGs in average.

To further explore our ACO implementation, we increased the number of ants and the number of iterations (indicated as a pair in TABLE XVIII), which clearly are parameter configurations whose increase can improve the solution quality of any ant-colony-based algorithm. The results in TABLE XVIII show a consistent, though small, solution quality

improvements with the parameters increased by factors of 2, and a runtime increment that has a similar trend (proportional to the product of the number of ants and the number of iterations). Since the ACO paper [25] does not present the absolute results of the 13 DFGs (other than idctcol and invert), we can only compare FALLS results to our ACO results for these DFGs. The results in TABLE XVIII show that FALLS is on the average 21.3%, 18.3% and 15.3% better than our implementation of ACO for its three parameter configurations in increasing order. Another issue that can be noticed is that our ACO results are very close to FALLS results for DFGs 1 to 9 (which are generally small and less complex) but ACO's solution quality deteriorates significantly when DFG size and complexity increases. Finally, the average runtime of FALLS is 194k smaller than ACO based on our experiments using the same parameters (10 ants and 150 iterations) as in [25]; ACO runtime increases significantly as these parameters increase beyond this initial setting, as Table XVIII shows.

TABLE XVII

FU ALLOCATION COMPARISON BETWEEN EXPLICITLY PUBLISHED AND OUR ACO RESULTS USING THE TRIVIAL LIBRARY

L_c Factor	idctcol (114, 164)					invert (333, 354)				
	Published Results		Our Results		% Diff.	Published Results		Our Results		% Diff.
	nFU	+, *	nFU	+, *		nFU	+, *	nFU	+, *	
1.0	11	5, 6	13	6, 7	18.18%	47	25, 22	68	28, 40	44.68%
1.1	10	4, 6	12	7, 5	20.00%	42	23, 19	58	22, 36	38.10%
1.2	9	4, 5	11	6, 5	22.22%	36	20, 16	50	19, 31	38.89%
1.3	8	3, 5	10	5, 5	25.00%	34	19, 15	47	17, 30	38.24%
1.4	8	3, 5	10	6, 4	25.00%	30	17, 13	40	17, 23	33.33%
1.5	7	3, 4	9	5, 4	28.57%	28	16, 12	39	13, 26	39.29%
1.6	7	3, 4	9	5, 4	28.57%	26	15, 11	35	13, 22	34.62%
1.7	7	3, 4	8	4, 4	14.29%	25	14, 11	31	11, 20	24.00%
1.8	7	3, 4	8	4, 4	14.29%	23	13, 10	32	11, 21	39.13%
1.9	6	3, 3	7	4, 3	16.67%	23	13, 10	31	11, 20	34.78%
2.0	6	3, 3	7	4, 3	16.67%	22	13, 9	29	10, 19	31.82%
Avg	7.8	3.4, 4.5	9.5	4.3, 3.2	20.93%	30.5	17.1, 13.5	41.8	15.6, 26.2	36.90%

TABLE XVIII

AVERAGE NFU AND RUNTIME RESULTS OF FALLS AND OUR ACO IMPLEMENTATION WITH DIFFERENT NUMBER OF ANTS AND ITERATIONS USING THE TRIVIAL LIBRARY

