AUTOMATED COMMUNICATION AND FLOORPLAN-AWARE HARDWARE/SOFTWARE CO-DESIGN FOR SOC

BY

JONG BIN LIM

THESIS

Submitted in 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 Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Professor Deming Chen

ABSTRACT

The main objective of modern SoC (system-on-chip) designs is to achieve high-performance while maintaining low power consumption and resource usage. However, achieving such a goal is a difficult and time-consuming engineering task due to the vast design space of hardware accelerators and HW/SW task partitioning. Depending on the partitioning decision, communication between parts of the SoC must be also optimized such that the overall runtime including both computation and communication would be fast. In this thesis, we propose an automated approach to iteratively search for a near-optimal SoC design with minimum latency within the targeted power and resource budget. Our approach consists of the following main components: (1) polyhedral-model-based hardware accelerator design space exploration, (2) modeling of various communication types and integration into LLVM-based integer linear programming for HW/SW task partitioning, (3) fast and efficient search algorithm to extract maximum operating frequency using floorplanner, and (4) back-annotation of extracted information to system level for iterative partitioning. Using FPGA as the target platform, we demonstrate that our approach consistently outperforms the previous state-of-the-art solutions for automated HW/SW co-design by 37.8% on average and up to 75.2% for certain designs.

To my family, for their constant love and support.

ACKNOWLEDGMENTS

I would like to thank Professor Deming Chen for his rich academic advice and guidance. I also would like to thank several fellow students in my research group for discussions and suggestions. Importantly, I would like to thank my parents, sister, and Somin for their endless love and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASIC	Application-Specific Integrated Circuit
AXI	Advanced Extensible Interface
BRAM	Block Random Access Memory
CDFG	Control Data Flow Graph
CPU	Central Processing Unit
DFG	Data Flow Graph
DDR	Double Data Rate
DRAM	Dynamic Random Access Memory
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
HLS	High-Level Synthesis
HW	Hardware
ILP	Integer Linear Programming
IR	Intermediate Representation
LLVM	Low-Level Virtual Machine
OCM	On-chip Memory
PL	Programmable Logic
PLL	Phased-Lock Loop
PS	Processing System
RIP	Randomized Integer-linear Programming

RTL	Register-Transfer Level
SoC	System-on-Chip
STA	Static Timing Analysis
SW	Software
WNS	Worst Negative Slack

CHAPTER 1

INTRODUCTION

Achieving an optimal system-on-chip (SoC) design that efficiently runs applications is a challenging and time-consuming problem. It is also well known that *power*, *performance*, and *area/resource* are the key metrics in hardware design optimality evaluation. Not only does optimal accelerator design for target applications itself involve much engineering effort, but appropriate task partitioning between accelerators and CPU is also a challenging problem. Additionally, connecting those selected accelerators with CPU into an integrated SoC through a communication system such that the overall design becomes optimal in those three aspects is also a tough problem. Therefore, in order to generate an optimal or near-optimal SoC design solution with appropriate task partitioning and hardware accelerator design, modern HW/SW co-design tools must be able to consider many and diverse possible designs, and apply optimizations at various levels of abstraction. Although it may be desirable to develop and model hardware accelerators and overall integrated SoC at register-transfer level (RTL) or gate-level with more accurate chip design parameters such as switching activities, critical path delay, and resource usage, the associated development and simulation time is often too long, especially for large designs. On the other hand, system-level simulation tools allow much faster simulation speed with correct design functionality, but detailed information such as power estimates, resource estimates and operating frequency may not be estimated well. To overcome such drawbacks and take advantage of both fast simulation speed and detailed low-level design information, data at lower abstraction level can be extracted and *back-annotated* into the corresponding higher-level simulation platform. This process of back-annotation (also called layout-aware or layout-driven design method) enables fast system-level simulation with accurate low-level details, thereby allowing designers to explore huge design spaces fast and efficiently.

There have been many research efforts to model and explore design spaces of HW/SW co-design for SoCs (e.g., [1, 2, 3]). However, many of them do not consider layout information during the system partitioning task so that they are not able to accurately model communication latency, which can be very critical for SoC performance. To address this important issue, in this work, we demonstrate a complete C-to-SoC approach that iteratively refines and searches for an optimal integrated-SoC solution for target applications. Our major contributions are the following:

- Consideration of a rich set of hardware accelerator designs by polyhedral based modeling and software implementations for code regions of interest

- Modeling of different types of communication and integration with LLVM compiler-based integer-linear-programming optimization under resource and power budget constraints

- Fast and efficient search of achievable maximum clock frequency using high-level floorplanner

- Iterative refining approach by back-annotation to search for a near-optimal integrated SoC solution

The remainder of the thesis is organized as follows. Chapter 2 provides background and related work. Chapter 3 explains the methodologies and algorithms of our approach. Chapter 4 presents the experimental results of our approach on FPGA as the evaluation platform. Lastly, Chapter 5 concludes the thesis.

CHAPTER 2

RELATED WORK

Many research works have been published on HW/SW co-design or partitioning. Zuo et al. [2] demonstrated Randomized Integer-linear Programming-based Partitioning (RIP) for system-level HW/SW co-design, but assumed that the communication between task graph nodes always happened by writing data to DRAM and reading data from DRAM. Also, many earlier works demonstrated HW/SW partitioning by various algorithms such as mixed and integer linear programming or randomized search. MAGELLAN [4] is a heuristic technique for mapping high-level control-dataflow graph on heterogeneous architectures. It consists of a scheduler and a partitioner, and optimizes the overall latency. Compared to this work, our proposed approach models not only computation but also communication between critical regions. Also, our approach considers frequency of processor and programmable logic differently. Eles et al. [5] formulated HW/SW partitioning as a graph partitioning problem and proposed two heuristics, simulated annealing and tabu search. The authors defined the metric values for partitioning, developed a cost function that guides partitioning, and considered minimization of communication cost and improvement of overall parallelism. However, as the design size grows and design space expands, it becomes much harder to tune the heuristics. Also, solving the graph partitioning problem also becomes time-consuming. Sha et al. [6] proposed two algorithms for the HW/SW partitioning problem to minimize power consumption. However, the communication latency is not considered, and the potential parallelism in the input code is not explored.

Also, many works were published to extract post-layout information and back-annotate to higher-level system design for optimization. Pasricha et al. [7] proposed an automated framework, CAPPS, to explore power-performance tradeoffs in bus matrix communication architecture synthesis. The authors develop performance and energy macromodels, based on which heuristic optimization techniques are applied to bus matrix to reduce components in the design. This work focuses on the modeling and optimization of bus communication architecture itself, without considering HW/SW partition of IP components. They also developed a synthesis approach by utilizing post-layout information such as wire delay and bus cycle time violation to iteratively reach an optimal design solution under certain constraints. However, this work only focused on bus architecture synthesis, and also assumed that the components and accelerators are already provided and left with final bus connection. Also this work did not fully consider how the integrated design could give the shortest program execution latency. Zheng et al. [8] demonstrated that post-layout critical path information can be back-annotated to a high-level synthesis (HLS) engine to generate a Verilog code such that the identified critical path is removed. A number of critical paths are identified to let the HLS engine iteratively improve the quality of Verilog code with less critical path delay.

CHAPTER 3

FRAMEWORK

3.1 Overview

The goal of the framework is to efficiently search for a near-optimal SoC design point and generate a complete design solution by utilizing post-layout information and integrating different communication types. The critical code region in an application's C code is a code region of interest that could potentially be accelerated by hardware accelerators, or could still be run on CPU based on the system task partitioning. Let Ω be the space or discrete set of all possible **SoC designs**, and let n be the number of **critical code regions**, and i be the index of the i^{th} critical code region. Then, let A_i be a discrete set of all possible candidate **hardware accelerator** designs for the i^{th} critical code region, and let S_i be a discrete set of **software** implementation for the i^{th} critical code region. Lastly, let Φ be a discrete set of all possible **clock frequencies** that the supported phase-locked loop (PLL) can provide for only accelerators. CPU has its own fixed clock frequency, which is much faster than clock for accelerators. $|\Omega|$ is often huge because of all these categorized variables above. Thus, with these definitions, the $|\Omega|$ is determined by Equation 3.1.

$$|\Omega| = \prod_{i=1}^{n} (|A_i|) \cdot |\Phi| \tag{3.1}$$

where $|\cdot|$ denotes the size of finite discrete sets. $|\Omega|$ can increase exponentially with respect to the number of critical code regions, n. $|A|$ is determined by how hardware accelerator modeling is performed as explained in Section 3.2. $|S|$ is always 1 assuming that our SoC design has only one CPU. Also, to avoid excessive increase of $|\Omega|$, the communication architecture of the SoC is assumed to have AMBA®AXI4 protocol where accelerators are connected via memory-mapped interfaces. It is also assumed that there is one PLL that provides clock signals to all accelerators.

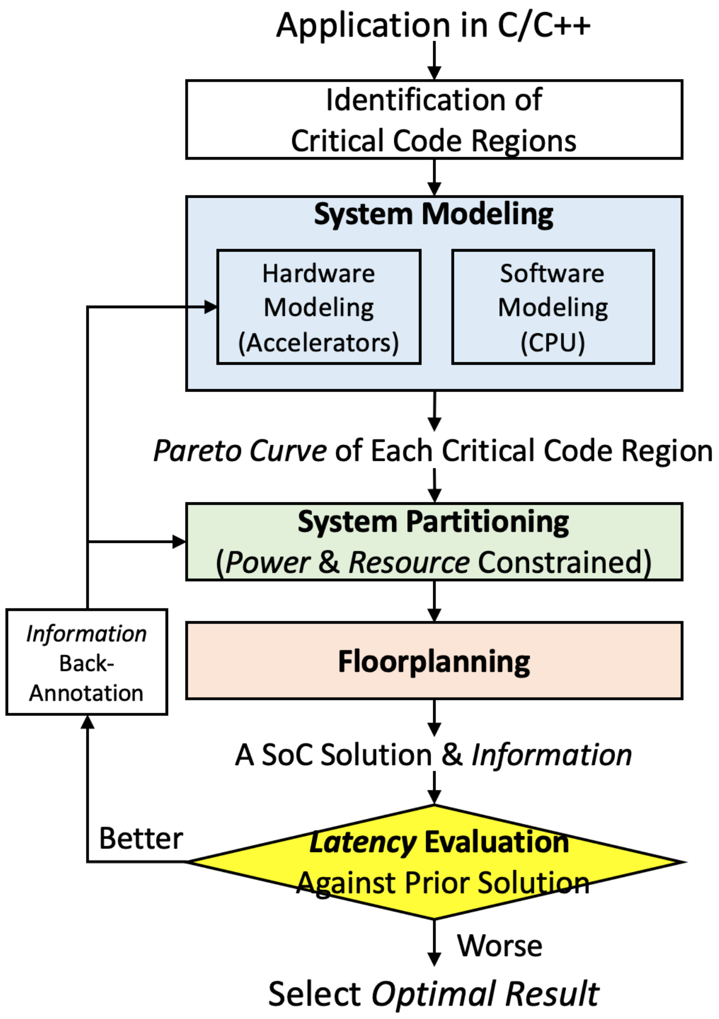

Application in C/C++
Identification of Critical Code Regions
System Modeling
Hardware Modeling (Accelerators)
Software Modeling (CPU)
Pareto Curve of Each Critical Code Region
System Partitioning
(*Power* & *Resource* Constrained)
Floorplanning
Information Back-Annotation
A SoC Solution & *Information*
Latency Evaluation Against Prior Solution
Better
Worse
Select *Optimal Result*

Figure 3.1: Overview of Framework

The overview of the framework (Figure 3.1) is a meta-heuristic approach in which the algorithm iteratively traverses the solution space toward a near-optimal point. Thus, choosing a good starting point in Ω is important to reach a near-optimal point in the least possible iterations. Starting point in Ω is determined by initial system modeling and partitioning without floorplan and communication information. Then, RTL synthesis and floorplanning are correspondingly performed to collect worst negative slack (WNS) and maximum achievable clock frequency of accelerators in Φ. This information is then back-annotated into the system-level partitioning stage and task graph node information to iteratively perform a more realistic implementation which now considers timing details. Since only the cycle-accurate simulation is performed at the system-level, iterative back-annotation of the post-layout information enables a search for a better design solution. Also, the application code is represented in the form of a task graph as shown in Figure 3.2, where each node has information of task-specific execution runtime or data communication runtime, depending on the node type – *computation node* or *communication node*. Figure 3.2 also shows how the task graph can be used to represent the data control flow of applications, and demonstrates how certain nodes can be duplicated to extract task-level parallelism, as explained in Chapter 4. Here, SW-mapped computation nodes and HW-mapped computation nodes are displayed in blue and green, respectively. Also, there exist communication nodes in orange between any two data-dependent computation nodes. In other words, since two computation nodes are data-dependent, there must exist a communication node to represent the communication. Then, the total program execution runtime is identified as an evaluation metric for comparison against the previous design solution in Ω. This search process continues until convergence, which means no better design result is generated.

3.2 System Modeling

3.2.1 Critical Code Identification

In system modeling stage, both hardware modeling for accelerators and software modeling for CPU are performed for critical code regions. Prior to the actual system modeling stage is the *identification* of critical code region. Often, a critical code region is referred to as a *critical loop* since the majority of lengthy computation time is consumed in loop regions. Our tool identifies the critical loops by profiling the application on CPU and examining the runtime of the critical code block. If the runtime of the code of interest exceeds