DFG		# of ops	ACO (10, 150)			ACO (20, 300)			ACO (40, 600)			FALLS	
			nFU	% Impro. of FALLS	Runtime (s)	nFU	% Impro. of FALLS	Runtime (s)	nFU	% Impro. of FALLS	Runtime (s)	nFU	Runtime (**ms**)
1	hal	11	3.5	1.3%	46.7	3.5	1.3%	153.0	3.5	1.3%	554.7	3.5	0.62
2	horner	18	2.8	-0.5%	78.6	2.7	-3.7%	285.8	2.7	-3.7%	1095.0	2.8	0.58
3	arf	28	4.9	9.1%	114.7	4.8	7.1%	434.4	4.7	5.1%	1677.3	4.5	0.98
4	motion	32	8.0	6.7%	126.3	7.9	5.5%	451.1	7.7	3.0%	1762.5	7.5	2.02
5	ewf	34	3.4	0.9%	145.8	3.3	-1.8%	561.2	3.2	-4.5%	2168.7	3.3	0.98
6	h2v2	51	4.5	9.1%	313.0	4.5	6.9%	1232.8	4.3	2.5%	4854.9	4.2	1.24
7	feedback	53	8.3	12.8%	258.6	8.0	9.1%	1001.9	7.6	4.1%	3970.0	7.3	2.29
8	collapse	56	9.6	7.1%	211.0	9.5	6.1%	811.9	9.5	6.1%	3207.0	9.0	1.95
9	write	106	11.3	4.1%	730.4	11.2	3.2%	2862.6	10.9	0.7%	11362.5	10.8	2.88
10	interpolate	108	16.5	24.1%	774.6	16.2	21.4%	3036.3	15.5	16.6%	11957.7	13.3	4.91
11	matmul	109	14.9	17.7%	755.8	14.6	15.6%	2971.3	14.3	12.7%	11744.7	12.7	5.16
12	idctcol	114	9.3	29.4%	1088.8	9.1	26.8%	4296.9	8.9	24.3%	16633.5	7.2	6.80
13	jpeg	134	14.1	30.1%	1294.9	13.6	25.9%	5091.2	13.3	22.5%	19891.8	10.8	10.22
14	smooth	197	24.4	14.2%	2190.2	23.6	10.8%	8643.3	23.6	10.8%	34078.6	21.3	10.56
15	invert	333	42.1	49.4%	6936.2	40.7	44.6%	27266.8	39.0	38.5%	107177.2	28.2	26.40
Avg (10 - 15)		165.8	20.2	27.5%	2173.4	19.7	24.2%	8551.0	19.1	20.9%	33580.6	15.6	10.7
Avg (all)		92.3	11.8	21.3%	1004.4	11.6	18.3%	3940.0	11.3	15.3%	15475.7	9.8	5.2

7. CONCLUSIONS

In this thesis, we proposed criteria for low-power high-level synthesis resource library construction and three operation scheduling algorithms for power, energy, resource minimization, respectively. We first introduced high-level synthesis, reviewed previous works for optimizing different objectives in high-level synthesis and categorized the operation scheduling problem. Then we discussed some important classical operation scheduling algorithms, including their objective, complexity and optimality. Motivated by power issues in module selection, we proposed two hypotheses for functional-unit construction for enhanced power optimization and empirically proved them by a flexible simulated-annealing based algorithm PSA. Further, we see that power optimization is not equivalent to energy optimization and the importance of simultaneous power and latency minimization. By improving the classical force-directed scheduling with scheduling probability determination, root-mean-square-based power optimization and greedy scheduling, our LPR-GPS algorithm yields decent energy minimization results. Finally, we solved the classical latency-constrained resource minimization scheduling problem with a new algorithm FALLS that incorporates a novel lookahead technique and fractional search, which produces close to optimal results with very low runtime.

APPENDIX: Copyright Transfer Agreements

Journal of Low Power Electronics

COPYRIGHT TRANSFER AGREEMENT

In order to expedite the publication process, the transfer of copyright from the contributor(s) should be clearly stated to enable American Scientific Publishers (ASP) to disseminate your work to the fullest extent. The following copyright transfer must be signed and returned to the American Scientific Publishers, 26650 The Old Road, Suite 208, Valencia, California 91381, USA, as soon as possible after the manuscript is accepted for publication. The copyright to the unpublished and original research article, including copyright to the cover illustration, abstract forming part thereof should be transferred. By signing this agreement, the contributors (authors) warrant that the entire work is original and unpublished; it is submitted only to this Journal and all text, data, figures/tables or other illustrations included in this work are completely original and unpublished, and these have not been previously published or submitted elsewhere in any form or media whatsoever. Each author(s) agree to transfer all copyrights of this work to the Journal.

Manuscript Title: Co-Exploration of Unit-time Leakage Power and Latency Spaces for Leakage Energy Minimization in High-Level Synthesis

Contributor(s): Ouwen Shi and Shantanu Dutt

is hereby transferred and assigned to American Scientific Publishers for the full term of exclusive copyright and any extensions or renewals of that terms thereof throughout the world, including but not limited to publish, disseminate, transmit, store, translate, distribute, sell, republish and use the Contribution and material contained therein in print and electronic form of the journal and in other derivative works, in all languages and any form of media of expression available now or in the future and to license or permit others to do so.

(1) The contributor(s) explicitly reserve the following rights:

(a) All proprietary rights other than copyright, such as patent rights.

(b) Make photocopies of his/her own work for his/her own classroom teaching and research purposes, and use part of the article and abstract, without revision or modification in own personal compilations provided that (i) such print copies and works are not resold or disseminated by authors and any other recipient parties where recipients are informed that no further photocopying or dissemination of Article is allowed and (ii) reference to the original source of publication and the name of the American Scientific Publishers as copyright holder is clearly stated on any use made of the article. (iii) Contributors should not post the Article on open websites or disseminate through an internet to anyone whatsoever.

(c) The right to use, figures/tables or other illustrations of Article (no text) in subsequent publications of the authors owns works provided that written permission is obtained from Publisher and that a proper acknowledgment is made to the original source of publication and to the Publisher.

(d) Any other use or reproduction in a collective work requires a fee and permission from Publisher.

(e) The right to grant or refuse permission to third parties to republish part of the article or translations thereof. However, the publisher except at the direction of the contributor(s) will not refuse such permission.

(f) American Scientific Publishers reserves the right to use all figures on any of its publication covers from the submitted manuscript. Each author(s) also grants American Scientific Publishers the rights to use his or her name and all biographical data in the article for its or the journal's promotion.

(2) If the article has been prepared by an employee within the duration of his or her employment, the employer reserves the right to make copies of the Work in print format for its own internal use or for promotional purposes only provided that proper reference is made to the original source of publication and to the Publisher. If the manuscript has been prepared as a work made for hire, both employer and employee should sign the copyright transfer.

(3) If the article was prepared under the U.S. Government contract, the transfer of copyright is effective to the extent that such copyright is transferable.

Each contributor(s) warrants that his or her research institution has fully approved the protocol for all scientific studies involving animals or humans and that all experiments of any kind were conducted in compliance with ethical and humane principles of research after approval.

The contributor(s) warrant that the work contains no unlawful or libelous statements and opinions and liable materials of any kind whatsoever, does not infringe on any copyrights, intellectual property rights, personal rights or rights of any kind of others, nor contains any plagiarized, fraudulent, improperly attributed materials, instructions, procedures, information or ideas that might cause any harm, damage, injury, losses or costs of any kind to person or property. The contributor(s) also represent and warrant that they have full power and authority to enter into this Agreement. All contributors are fully responsible for the complete contents of the manuscript. Each contributor(s) agrees to defend, indemnify, and hold harmless American Scientific Publishers and the Editors for any breach of warranties under this agreement. The undersigned hereby transfer the exclusive copyright interests in the above cited manuscript to the American Scientific Publishers, with the consent of all contributors.

Whenever publisher is contacted by third parties for individual permissions to use in a collective work, reprint for library reserve or classroom or otherwise, the undersigned Contributor's or employer's permissions will also be required.

This agreement should be signed by the Contributor(s) or in the case of multiple contributors, by at least one of the contributors who agrees to inform other co-contributors the full terms of this agreement and have their full permission to sign on their behalf.