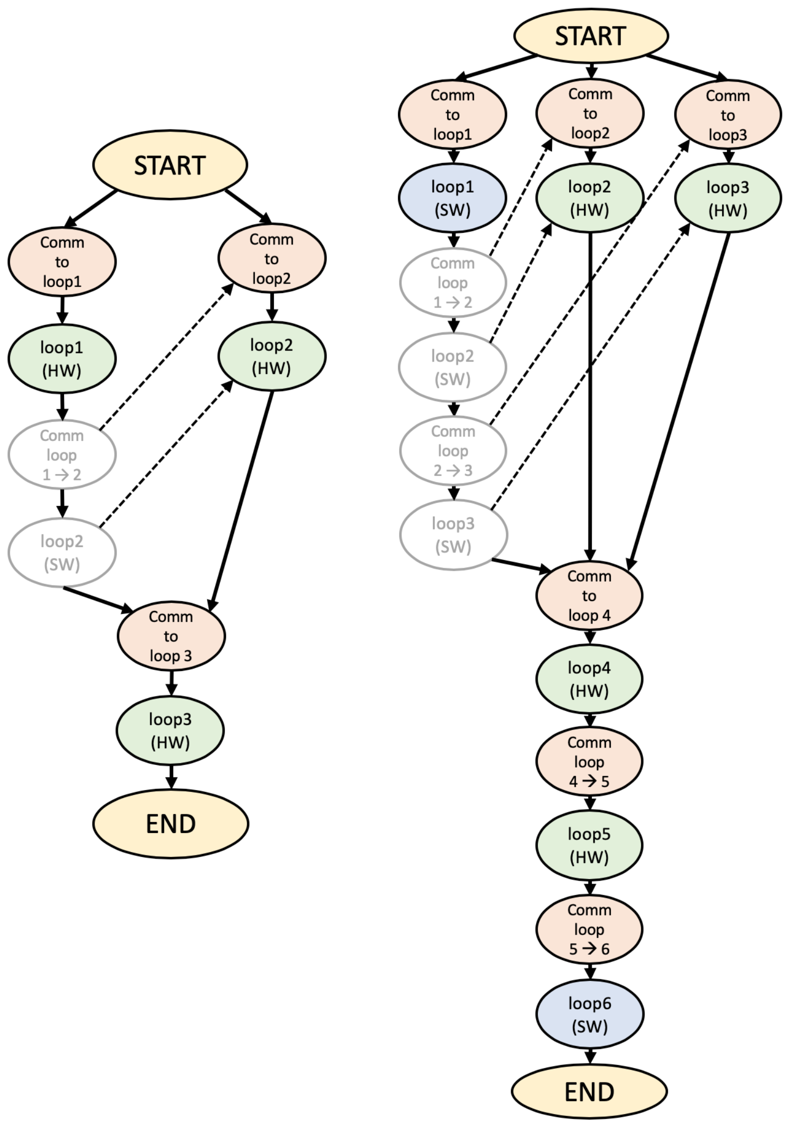


Figure 3.2: Task Graph Reformation of 3-mm (left) and RSA (right)

a certain threshold value, then it is noted to a designer as a critical code region. Also, by checking the fundamental properties of the code region, such as atomicity, regular behavior, and whether the code is an affine loop, n is determined. Through this stage, the number of critical loops, n in equation 3.1, is determined.

3.2.2 Hardware and Software Modeling

For each of these n critical loops individually, hardware modeling is performed. Sets of rich hardware design candidate implementations are generated by using polyhedral-based automatic SystemC open-source modeling tools [1]. Using a powerful compiler technique such as complex program restructuring through loop fusion and tiling with a polyhedral model [9], C-to-SystemC design flow is utilized to automatically generate numerous implementations. This technique enables fine-grained code transformation into SystemC hardware description considering different parallelism architectures in aspects of tiling and pipelining. By exploring various tile sizes, loop unroll factors and switching activity for each of the n critical loops, n Pareto curves are generated [1]. The number of points of the Pareto curve determine $|A_i|$.

For software modeling, CPU architecture is rather fixed with cores and caches in contrast to huge hardware accelerator design space A_i. Utilizing phase convergence modeling with fine granularity region-of-interest profiling [3], we adopt this automated framework which provides performance and energy of application source code through code-specific CPU profiling using Sniper and gem5.

3.3 Communication Modeling

The data communication latency is an important factor that could potentially change the HW/SW partitioning scheme. Although RIP [2] demonstrates a good HW/SW partitioning approach, it is assumed that the communication always happens through DRAM interface. However, depending on how n nodes are mapped, data may not always travel through DRAM between two consecutive nodes. Instead, source and destination of data communication can vary. Since the mapping decision can be either HW or SW, there are four possible combinations of communications: SW-SW, HW-SW, SW-HW, and HW-HW. In our work, each of these four possible communications is characterized and modeled to work with the existing cycle-accurate simulation and estimate the cycle counts of data transfer. XC7Z045 SoC of Xilinx ZC706 was chosen as target platform.

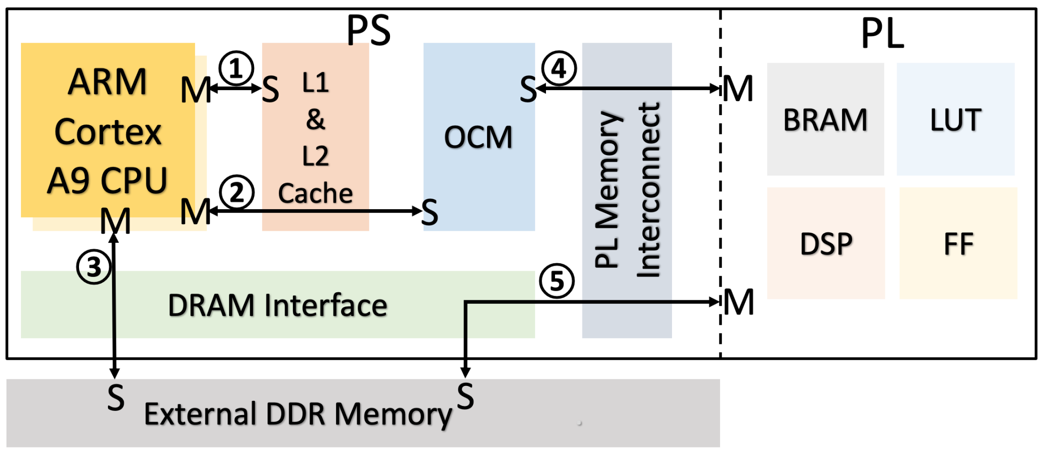


Figure 3.3: XC7Z045 Zynq Processor Architecture and Communication

3.3.1 Software-to-Software Communication

This type of communication happens where there are two consecutive software-mapped nodes in a task graph, notated as S_i and S_{i+1}. In Processing System (PS) of XC7Z045 SoC, there is a dual core ARM Cortex A9 CPU with L1 and L2 caches, 256-KB on-chip memory (OCM), memory interface to DRAM, and memory interconnects for programmable logic (PL), as shown in Figure 3.3 with different communication types labeled from ① to ⑤. OCM is located inside PS whereas BRAM is located in PL. When a node of the task graph is mapped to software, CPU becomes responsible for computation. Although hardware modeling allows designers to freely explore various architectures and design schemes, the general architecture of CPU is rather well established with computation cores, local caches, pipelined execution stages, memory interface, and registers. Since CPU has cache structure and fast clock, communication between two software-mapped nodes is much faster than that for accelerators in PL. Only at the beginning of a program might data need to be loaded from DRAM to "warm up" the caches, where the path is labeled as ③. For the next software-mapped node, the CPU again does the computation by quickly reading data from cache, as shown in the path ① in Figure 3.3. The node dependence naturally leads to data sharing between two nodes in the cache as well.