Contributor's Own Work [X] Work made for hire for Employer [ ] U. S. Government Work [ ]
(Author(s), please tick mark kind of your work)

University of Illinois at Chicago, Dept. of ECE, 851 South Morgan Street, 1020 SEO, Chicago, IL 60607
Name of Employer & Address (Institution/Company)

Shantanu Dutt, Professor
Contributor's Name & Title (Print)


Contributor's Signature

Oct. 16, 2016
Date

FAX (first sheet only) to: **+1-408-904-6997**

Fill in abstract/paper number: _____

IEEE COPYRIGHT FORM

To ensure uniformity of treatment among all contributors, other forms may not be substituted for this form, nor may any wording of the form be changed. This form is intended for original material submitted to the IEEE and must accompany any such material in order to be published by the IEEE. Please read the form carefully and keep a copy for your files.

TITLE OF PAPER (hereinafter, "the Work"):

COMPLETE LIST OF AUTHORS:

IEEE PUBLICATION TITLE (Conference): **Int'l Symposium on Quality Electronic Design (ISQED 2012), Santa Clara, CA USA**

Copyright Transfer

The undersigned hereby assigns to the Institute of Electrical and Electronics Engineers, Incorporated (the "IEEE") all rights under copyright that may exist in and to the above Work, any revised or expanded derivative works submitted to the IEEE by the undersigned based on the Work, and any associated written, audio and/or visual presentations or other enhancements accompanying the Work. The undersigned hereby warrants that the Work is original and that he/she is the author of the Work; to the extent the Work incorporates text passages, figures, data or other material from the works of others, the undersigned has obtained any necessary permissions. **See reverse side for Retained Rights and other Terms and Conditions.**

Author Responsibilities

The IEEE distributes its technical publications throughout the world and wants to ensure that the material submitted to its publications is properly available to the readership of those publications. Authors must ensure that their Work meets the requirements as stated in section 8.2.1 of the IEEE PSPB Operations Manual, including provisions covering originality, authorship, author responsibilities and author misconduct. More information on IEEE's publishing policies may be found at http://www.ieee.org/web/publications/pubtoolsandpolicyinfo/index.html. Authors are advised especially of IEEE PSPB Operations Manual section 8.2.1.B12: "It is the responsibility of the authors, not the IEEE, to determine whether disclosure of their material requires the prior consent of other parties and, if so, to obtain it." Authors are also advised of IEEE PSPB Operations Manual section 8.1.1B: "Statements and opinions given in work published by the IEEE are the expression of the authors."

General Terms

- The undersigned represents that he/she has the power and authority to make and execute this assignment.
- The undersigned agrees to indemnify and hold harmless the IEEE from any damage or expense that may arise in the event of a breach of any of the warranties set forth above.
- In the event the above work is not accepted and published by the IEEE or is withdrawn by the author(s) before acceptance by the IEEE, the foregoing copyright transfer shall become null and void and all materials embodying the Work submitted to the IEEE will be destroyed.
- For jointly authored Works, all joint authors should sign, or one of the authors should sign as authorized agent for the others.

(1)_____ _____
Author/Authorized Agent for Joint Authors **Date**

U.S. Government Employee Certification (where applicable)

This will certify that all authors of the Work are U.S. government employees and prepared the Work on a subject within the scope of their official duties. As such, the Work is not subject to U.S. copyright protection.

(2)_____ _____
Authorized Signature **Dat** **e**

(Authors who are U.S. government employees should also sign signature line (1) above to enable the IEEE to claim and protect its copyright in international jurisdictions.)

Crown Copyright Certification (where applicable)

This will certify that all authors of the Work are employees of the British or British Commonwealth Government and prepared the Work in connection with their official duties. As such, the Work is subject to Crown Copyright and is not assigned to the IEEE as set forth in the first sentence of the Copyright Transfer Section above. The undersigned acknowledges, however, that the IEEE has the right to publish, distribute and reprint the Work in all forms and media.

(3)_____ _____
Authorized Signature **Dat** **e**

(Authors who are British or British Commonwealth Government employees should also sign line (1) above to indicate their acceptance of all terms other than the copyright transfer.)

rev. 041808

IEEE COPYRIGHT FORM *(continued)*

RETAINED RIGHTS/TERMS AND CONDITIONS

1. Authors/employers retain all proprietary rights in any process, procedure, or article of manufacture described in the Work.

2. Authors/employers may reproduce or authorize others to reproduce the Work, material extracted verbatim from the Work, or derivative works for the author's personal use or for company use, provided that the source and the IEEE copyright notice are indicated, the copies are not used in any way that implies IEEE endorsement of a product or service of any employer, and the copies themselves are not offered for sale.

3. Authors/employers may make limited distribution of all or portions of the Work prior to publication if they inform the IEEE in advance of the nature and extent of such limited distribution.

4. In the case of a Work performed under a U.S. Government contract or grant, the IEEE recognizes that the U.S. Government has royalty-free permission to reproduce all or portions of the Work, and to authorize others to do so, for official U.S. Government purposes only, if the contract/grant so requires.

5. For all uses not covered by items 2, 3, and 4, authors/employers must request permission from the IEEE Intellectual Property Rights office to reproduce or authorize the reproduction of the Work or material extracted verbatim from the Work, including figures and tables.

6. Although authors are permitted to re-use all or portions of the Work in other works, this does not include granting third-party requests for reprinting, republishing, or other types of re-use. The IEEE Intellectual Property Rights office must handle all such third-party requests.

INFORMATION FOR AUTHORS

IEEE Copyright Ownership

It is the formal policy of the IEEE to own the copyrights to all copyrightable material in its technical publications and to the individual contributions contained therein, in order to protect the interests of the IEEE, its authors and their employers, and, at the same time, to facilitate the appropriate re-use of this material by others. The IEEE distributes its technical publications throughout the world and does so by various means such as hard copy, microfiche, microfilm, and electronic media. It also abstracts and may translate its publications, and articles contained therein, for inclusion in various compendiums, collective works, databases and similar publications.

Author/Employer Rights

If you are employed and prepared the Work on a subject within the scope of your employment, the copyright in the Work belongs to your employer as a work-for-hire. In that case, the IEEE assumes that when you sign this Form, you are authorized to do so by your employer and that your employer has consented to the transfer of copyright, to the representation and warranty of publication rights, and to all other terms and conditions of this Form. If such authorization and consent has not been given to you, an authorized representative of your employer should sign this Form as the Author.

Reprint/Republication Policy

The IEEE requires that the consent of the first-named author and employer be sought as a condition to granting reprint or republication rights to others or for permitting use of a Work for promotion or marketing purposes.

PLEASE DIRECT ALL QUESTIONS ABOUT THIS FORM TO:
Manager, IEEE Intellectual Property Rights Office, 445 Hoes Lane, Piscataway, NJ 08855-1331.
Telephone +1 (732) 562-3966

CITED LITERATURE

[1] J. Xu, S. Dutt and O. Shi, "Power optimization in HLS Using Multi-Design Function Units," in *poster presentation at 51th Design Automation Conference,* San Francisco, CA, June 2014.

[2] S. Dutt, O. Shi and X. Zhang, "A Novel Power-Driven Hierarchical Framework for Scheduling and Module Selection," UIC Technical Report, 2017.

[3] S. Dutt and O. Shi, "Power-delay product based resource library construction for effective power optimization in HLS," in *International Symposium on Quality Electronic Design,* Santa Clara, CA, Mar. 2017, pp. 229-236.

[4] O. Shi and S. Dutt, "Co-exploration of unit-time leakage power and latency spaces for leakage energy minimization in high-level synthesis," *Journal of Low Power Electronics,* vol. 12, no. 4, pp. 295-308, Dec. 2016.

[5] S. Dutt and O. Shi, "A Fast and Effective Fractional Search and Lookahead Based List Scheduling Algorithm for High-Level Synthesis," in *poster presentation at 54th Design Automation Conference,* Austin, TX, June 2017.