3.3.2 Hardware-to-Software Communication and Vice Versa

This type of communication happens when a node is mapped to hardware and the next is mapped to software, or vice versa, notated as A_i and S_{i+1}, or S_i and A_{i+1}. The subsequent node has data dependency on the previous node. Unlike the cache hierarchy structure of CPU, accelerators in the PL have simpler memory structure. *Block RAM (BRAM)* is often used as internal memory of an accelerator, which must be provided with data through bus. The path on which data travel is also shown as ④ and ⑤ in Figure 3.3. Data can come from two different sources – *OCM* and external *DRAM*. Although DRAM has big capacity, it is located off-chip and the access time is much slower than that to OCM. Therefore, for HW-SW communication, these two different communication types were characterized and modeled. For experiment setup, an empty accelerator with an internal memory (BRAM) is instantiated and implemented on FPGA. Then, data transfer cycles were measured by using AXI Timer where the data is visible to the programmer through software development kit. For the case of communication between OCM and accelerator, i.e., ④, common data were loaded from DRAM to OCM for faster access. However, since not all data can fit into OCM, hardware accelerators often directly access the DRAM to obtain required data, as shown in ⑤. Unlike the freedom of bus design choice in ASIC, we chose a fixed AXI-4 crossbar interconnect architecture with memory-mapped interface and 64-bit wide bus. We also let the accelerators have master interface to allow direct access to DRAM for high performance.

3.3.3 Hardware-to-Hardware Communication

The last type is HW-HW communication, where two nodes of a task graph are consecutively mapped to hardware (notated as A_i and A_{i+1}). In the work [2], communication is assumed to happen only through DRAM where the data travel from DRAM to accelerators and back to DRAM. However, if the previous accelerator can directly send data to the next accelerator, then these accelerators do not need waste cycles to access DRAM. Using FIFO to connect the two accelerators is a better communication method, where the experimental setup is shown in Figure 3.4. Using the same measurement method, communication from one empty accelerator with BRAM interface to another empty accelerator with BRAM interface via AXI-4 Stream FIFO was measured. The size of FIFO was chosen with width being 32-bit (4-Byte) and depth being 1024, which works well with the chosen FPGA board. The characterization of communication via FIFO was also performed over various ranges of data size, and results are in Chapter 4.

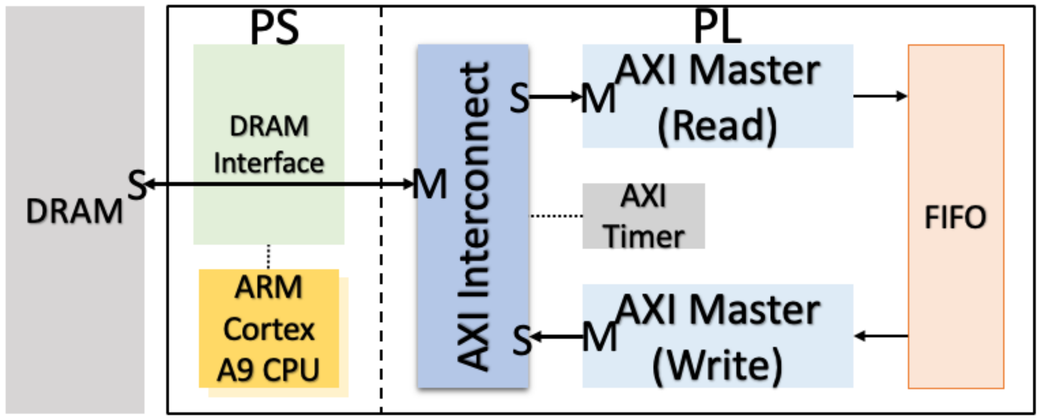


Figure 3.4: Hardware-to-Hardware Communication using AXI-4 Stream FIFO

3.4 System Partitioning

3.4.1 Overview of the Flow

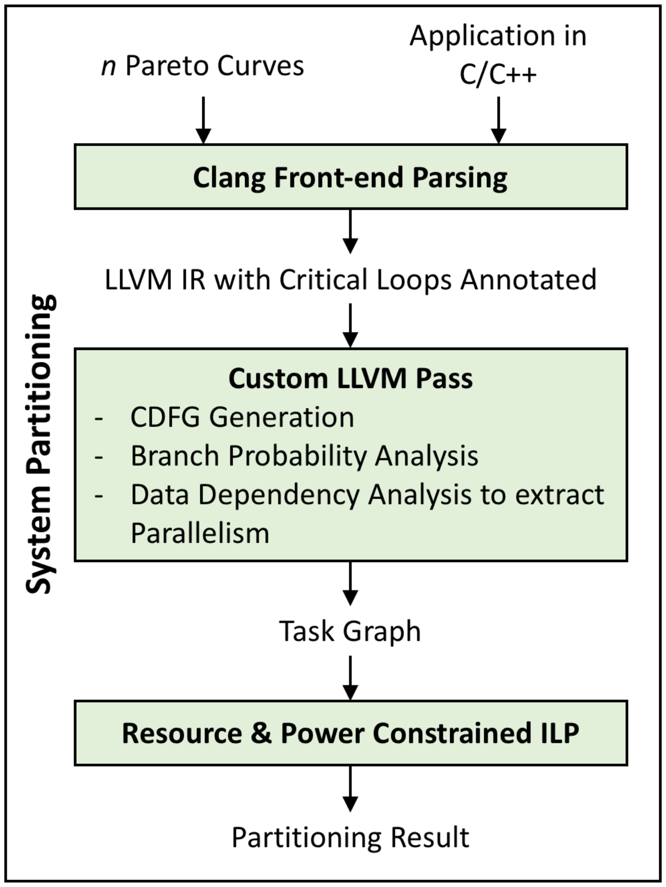
The system partitioning stage performs the actual HW/SW task partitioning, and determines the best implementation of each critical code region such that the overall program runtime is minimized under area/resource and power constraints. Also, just like computation nodes, communication nodes have communication latency information and exist between each pair of computation nodes as long as there is data dependence between the two nodes. LLVM-based Clang front-end parser initially translates the application code to intermediate representation (IR), where the basic block contains information of the code behavior, resource, power and latency for potential hardware and software implementation. Then, custom LLVM pass translates the LLVM IR to a form of control data flow graph (CDFG). The CDFG goes through branch probability and data dependency analysis to extract parallelism, as shown in Figure 3.5. Then, task graph undergoes reformation as shown in Figure 3.2, where each node represents a critical code region with hardware and software latency, power, and resource information. Also, ILP solver takes this information to make a mapping decision such that the overall program latency is minimized.

Randomized Integer-linear Programming (RIP) [2] was adopted to optimize and determine the initial HW/SW partitioning. The initial system partitioning must be performed without data communication latency and post-layout information because the nature of the i^{th} and $(i+1)^{th}$ nodes can only be identified after initial partitioning. This helps to find a good initial search point in Ω. As shown in Figure 3.2, there exists a communication node between any pair of computation nodes when the pair of nodes have a dependence edge between them. Although the initial HW/SW partitioning is performed without communication node information, communication nodes do exist from the beginning with latency values initialized to zero. Only after computation node types are identified either as HW or SW, are the corresponding communication type models used to estimate and update the communication latency between the two data-dependent computation nodes. In other words, the communication latency (or delay) is back-annotated into the communication node, and thus the node gets the updated value. Similarly, when post-layout timing information is extracted, back-annotation works by updating the latency value of each computation node of the task graph. This way, both computation and communication node latency values are updated.