[6] S. Dutt and O. Shi, "A Fast and Effective Lookahead and Fractional Search Based Scheduling Algorithm For High-Level Synthesis," accepted for publication in Proceedings of the *Design Automation and Test in Europe,* Dresden, Germany, Mar. 2018.

[7] Z. Zhang, D. Chen, S. Dai and K. Campbell, "High-level synthesis for low-power design," *IPSJ Transactions on System LSI Design Methodology,* vol. 8, no. 12, pp. 12-25, Feb. 2015.

[8] D. Chen, J. Cong, Y. Fan and J. Xu, "Optimality study of resource binding with multi-Vdds," in *43rd ACM/IEEE Design Automation Conference,* San Francisco, CA, Jul. 2006, pp. 580-585.

[9] Y. Chen, Y. Xie, Y. Wang and A. Takach, "Minimizing leakage power in aging-bounded high-level synthesis with design time multi-Vth assignment," in *15th Asia and South Pacific Design Automation Conference (ASP-DAC),* Taipei, Taiwan, Jan. 2010, pp. 689-694.

[10] K. S. Khouri and Niraj K. Jha, "Leakage Power Analysis and Reduction During Behavioral Synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 10, no. 6, pp. 876-885, Dec. 2002.

[11] 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, Aug. 2003.

CITED LITEREATURE (Continued)

[12] S.-H. Huang, W.-P. Tu and B.-H. Li, "High-Level Synthesis for Minimum-Area Low-Power Clock Gating," *Journal of Information Science and Engineering*, vol. 28, no. 4, pp. 971-988, Sep. 2012.

[13] P. G. Pauline and J. P. Knight, "Force-Directed Scheduling in Automatic Data Path Synthesis," in *24th Design Automation Conference*, Miami Beach, Florida, Jul. 1987, pp. 195-202.

[14] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 6, pp. 661-679, Jun. 1989.

[15] S. Gupta and S. Katkoori, "Force-directed scheduling for dynamic power optimization," in *IEEE Computer Society Annual Symposium on VLSI*, Pittsburgh, PA, Aug. 2002, pp. 68-73.

[16] A. K. Allam and J. Ramanujam, "Modified Force-Directed Scheduling for Peak and Average Power Optimization using Multiple Supply-Voltages," in *IEEE International Conference on Integrated Circuit Design and Technology (ICICDT-06)*, Padova, Italy, May 2006, pp. 1-5.

[17] H. D. Cheng and L. Wang, "Energy-function approach to high-level synthesis scheduling," in *IEEE International Symposium on Circuits and Systems*, San Diego, CA, May 1992, pp. 2961-2964.

[18] Z. Zhao, J. Bian, Z. Liu, Y. Wang and K. Zhao, "High Level Synthesis with Multiple supply Voltages for Energy and Combined Peak Power Minimization," in *IEEE Asia Pacific Conference on Circuits and Systems*, Singapore, Singapore, Dec. 2006, pp. 864-867.

[19] R. Liu, S. Chen and T. Yoshimura, "Post-scheduling frequency assignment for energy-efficient high-level synthesis," in *IEEE Asia Pacific Conference on Circuits and Systems*, Kuala Lumpur, Malaysia, Dec. 2010, pp. 588-591.

[20] J.-H. Lee, Y.-C. Hsu and Y.-L. Lin, "A new integer linear programming formulation for the scheduling problem in data path synthesis," in *IEEE International Conference on Computer-Aided Design (ICCAD-89)*, Santa Clara, CA, Nov. 1989, pp. 20-23.

[21] C.-T. Hwang, J.-H. Lee and Y.-C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 4, pp. 464-475, Apr. 1991.

CITED LITEREATURE (Continued)

[22] W. F. J. Verhaegh, E. H. L. Aarts and J. H. M. Korst, "Improved force-directed scheduling," in *European Design Automation Conference (EDAC)*, Amsterdam, Netherlands, Feb. 1991, pp. 430-435.