Additionally, through LLVM pass dependence analysis, basic blocks can be duplicated to extract task-level parallelism. Figure 3.2 shows how the task graphs of selected benchmarks have transformed at the end of the iteration. After the iteration has finished and converged, the initially SW-mapped nodes are no longer selected as a result of HW/SW partitioning. Instead, the duplicated nodes are chosen by the ILP solver in an effort to explore different paths from source to sink node to minimize cost function, where cost is a function of latency values of each node (explained more in Section 3.4.2). Taking the benefit of node duplication and task-level parallelism, the ILP solver chooses a path with minimum latency. Because of the duplicated node, the path from source to sink node via originally SW-mapped gray nodes is more expensive than the path via duplicated HW-mapped nodes. Since the path via all SW-mapped node is long with large latency values, the ILP solver does not choose this path as an optimal solution. Instead, the ILP solver takes the path via duplicated nodes from top to bottom as an optimal solution. Therefore, it is more beneficial to select the duplicated HW-mapped node, and thus perform optimal HW/SW task partitioning. This process is again portrayed in Figure 3.2, where the initially SW-mapped nodes are illustrated in gray to indicate that they are no longer chosen and considered. Also, the dashed arrows explain how the initially SW-mapped nodes get duplicated and mapped to HW for parallelism. When a computation node gets duplicated, it needs to be accompanied by the corresponding communication node to represent the required communication latency. With such transformation of the task graph, duplicated computation nodes and communication nodes become a possible decision for the ILP solver. Consequently, as the iteration continues through back-annotation of the communication and post-layout information, our method searches for the near-optimal SoC solution in Ω. Also, the ILP solving step again tries to find a solution iteratively that minimizes the overall execution latency, which consists of both computation and communication.

3.4.2 ILP Formulation

Regarding the actual formulation of the integer-linear programming, in recently published work, Randomized Integer-linear Programming (RIP) [2] was adopted and summarized in this section. The same set of equations and constraints for integer-linear programming (ILP) formulation from [2] was used. Assume that the graph has $|V|$ vertices, where $|V_c|$ are nodes for critical regions. The objective is to search for the path from source node to sink node on the task graph such that the overall latency is minimized under resource and power constraints. Let P_{ij}, L_{ij}, and R_{ij} be the power, latency and resource usage of j^{th} selected design point for the i^{th} critical node, where P_{ij}, L_{ij}, $R_{ij} \in \mathbb{R}$. The objective is to find a mapping decision such that the overall latency L of task graph is minimized. Thus, by representing source



System Partitioning
n Pareto Curves
Application in C/C++
Clang Front-end Parsing
LLVM IR with Critical Loops Annotated
Custom LLVM Pass
- CDFG Generation
- Branch Probability Analysis
- Data Dependency Analysis to extract Parallelism
Task Graph
Resource & Power Constrained ILP
Partitioning Result

Figure 3.5: Resource & Power Constrained System Task Partitioning Flow

node as V_{source} and sink node as V_{sink}, the goal is to minimize the path from V_{source} to V_{sink}. If there can be resource reuse which does not affect the optimality of the latency, then it is factored in to maximize the resource reuse. Decision variables $X_{ij} \in \{0, 1\}$ are also used to determine whether the j^{th} design point of i^{th} node is selected or not. Also, for all vertices V in task graph G, let $S_V \in \mathbb{R}$ denote the time stamp at which the node execution starts. There are several constraints as well. Regarding the variable X_{ij}, there is only one design point selected for implementation, as shown in Equation 3.2. Also, if there is a duplicated node, only one of the two should be selected, where $X_{i'j}$ is a duplicate of X_{ij} for the i^{th} critical node.

$$\sum_j X_{ij} = 1, \forall i \in \{i : V_i \in V_c\}, \text{ and } \sum_j (X_{ij} + X_{i'j}) = 1 \tag{3.2}$$

If two k^{th} designs are identical for all k in task graph, then they can share the resource. For each node, the beginning time stamp must happen after all its predecessor nodes are finished if there is data dependence. Also, there is a resource constraint. For some nodes which do not share the resource with other nodes, the total resource is just a sum of resources of all nodes, as shown in Equation 3.3. But, for certain nodes where the resource may be shared, only the associated resource of j^{th} design point is considered. Also, the sum of those resources must not exceed the total available resource. For FPGA, the resource was represented in BRAM, DSP, FF and LUT.

$$\sum_{i \in \{i : V_i \in V_{C,\text{unshared}}\}} \sum_j R_{ij} X_{ij} + R_{\text{shared}} < R_{\text{total}} \tag{3.3}$$

Lastly, there is a power constraint. The power consumption of the design should not exceed the maximum power budget available. When the task graph is formed without any duplicated nodes, then there are no modules running in parallel. However, if there is any duplicated node where two active nodes in different execution paths on task graph are active, the total power is the sum of all m non-critical node modules and r critical node modules considering whether resource is shared or not, as shown in Equation 3.4.

$$\sum_{i=1}^{m} P_i + \sum_{j=1}^{r} \sum_{k=1}^{S} P_{jk} X_{jk} < P_{\max} \tag{3.4}$$

3.5 Static Timing Analysis and Search for Maximum Clock Frequency

The major purpose of floorplanner is to find WNS through Static Timing Analysis (STA) and apply it to quickly identify maximum achievable clock frequency, a single element in Φ, so that this information can be back-annotated to system task partitioning stage. We utilize high-level FPGA floorplanner, which is integrated in Xilinx Vivado. Then, an efficient search algorithm for maximum clock frequency in Φ was developed and employed.

Naively, clock period can be constrained by the user until WNS becomes minimum to achieve maximum clock frequency in a brute-force manner. However, repetitive floorplanning from the highest frequency down to a realistic maximum frequency that the PLL can support is a time-consuming search process. This kind of linear search method has the computational complexity of O(N), where N is $|\Phi|$. However, WNS can be used to guide the search process more efficiently. Since WNS is the maximum difference between required delay time and arrival time for a signal to travel from one point to another, it is possible to reduce the clock period by the amount of WNS, and thus increase clock frequency. Equation 3.5 explains how the next target frequency can be calculated based on the current frequency and WNS.

$$Next\,freq\,[\text{GHz}] = \left(\frac{1}{Current\,freq\,[\text{GHz}]} - WNS\,[\text{ns}]\right)^{-1} \tag{3.5}$$

Unless the constraint specifies the desired clock frequency, the floorplanner does not try to achieve maximum possible frequency at the first trial. Therefore, in Φ, our search algorithm efficiently finds the f_{max} with timing satisfaction ($WNS > 0$) as soon as possible. Algorithm 1 demonstrates an efficient search algorithm for f_{max} for the provided system after HW/SW partitioning. The search initially starts with the maximum possible frequency that PLL can support, which is 250 MHz. If the timing requirement is satisfied by having $WNS > 0$, then the search process immediately quits and returns 250 MHz. However, it is often unlikely that a design will run at 250 MHz. If $WNS < 0$, the WNS is used to guide the next feasible clock frequency. Instead of linear search in Φ for maximum clock frequency that satisfies timing requirement, the next target frequency is calculated for binary search. At line 8 of Algorithm 1, the binary direction at which the search continues is determined. From this point, linear search is performed for more fine-grained search. If the first binary search result is "$WNS > 0$", then the search continues for higher frequency until timing failure, which is "$WNS < 0$". Otherwise, the search continues for lower frequency until timing success, which is "$WNS > 0$". The best case of the algorithm is a successful search on the first trial. However, it is very unlikely that a design will meet the timing at the first trial. A more feasible case will be a successful search at third trial because the first step will start with the tightest frequency, and the second trial will be either success or failure. The third trial will be the opposite of the second. Considering the worst case, the algorithmic complexity will be O(N) because the linear search could continue until it reaches the limit of the search space.