[23] W. F. J. Verhaegh, P. E. R. Lippens, E. H. L. Aarts, J. H. M. Korst, A. v. d. Werf and J. L. v. Meerbergen, "Efficiency improvements for force-directed scheduling," in *1992 IEEE/ACM International Conference on Computer-Aided Design (ICCAD-92)*, Santa Clara, CA, Nov. 1992, pp. 286-291.

[24] J. A. Nestor and G. Krishnamoorthy, "SALSA: a new approach to scheduling with timing constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 8, pp. 1107-1122, Aug. 1993.

[25] G. Wang, W. Gong, B. DeRenzi and R. Kastner, "Ant Colony Optimizations for Resource- and Timing-Constrained Operation Scheduling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 6, pp. 1010-1029, Apr. 2007.

[26] M. J. M. Heijligers and J. A. G. Jess, "High-level synthesis scheduling and allocation using genetic algorithms based on constructive topological scheduling techniques," in *IEEE International Conference on Evolutionary Computation*, Perth, WA, Dec. 1995, pp. 56-61.

[27] A. Sharma and R. Jain, "InSyn: integrated scheduling for DSP applications," *IEEE Transactions on Signal Processing*, vol. 43, no. 8, pp. 1966-1977, Aug. 1995.

[28] S. Gupta, N. Dutt, R. Gupta and A. Nicolau, "SPARK: a high-level synthesis framework for applying parallelizing compiler transformations," in *16th International Conference on VLSI Design*, New Delhi, India, Jan. 2003, pp. 461-466.

[29] S. J. Beaty, "List Scheduling: Alone, with Foresight, and with Lookahead," in *the First International Conference on Massively Parallel Computing Systems*, Ischia, Italy, May 1994, pp. 343-347.

[30] L. F. Bittencourt, R. Sakellariou and E. R. M. Madeira, "DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm," in *18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, Pisa, Italy, Feb. 2010, pp. 27-34.

[31] C. Chantrapornchai, E. H.-M. Sha and X. Hu, "Efficient algorithms for finding highly acceptable designs based on module-utility selections," in *9th Great Lakes Symposium on VLSI*, Ypsilanti, MI, Aug. 2002, pp. 128-131.

CITED LITEREATURE (Continued)

[32] L. Goodby, A. Orailoglu and P. M. Chau, "Microarchitectural synthesis of performance-constrained, low-power VLSI designs," in *IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD-94)*, Cambridge, MA, Oct. 1994, pp. 323-326.

[33] M. Ishikawa and G. D. Micheli, "A module selection algorithm for high-level synthesis," in *IEEE International Sympoisum on Circuits and Systems*, Singapore, Singapore, Jun. 1991, pp. 1777-1780.

[34] T. Wiangtong, P. Y. K. Cheung and W. Luk, "Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware-Software Codesign," *Design Automation for Embedded Systems*, vol. 6, no. 4, pp. 425-449, Jul. 2002.

[35] M. D. Huang, F. Romeo and A. S. Vincentelli, "An efficient general cooling schedule for simulated annealing," in *IEEE International Conference on Computer Aided Design*, Santa Clara, CA, Nov. 1986, pp. 381-384.

[36] B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs, Second Edition: Oxford University Press, Oct. 2009.

[37] "ExPRESS Benchmarks," [Online]. Available: http://www.ece.ucsb.edu/EXPRESS/benchmark/. [Accessed 19 8 2017].

[38] X. Tang, H. Zhou and P. Banerjee, "Leakage power optimization with dual-vth library in high-level synthesis," in *42nd Design Automation Conference*, Anaheim, CA, Jun. 2005, pp. 202-207.

[39] N. Wang, S. Chen and T. Yoshimura, "Min-Cut based leakage power aware scheduling in high-level synthesis," in *International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, Mar. 2013, pp. 164-169.