Algorithm 1: Algorithmic Skeleton to Search for the Maximum Achievable Clock Frequency of PLL for Accelerators

```
   Result: Maximum Achievable Clock Frequency
 1 Set $freq \leftarrow$ max possible frequency;
 2 Run Floorplanner & Report $WNS$;
 3 if $WNS > 0$ then
 4     return $freq$;
 5 else
 6     Update $freq \leftarrow$ next calculated target frequency;
 7     Run Floorplanner & Report $WNS$;
 8     if $WNS > 0$ then
 9         repeat
10             Update $freq \leftarrow$ next calculated target frequency;
11             Run Floorplanner & Report $WNS$;
12         until $WNS < 0$;
13         return the most recent $freq$ where $WNS > 0$ ;
14     else
15         repeat
16             Update $freq \leftarrow$ next calculated target frequency;
17             Run Floorplanner & Report $WNS$;
18         until $WNS > 0$;
19         return $freq$;
20     end
21 end
```

CHAPTER 4

RESULTS

4.1 System Modeling Result

Our complete floorplan and communication-aware C-to-SoC flow is evaluated by five benchmarks, which were adopted from [2]. Three of them (Correlation, Covariance and 3-mm) are from PolyBench/C. The first two benchmarks calculate correlation matrix and covariance matrix as result. 3-mm performs matrix multiplication three times: $A = B \times C, D = E \times F, G = A \times D$. These benchmarks contain affine loop structure, where polyhedral modeling and analysis could explore rich design spaces. RSA is one of the most popular asymmetric encryption and decryption algorithms, which contains original message, key, encrypted message, and decrypted message. The RSA algorithm execution runtime depends on the value of prime numbers for key generation, and 16-bit integer value was chosen for the prime numbers. Among the five benchmarks, AlexNet [10], with 5 convolution layers and 3 fully connected layers and input size of 224 by 224, was tested.

Table 4.1: Possible SoC Designs for Each Benchmark after System Modeling

Benchmark	$\boldsymbol{n}$	$\lvert \boldsymbol{A_i} \rvert$	$\lvert \boldsymbol{\Phi} \rvert$	$\lvert \boldsymbol{\Omega} \rvert$
`Correlation`	2	173	51	1.526×10^6
`Covariance`	2	173	51	1.526×10^6
`3-mm`	3	62	51	1.215×10^7
`RSA`	6	100	51	5.1×10^{13}
`AlexNet`	8	100	51	5.1×10^{17}

First, our tool identifies the number of critical code regions, n, for each of the five benchmarks, and performs polyhedral model based design space exploration for the hardware accelerators. Table 4.1 summarizes the result of system modeling in Section 3.2. Considering $|S_i|$ and $|\Phi|$, the total possible number of design spaces for SoC is determined as shown in $|\Omega|$, which can exponentially increase as the number of n increases. For correlation and covariance benchmarks, $|A_i|$ were both 173. However, since there are only 2 critical loops, the $|\Omega|$ are much less than those of other benchmarks.

It is noticeable that $|\Omega|$ exponentially increases as the number of n increases. For correlation and covariance benchmarks, the $|A_i|$ are both 173. However, since there are only 2 critical loops, $|\Omega|$ of these benchmarks are much smaller than those of other benchmarks.

4.2 Communication Modeling Result

As described in Section 3.3, different types of communication were characterized, and average of data transfer cycles in both directions between the two origins were measured. Thus, an average transfer cycle of both directions was calculated. Then, communication models for different sources and destinations are inferred, and adopted in the integer linear programming based optimization stage by annotating communication latency overhead just like computation latency. Depending on the amount of data between nodes of task graph, cycle counts are estimated and annotated into the communication node of task graph.

First, HW-SW communication (and vice versa) is considered. As shown in Figure 4.1, there is a directly proportional relationship between the data transfer cycle and data size with the regression coefficient R of 0.9999 and slope of 5.05, which means that the data transfer cycle increases by approximately 5.05 cycles per byte, or 20.2 cycles per 4-byte word. Between OCM and BRAM, Figure 4.2 also shows a strong directly proportional relationship with the regression coefficient of 1 and slope of 0.5091. The ratio between the two identified slopes is 9.92, which means the cycle increase rate of data transfer between OCM and BRAM is 9.92 times faster than that between DRAM and BRAM. Looking at the architecture of XC7Z045 in Figure 3.3, OCM is physically much closer to BRAM in the PL side than the external DRAM.

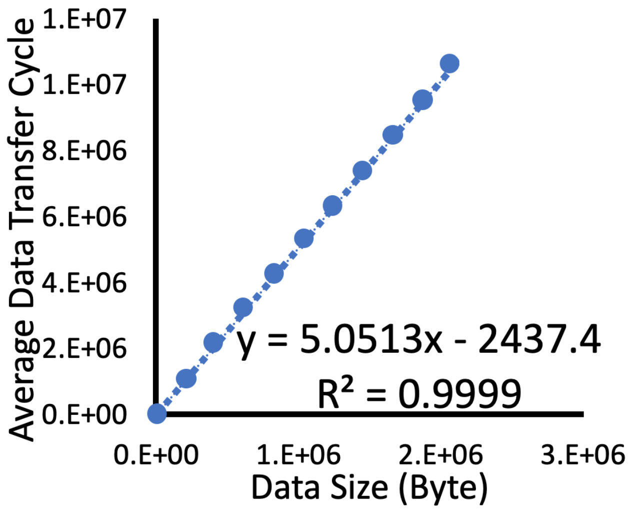


Figure 4.1: Data Transfer Cycle vs. Data Size between DRAM and BRAM

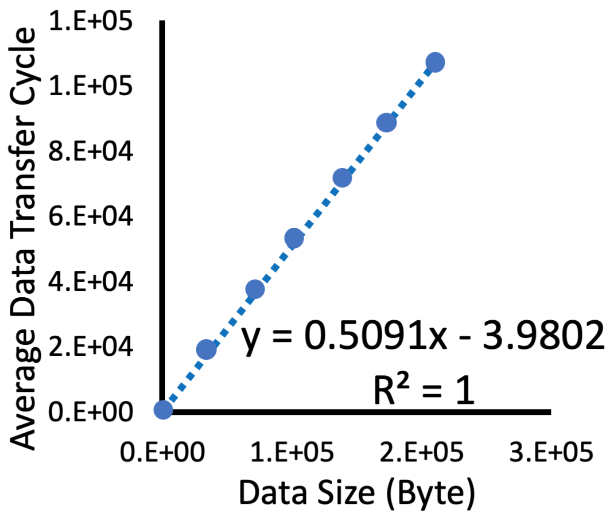


Figure 4.2: Data Transfer Cycle vs. Data Size between OCM and BRAM

The next type of communication is HW-HW communication. To allow efficient communication between the two hardware-mapped accelerators, the designer may let the result data stay on-chip so that the next accelerator can directly use it. Thus, in the case of two consecutive hardware mapped nodes, communication via FIFO is a much better communication than always accessing DRAM. The slope of the blue trendline in Figure 4.3 is 0.25, which can be interpreted as the FIFO's data transfer cycle rate for each byte of data. On the other hand, the red trendline of Figure 4.3 shows a slope of 9.85. Thus, the ratio between the two slopes is 39.4, which means that the communication between the two accelerators by accessing DRAM is

39.4 times slower than the communication through FIFO streamed interface. Due to this significant difference, FIFO enabled communication is a much better communication method. Also, the slope of communication between two BRAMs via DRAM in Figure 4.3 is 9.85 while that between DRAM and BRAM in Figure 4.1 is 5.0513. Thus, the ratio between the two slopes is 1.95, which explains that the communication path of the former is twice the path of the latter. Through the accurate modeling and characterization of different types of communication, numeric models of average data transfer cycle per data size increase were extrapolated and integrated into our HW/SW co-design framework so that appropriate communication method can be chosen by minimizing cost function.

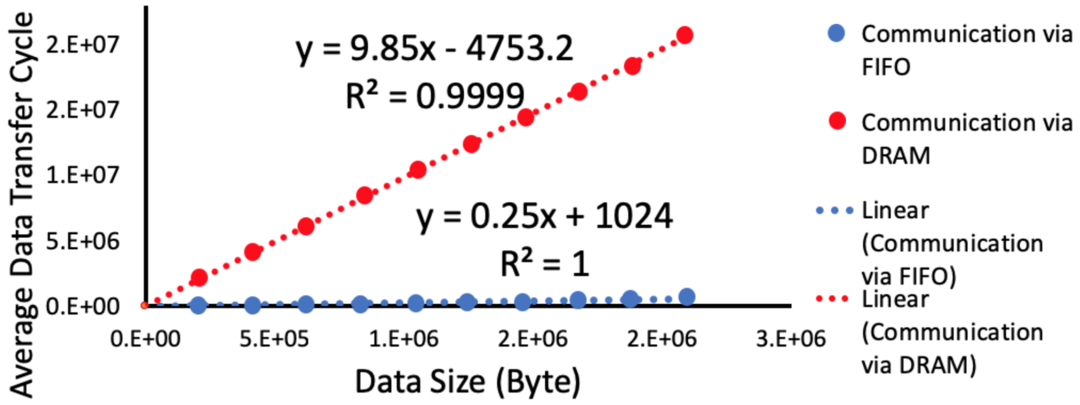


Figure 4.3: Data Transfer Cycle vs. Data Size between Two BRAMs via FIFO Interface and DRAM Interface

4.3 System Partitioning Result

For all benchmarks, our framework generated overall better HW/SW partitioning results. This is significant. Reference [2] used randomized ILP-based method and demonstrated that they achieved optimal or near-optimal HW/SW partitioning solution compared to known optimal solutions obtained through brute-force enumeration. Our work demonstrates that when we incorporate the right communication latency through our layout-aware approach, we can further improve their results significantly. Table 4.2 shows the result of final HW/SW partitioning. Also, Table 4.3 reports both baseline and our final results with 37.78% performance improvement on average. Figures 4.4 and 4.5 show how the iterative partitioning method changed the selection of hardware accelerator from the initial point to the final point. For correlation and covariance, performance improvements were not as significant as the other benchmarks because these had only 2 critical loops and initial design points were already close to the near-optimal design. For 3-mm, there was an improvement of 60.83%. The final result showed that all three critical loops were mapped to hardware accelerators. During the LLVM analysis of system partitioning in Figure 3.5, loop 2 was identified as a duplicated node, and thus loop 1 and loop 2 were both mapped to hardware so that these hardware accelerators can run in parallel. Loop 3 was also mapped to hardware, but was not identified as a duplicated node. Since all three loops were mapped to hardware, a FIFO enabled communication interface between the two adjacent hardware mapped nodes was chosen. The extracted maximum clock frequency of 125 MHz allowed the system partitioning stage to choose from the 58^{th} to the 21^{st} candidate point on the generated Pareto curve, as shown in Figure 4.4. Next, RSA showed the best improvement of 75.15% among the five benchmarks. Although the initial partitioning resulted in two critical loops (3 and 4) being mapped to hardware, the final partitioning showed that four loops (2, 3, 4 and 5) were mapped to hardware as shown in Table 4.2 and Figure 3.2. Floorplanner reported the largest WNS for this benchmark, and thus the maximum achievable clock frequency was reported to be 187.5 MHz. Therefore, the final partitioning scheme completely changed, and new design candidates were selected on Pareto curves of loop 2 and 5. Interestingly, there was a change in design point from the 51^{st} to the 31^{st} point for loop 3, as shown in Figure 4.5. As the iteration continues and generates more hardware mapped nodes, streamed communication via FIFO between two hardware accelerators was also chosen. Lastly, AlexNet initially had 6 loops being mapped to hardware and 2 loops being mapped to software. As shown in Table 4.2, the number of final mapping decisions did not change. However, the design point on Pareto curve of loop 1 was changed from the 42^{nd} point to the 1^{st} point. Also, the floorplanner reported maximum frequency of 111.11 MHz, and consequently, the overall latency improved by 35.17%. Overall, all five benchmarks showed performance improvement with our communication and floorplan-aware partitioning.

Table 4.2: Final Partitioning Result of Five Benchmarks

	Correlation	Covariance	3-mm	RSA	AlexNet
HW	2	2	3	4	6
SW	0	0	0	2	2

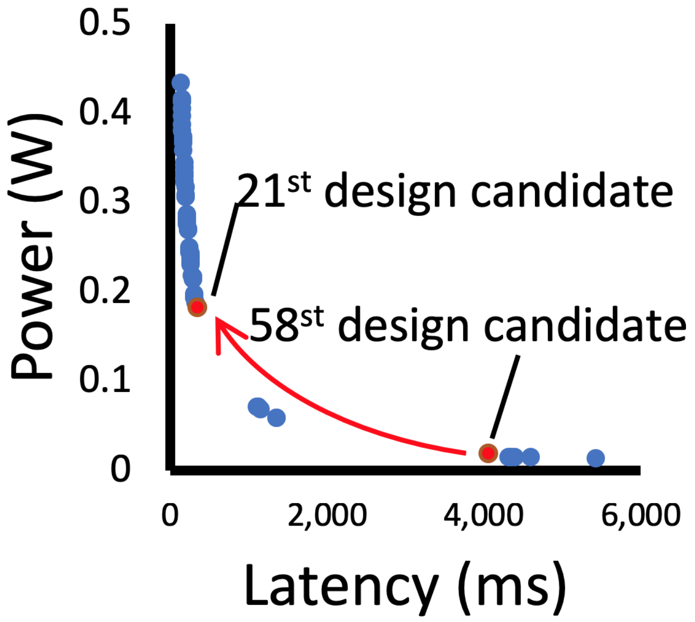


Figure 4.4: Change in Design Candidate Selection over Iterative HW/SW Partitioning for Third Loop of 3-mm