[40] N. Wang, S. Chen and T. Yoshimura, "Min-cut based leakage power aware scheduling in high-level synthesis," in *14th International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, Mar. 2013, pp. 164-169.

[41] K. R. G, "Low power-area designs of 1bit full adder in cadence virtuoso platform," *International Journal of VLSI design & Communication Systems (VLSICS)*, vol. 4, no. 4, pp. 55-64, Aug. 2013.

[42] Z. Huang, "High-Level Optimization Techniques for Low-Power Multiplier Design," PhD thesis, University of California at Los Angeles, 2003.

CITED LITEREATURE (Continued)

[43] S. F. Obermann and M. J. Flynn, "Division algorithms and implementations," *IEEE Transactions on Computers,* vol. 46, no. 8, pp. 833-854, Aug. 1997.

[44] A. Components, "TSMC 0.18µm Process 1.8-Volt SAGE-X Standard Cell Library Databook," Artisan Components, Oct. 2001.

[45] A. Fish, V. Mosheyev, V. Linkovsky and O. Yadid-Pecht, "Ultra low-power dff based shift registers design for cmos image sensors applications," in *11th IEEE International Conference on Electronics*, Tel Aviv, Israel, Dec. 2004, pp. 658-661.

[46] A. Siwach and R. Rishi, "Asymmetric SRAM-Power Dissipation and Delay," *IJCEM International Journal of Computational Engineering & Management,* vol. 11, pp. 28-31, Jan. 2011.

[47] X. Niu and J. Fan, "System-on-a-Chip (SoC) Based Hardware Acceleration for Video Codec," *Optics and Photonics Journal,* vol. 3, pp. 112-117, Jun. 2013.

[48] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," in *8th Design Automation Workshop*, Atlantic City, NJ, Jun. 1971, pp. 155-169.

[49] R. Singh, J. Singh and M. Singla, "Comparative analysis of tg based16 bit adders using 180 nm technology," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering (IJAREEIE),* vol. 2, no. 10, Oct. 2012.

[50] X. Zhang and S. Dutt, "PMAPS: A Probability Map based Power-Driven Simultaneous Scheduling and Module Selection Algorithm for High Level Synthesis," UIC Technical Report, 2017.

VITA

NAME	Ouwen Shi
EDUCATION	Master of Science, Electrical and Computer Engineering, University of Illinois at Chicago (UIC), Chicago, Illinois, Dec. 2017
	Bachelor of Engineering, Electronic Packaging Technology, Xidian University, Xi'an, China, Jun. 2013
EXPERIENCE	Research Assistant at Design Automation and Reconfiguration Technology (DART) Lab, Department of Electrical and Computer Engineering, UIC, Feb. 2014 – July 2017
	Teaching Assistant, Department of Electrical and Computer Engineering, UIC, Aug. 2014 – May 2017
PUBLICATIONS	S. Dutt and O. Shi, "A Fast and Effective Lookahead and Fractional Search Based Scheduling Algorithm for High-Level Synthesis", Design Automation and Test in Europe (DATE), Dresden, Germany, in press.
	S. Dutt and O. Shi, "Power-Delay Product Based Resource Library Construction for Effective Power Optimization in HLS", 18th International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, Mar. 2017, pp. 229-236.
	O. Shi and S. Dutt, "Co-exploration of unit-time leakage power and latency spaces for leakage energy minimization in high-level synthesis", Journal of Low Power Electronics (JOLPE), vol. 12, no. 4, pp. 295-308, Dec. 2016.
POSTER PRESENTATIONS	S. Dutt and O. Shi, "A Fast and Effective Fractional Search and: Lookahead Based List Scheduling Algorithm for High-Level Synthesis", 54th Design Automation Conference (DAC), Austin, TX, June 2017.
	J. Xu, S. Dutt and O. Shi, "Power optimization in HLS Using Multi-Design Function Units", 51th Design Automation Conference (DAC), San Francisco, CA, June 2014.