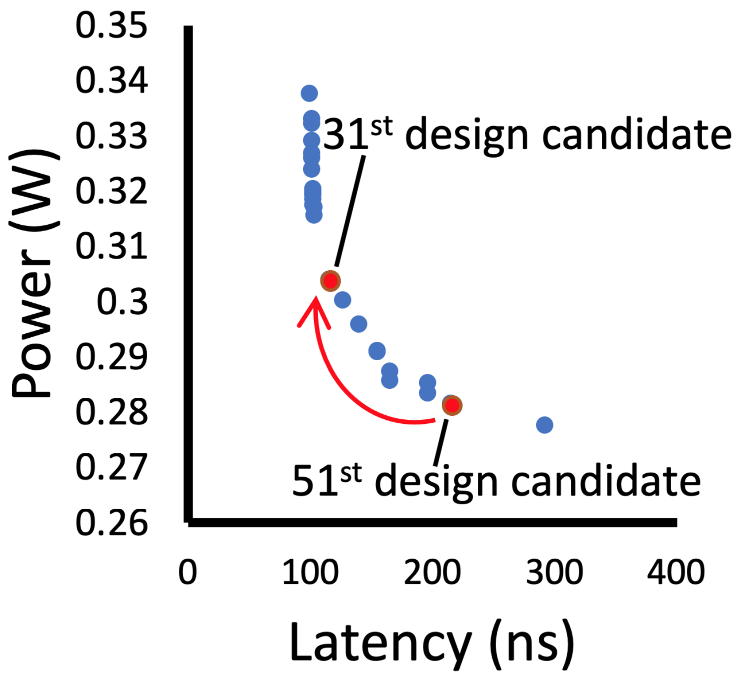


Figure 4.5: Change in Design Candidate Selection over Iterative HW/SW Partitioning for Third Critical Loop of RSA

Table 4.3: Latency Improvement of Final SoC Solution through Iterative HW/SW Partitioning for Five Benchmarks

		Benchmarks				
		Correlation	Covariance	3-mm	RSA	AlexNet
Baseline	Latency (ms)	640	600	450	3690	1040
	Power (W)	5.91	5.02	5.84	2.25	7.75
	BRAM	384	256	382	8	504
	FF	34185	23618	131093	18444	103340
	DSP	812	820	848	72	885
	LUT	84266	61618	80865	18867	104266
Final	Latency (ms)	613.5	529	279.8	2106.8	769.4
	Power (W)	5.9258	4.92	5.86	2.7	7.872
	BRAM	397	392	542	9	705
	FF	25916	25509	49126	18633	108423
	DSP	818	818	840	48	570
	LUT	39446	40795	51552	17341	121686
Perf. Improvement(%)		**4.32**	**13.42**	**60.83**	**75.15**	**35.17**
Avg. Perf. Improvement(%)		**37.78**				

CHAPTER 5

CONCLUSION

The importance of an automated HW/SW co-design tool which considers a wide range of abstraction levels during the HW development process is significant. HW/SW co-design is also an area of research where many scholars studied and published in the past. Although many works were published to optimize certain levels of abstraction in the process of SoC design, data communication and physical delay information, which can only be identified after floorplanning, were not considered much. Here, we not only generate sets of many candidate accelerator designs for each task in given benchmark applications using polyhedral model based analysis, but also perform integer linear programming based HW/SW partitioning to model the application code behavior and iteratively integrate and search for the system such that a near-optimal SoC design is quickly discovered under user-defined power and resource constraint. With several communication models being integrated and post-layout timing information being annotated, our HW/SW co-design tool was able to find a near-optimal SoC solution, and showed performance improvement for all five benchmarks. Some applications with small number of critical loops demonstrated that the result of initial partitioning was already near-optimal. However, some applications with data parallelism and more critical loops demonstrated that identification and back-annotation of WNS and maximum achievable clock frequency could actually help the integer linear programming solver to choose a different partitioning scheme by selecting a different candidate on the Pareto curve, or by changing an initially SW-mapped node to HW-mapped node. Also, exploration of various communication models helped the integer linear programming solver to choose the most appropriate communication interface.

REFERENCES

[1] W. Zuo, W. Kemmerer, J. B. Lim, L.-N. Pouchet, A. Ayupov, T. Kim, K. Han, and D. Chen, "A polyhedral-based SystemC modeling and generation framework for effective low-power design space exploration," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '15. Piscataway, NJ, USA: IEEE Press, 2015. [Online]. Available: http://dl.acm.org/citation.cfm?id=2840819.2840870 pp. 357–364.

[2] W. Zuo, L. N. Pouchet, A. Ayupov, T. Kim, C.-W. Lin, S. Shiraishi, and D. Chen, "Accurate high-level modeling and automated hardware/software co-design for effective SoC design space exploration," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2017, pp. 1–6.

[3] W. Kemmerer, W. Zuo, and D. Chen, "Parallel code-specific CPU simulation with dynamic phase convergence modeling for HW/SW co-design," in *Proceedings of the 35th International Conference on Computer-Aided Design*, ser. ICCAD '16. New York, NY, USA: ACM, 2016. [Online]. Available: http://doi.acm.org/10.1145/2966986.2967063 pp. 79:1–79:8.

[4] K. S. Chatha and R. Vemuri, "MAGELLAN: Multiway hardware-software partitioning and scheduling for latency minimization of hierarchical control-dataflow task graphs," in *Proceedings of the Ninth International Symposium on Hardware/Software Codesign*, ser. CODES '01. New York, NY, USA: ACM, 2001. [Online]. Available: http://doi.acm.org/10.1145/371636.371671 pp. 42–47.

[5] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli, "System level hardware/software partitioning based on simulated annealing and Tabu search," *Design Automation for Embedded Systems*, vol. 2, no. 1, pp. 5–32, Jan 1997. [Online]. Available: https://doi.org/10.1023/A:1008857008151

[6] E. Sha, L. Wang, Q. Zhuge, J. Zhang, and J. Liu, "Power efficiency for hardware/software partitioning with time and area constraints on MPSoC," *International Journal of Parallel Programming*, vol. 43, no. 3, pp. 381–402, Jun 2015. [Online]. Available: https://doi.org/10.1007/s10766-013-0283-4

[7] S. Pasricha, Y. H. Park, F. J. Kurdahi, and N. Dutt, "CAPPS: A framework for power-performance tradeoffs in bus-matrix-based on-chip communication architecture synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 2, pp. 209–221, Feb 2010.

[8] H. Zheng, S. T. Gurumani, K. Rupnow, and D. Chen, "Fast and effective placement and routing directed high-level synthesis for FPGAs," in *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, ser. FPGA '14. New York, NY, USA: ACM, 2014. [Online]. Available: http://doi.acm.org/10.1145/2554688.2554775 pp. 1–10.

[9] W. Zuo, Y. Liang, P. Li, K. Rupnow, D. Chen, and J. Cong, "Improving high level synthesis optimization opportunity through polyhedral transformations," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '13. New York, NY, USA: ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2435264.2435271 pp. 9–18.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf