Speed up numbers in simulation versus actual hardware numbers and FPGA profiling with a probe inserted in Verilog

Prof. Callie Hao, Yalaj Goyal, Chinmay Galande

***Abstract*—The process of simulating accelerated hardware using High-Level Synthesis (HLS) techniques is a crucial aspect of hardware-software co-design. In this study, we investigate the accuracy of simulation latency numbers in predicting actual speed-up, recognizing that simulation results merely represent a prediction of actual hardware performance. To this end, we evaluate five distinct types of computations, each optimized in four different ways, resulting in a total of twenty codes. We collect latency data for each code through simulation runs on C-Synthesis, Co-Simulation, and the actual hardware run. The study aims to demonstrate that the latency numbers obtained from simulations are not a reliable indicator of actual hardware speed-up. To elucidate the reasons for this discrepancy, we investigate whether it originates from computations or memory operations. We propose a modular approach, incorporating a probe into the Verilog code to measure the latency of different modules. Overall, our research aims to contribute to a more comprehensive understanding of the limitations and strengths of simulation techniques for accelerated hardware design.**

***Index Terms*—FPGA, HLS, C-synth, Co-Sim, synthesis, profiling, latency, speed-up.**

I. Introduction

THE development and evaluation of hardware accelerators have become increasingly important in recent years. These specialized computing devices are designed to perform specific tasks more efficiently than general-purpose processors, making them critical components in many applications such as artificial intelligence, data analytics, and scientific simulations.

Performance characterization is a fundamental process in understanding the capabilities and limitations of hardware accelerators. It enables developers to benchmark accelerators against other devices, verify and validate their designs, and optimize their performance. As such, accurate performance characterization is crucial to ensure optimal accelerator performance.

One of the primary metrics used in Vitis High-Level Synthesis (HLS) for performance characterization is simulation latency numbers. These numbers provide valuable insights into the efficiency and performance of HLS code during the simulation stage, which is instrumental in optimizing code for better performance in hardware. However, there are issues when it comes to the real-world application of research numbers based on simulation results. Simulation numbers reported by Vitis HLS differ from the actual runs, creating a false impression of higher speed-ups from implemented optimizations.

To address this discrepancy, this paper aims to investigate the differences between simulation results and hardware runs. Specifically, we aim to explore the reasons for the discrepancy, to identify potential limitations in using simulation numbers as predictors of hardware performance. Ultimately, our research aims to provide a deeper understanding of the capabilities and limitations of performance characterization techniques and to contribute to the development of more accurate and reliable methods for evaluating hardware accelerator performance.

II. Problem description

This project aims to address the discrepancy between the simulation latency numbers and actual speed-up (as seen after implementing on the hardware) of a set of twenty codes, each of which involves five different types of computations with four different optimizations. The study seeks to determine whether the discrepancy arises from the computations or the memory operations. The latencies reported by C-synthesis are critical indicators of the performance parameters of the FPGA design. C-synthesis uses sophisticated algorithms to estimate latency based on RTL implementations derived from high-level design descriptions. C-synthesis creates latency estimates by methodically analyzing the control and data flow and taking into account various optimization approaches used during synthesis. To bridge the gap between simulation and hardware execution, C-RTL co-simulation simulates the design's RTL implementation to provide a more accurate prediction of latencies. This method considers low-level features such as individual hardware components and interconnections, resulting in a more accurate approximation of actual hardware behavior. Even with this increased level of simulation fidelity, differences between co-simulation delay reports and actual hardware implementation might occur. These variances could be due to flaws in the simulation models or inconsistencies between the simulation and hardware execution environments. Inaccurate predictions of computer system performance can lead to poor design decisions and sub-optimal system performance. Simulation tools are commonly used to estimate computer system performance, but their accuracy has been questioned. Therefore, understanding the source of the discrepancy between simulation latency numbers and actual speed-up is crucial for improving the accuracy of simulation tools and optimizing computer systems.

This project is significant because it provides insights into the factors that influence the accuracy of simulation tools like Vitis HLS. By considering the sources of latency in computations and memory operations, the project will provide

guidelines for enhancing the design and optimization of computer systems. Consequently, this study's findings can be used particularly for improving the accuracy of simulation tools like Vitis HLS.

III. Related Work

In the field of FPGA design and performance analysis, some approaches have been proposed to address the challenge of accurately profiling and measuring the runtime of individual modules within an HLS-based design. In this section, we discuss the existing related work that explores different methodologies and tools for module-level profiling in FPGA designs.

One common technique employed in FPGA profiling is the utilization of debug applications available in popular FPGA design tools such as Xilinx Vitis HLS[2]. Vitis gives us the option to time each individual module by passing the module under consideration through a counter implemented on the device itself. This excellent method contains the module of interest within a counter module, allowing the start and finish times of the module to be measured. While this approach provides a simple way to access module-level runtime statistics, it does have some restrictions and trade-offs.

This is a very elegant solution that encloses the module into a counter module and measures the start and end times of the module. One notable drawback of using the debug application approach is the significant overhead it introduces to the profiling process. The insertion of counters within the design can impact the overall execution time and potentially influence the resulting measurements. The added logic and additional routing required for the counters can lead to resource utilization constraints, particularly when dealing with complex designs that are already optimized for maximum performance. Thus, careful consideration must be given to the trade-off between accurate profiling and the potential impact on the overall design performance.

IV. Proposed solution

We will be gathering twenty codes and getting the software numbers for each of them. The special relation in these codes is that they are grouped into 5 groups based on the function they perform. Each group has four different optimizations of the function resulting in different latencies of the same function. This will give us a direct indication of the discrepancy (if present) and its impact on the speed-up numbers. The different functions being implemented are:

1) Vector Addition
2) Shift Register
3) Matrix multiplication
4) Convolution
5) Apply Watermark (apply small x of size 16x16 pixels for input image)

The above codes are referenced from the Xilinx Github repository for Vitis Accel example codes[1]. We chose these specific functions for a number of reasons pertaining to our research. For starters, these functions cover a wide range of computing tasks found in FPGA-based systems. We want to obtain insights into the general behavior of many types of computations using FPGA hardware by investigating these representative functions.

Second, we added alternative optimizations within each function group. These optimizations are intended to increase the performance and efficiency of the corresponding function, resulting in varying latencies for the same operation. We can directly analyze the influence of these optimizations on the observed latency mismatch between simulation and hardware execution by comparing the latencies across the different optimizations.

Furthermore, the availability and widespread use of these functions and their optimizations in the Xilinx community influenced their selection and optimization. For Vitis Accel example codes, we referred to the Xilinx GitHub repository [1]. Commonly used examples assure that our findings and conclusions apply to real-world FPGA development environments.

Lastly, we also added a variation of data size (16-bit or 32-bit) used to send the pixels in the "apply watermark" code. The expectation is that this would allow some insight into latency variation and the size of the data being operated on. The important point to note here is that the total data remains the same in bits, only how the data is packaged may vary.

We can find any significant discrepancies between simulation results and actual hardware performance by methodically evaluating the latency variances among the different optimizations within each function group. This method allows us to identify individual functions and optimizations that display significant discrepancies, directing additional research and enhancement of HLS simulations and FPGA performance assessment methodologies.

V. Optimizations in High-Level Synthesis for Accelerated Hardware Design

We produced a sample space of programs with various forms of optimizations for these operations before inserting the probe and are monitoring their latencies on VITIS HLS. The reason for choosing different optimizations of the same type of code operation is that the majority of research in computer architecture and hardware-software co-design is focused on how much speed increase is gained by which method. The project's first goal is to demonstrate how significantly the numbers of simulation and hardware runs differ. This will result in more accurate research results, which, when applied to industry or hardware, will result in theoretical speedups. In this section, we present the optimizations performed on five distinct types of computations using High-Level Synthesis (HLS) techniques for accelerated hardware design. The goal of these optimizations is to improve the performance and efficiency of hardware implementations. Each computation type is optimized in multiple ways, resulting in a total of twenty codes. The optimizations are as follows:

A. Vector Addition

1) Baseline (Loop Tiling): The baseline optimization technique for vector addition involves loop tiling, which partitions

the input data into smaller tiles to minimize data movement between memory and compute units.

2) Data Streaming: The data streaming optimization technique focuses on efficient data transfer between memory and computation units, reducing the memory latency by streaming data in a continuous manner.

3) Data Streaming with Parallel Operations: This optimization combines data streaming with parallel operations to exploit parallelism at both the data transfer and computation stages, achieving higher performance and throughput.

B. Apply Watermark

1) Baseline: The baseline optimization technique for applying a watermark involves the sequential processing of image pixels to embed a 16x6 pixel watermark.

2) Loop Parallelization for 16-bit and 32-bit Data: This optimization technique parallelizes the watermark application process by utilizing multiple processing elements for both 16-bit and 32-bit data, enabling faster watermark embedding.

3) Data Streaming with Loop Parallelization for 16-bit and 32-bit Data: In this optimization, data streaming is combined with loop parallelization to achieve efficient watermark embedding for both 16-bit and 32-bit data types, reducing processing time.

C. Matrix Multiplication

1) Baseline: The baseline optimization for matrix multiplication involves the sequential processing of matrix elements using nested loops.

2) 1D Loop Tiling: This optimization technique employs loop tiling in one dimension to break down the matrix multiplication computation into smaller tiles, reducing memory access overhead.

3) 2D Loop Tiling and Array Partition: By combining 2D loop tiling and array partitioning, this optimization technique further improves memory access patterns and reduces data dependencies, leading to increased parallelism and performance.

D. Convolution

1) Baseline: The baseline optimization for convolution involves the sequential processing of input data and a filter.

2) Array Partition and Pipelining: This optimization technique leverages array partitioning and pipelining to parallelize the convolution operation and reduce the latency of data transfer.

3) Addition of Ping Pong Buffers: By incorporating ping pong buffers, this optimization reduces data transfer latency and increases the throughput of the convolution operation.

4) Optimization of Ping Pong Buffer Implementation: This optimization further refines the ping pong buffer implementation, maximizing memory utilization and minimizing data dependencies for improved performance.

E. FIR Shift Register for an Array

1) Baseline (No Array Partition, No Data Stream): The baseline optimization for the FIR shift register involves sequential processing without any array partitioning or data streaming.

2) Array Partitioning + Pipelining: This optimization technique partitions the input array and utilizes pipelining to increase parallelism and reduce latency, improving overall performance.

3) Data Streaming + Pipelining: By combining data streaming and pipelining, this optimization further enhances parallelism and reduces latency, resulting in improved performance for the FIR shift register.

To this end, we have created a table of results depicting the latencies of different codes with different types of optimizations. We believe this approach will allow us to gain insights into the sources of latency in computations and memory operations and provide guidelines for enhancing the design and optimization of computer systems.

TABLE I
OPTIMIZATIONS AND SYNTHESIS LATENCIES FOR DIFFERENT TYPES OF CODE IN VITIS HLS

Type of Code	Optimizations	C-synth	Co-sim
Vector Add	1) Baseline	1.46 us	-
	2) Only Data-streaming	1.48 us	2.44 us
	3) Data-streaming + loop unroll	2.74 us	3.7 us
Shift Register	1) Baseline	1.14 ms	1.187 ms
	2) Array partitioning + pipelining	0.819 ms	0.825 ms
	3) Data streaming	0.0414 ms	0.0467 ms
Matrix Multiplication	1) Baseline	32 ms	32.5 ms
	2) Array partition	3.29 ms	3.42 ms
	3) 1-D loop tiling	1.10 ms	1.13 ms
	4) 2-D loop tiling	0.65 ms	0.685 ms
Convolution	1) Baseline	22400 ms	-
	2) Array partitioning + Pipelining	402.2 ms	-
	3) Restructured code with pipelining	390.8 ms	-
	4) Basic Ping-Pong buffers	492.5 ms	-
	5) Optimized Ping-Pong	198 ms	-
Apply Watermark	1) Baseline	5.368 ms	5.368 ms
	2) Loop parallelization (16-bit)	0.317 ms	0.35724 ms
	3) Loop parallelization (32-bit)	0.317 ms	0.35724 ms
	4) Data streaming (16-bit)	0.317 ms	0.35724 ms
	5) Data streaming (32 bit)	0.317 ms	0.35724 ms

VI. QUANTIFYING LATENCY DISCREPANCIES: A COMPARATIVE ANALYSIS OF HLS SIMULATION AND ACTUAL HARDWARE IMPLEMENTATION

In this section, we present the findings of our research focused on comparing the latencies obtained from HLS simula-

tion and actual hardware implementation for matrix multiplication, convolution, and the "apply watermark" operations. The objective was to assess the reliability of the HLS simulation results and investigate the existence of latency discrepancies between simulation and hardware performance. The research methodology involved a systematic approach, starting with the generation of bitstreams for each optimization level of the respective codes. These bitstreams were then implemented on the n the Pynq Z-2 FPGA board to measure the actual onboard latencies. The obtained results were subsequently compared with the latencies predicted by the HLS simulation. The obtained latencies for matrix multiplication, convolution, and the "apply watermark", both from the HLS simulation and the actual hardware implementation, were meticulously recorded and analyzed. A comprehensive table was created, comparing the latency differences across various optimization levels for each code.

TABLE II
COMPARISON OF LATENCIES FOR MATRIX MULTIPLICATION IN HLS SIMULATION AND ACTUAL HARDWARE

Type of code	Latency in Vitis HLS	Onboard Latency	Difference
Unoptimized	32 ms	53.04 ms	65.75%
Array partition	3.29 ms	6.04 ms	83.58%
1-D loop tiling	1.1 ms	3.25 ms	195.18 %
2-D loop tiling	0.65 ms	1.94 ms	198.46%

TABLE III
COMPARISON OF LATENCIES FOR CONVOLUTION IN HLS SIMULATION AND ACTUAL HARDWARE

Type of code	Latency in Vitis HLS	Onboard Latency	Difference
Baseline unoptimized	22.4 s	36.37 ms	62.37%
Array partition	402.2 ms	1178 ms	192.88%
Restructured code + array partition	390.8 ms	1162.3 ms	197.41%
Addition of ping pong buffers	492.5 ms	1323 ms	168.62%
Optimized ping pong buffers	198 ms	848 ms	328.63%

The findings from our investigation, as depicted in the table and observed trends, clearly demonstrate a significant difference between the expected speedup promised by the VITIS HLS simulation and the actual onboard latency observed during hardware implementation. For instance, in the case of matrix multiplication, the HLS simulation predicted a speedup of 422x, whereas our onboard latency measurements showed a speedup of only 144x. This notable disparity highlights the need for further exploration into the accuracy and reliability of HLS simulations inaccurately predicting the onboard performance of FPGA designs. Furthermore, a general trend can be observed across all the computations analyzed. As the level of optimizations increases, aimed at reducing latency, the discrepancy between the HLS simulation latency and the actual onboard latency seems to widen. This trend raises critical questions regarding the limitations of the HLS simulation tool in accurately capturing the intricate performance nuances inherent in FPGA-based systems. Of particular interest is the significant difference observed in the case of the "apply watermark" operation when optimized. Here, the actual hardware latency was found to be nearly 16 times slower than the predicted simulation latency in the VITIS HLS tool. Such substantial discrepancies warrant deeper investigation to understand the underlying factors contributing to the observed variations.

TABLE IV
COMPARISON OF LATENCIES FOR APPLY WATERMARK IN HLS SIMULATION AND ACTUAL HARDWARE

Type of code	Latency in Vitis HLS	Onboard Latency	Difference
Baseline	5.368 ms	9.77 ms	82.00%
Loop parallelization (16-bit)	317.13 us	5.19 ms	1536.55.00%
Loop parallelization (32-bit)	317.13 us	5.48 ms	1628.02%
Data streaming (16-bit)	317.3 us	5.19 ms	1535.67%
Data streaming (32-bit)	317.3 us	5.21 ms	1541.97%

We intend to look into the causes of the disparity between the simulation and hardware results in light of these discoveries. We'll assess the effects of many variables, including the FPGA board specs, the effectiveness of the optimization techniques used in HLS, and the constraints of the simulation models used to predict the design performance. The outcomes of this investigation, in our opinion, will shed important light on the efficacy of the HLS optimization procedure and the significance of confirming the findings using actual hardware testing.

VII. INFERENCES

In this section, we analyze the latency numbers obtained from the VITIS HLS synthesis, co-simulation, and hardware runs for two different code functions: "Apply Watermark" and "Convolution." The objective is to investigate and understand the observed latency discrepancies between the simulation results and the actual hardware implementation. By examining the variation in latency across different code variations and optimizations, we aim to provide insights into the factors contributing to these differences.

A. Analysis and Findings

The analysis of latency numbers for the "Apply Watermark" code function reveals significant discrepancies between the expected latencies from the HLS simulations and the actual hardware run results. The baseline implementation yields a latency of 5.368 ms during both Csynth and co-Sim stages. However, during the hardware run, the latency increases to 9.77 ms, indicating a noticeable deviation from the simulation results. Furthermore, when applying loop parallelization and

modularized data streaming techniques with both 16-bit and 32-bit data variations, we observe a further increase in latency during the hardware run compared to the simulation results. These variations in latency range from 5.19 ms to 5.48 ms, demonstrating a substantial deviation from the simulation predictions.

The analysis of latency numbers for the "Convolution" code function also reveals significant differences between the expected latencies from the HLS simulations and the actual hardware run results. The baseline implementation exhibits a latency of 22,400 ms during Csynth, while the latency during the hardware run increases to 36,373 ms, indicating a substantial disparity. Upon implementing optimizations such as array partitioning and pipelining, restructuring the code with partitioning and pipelining, and incorporating a ping pong buffer, we observe varying degrees of latency reduction during the Synthesis and C-RTL co-simulation stages. However, during the hardware run, the latency increases compared to the simulation results, with percentage differences ranging from 168.63% to 328.64%.

B. Inference and Hypothesis

Based on our analysis, we hypothesize that the observed latency discrepancies between the HLS simulations and the actual hardware runs can be attributed to several factors, with one potential reason being the differences in memory behavior between the simulation environment and the physical hardware implementation. The simulation environment may not fully replicate the intricacies and limitations of the memory subsystem in the hardware, leading to disparities in latency predictions.

Additionally, the utilization of various optimizations and techniques, such as loop parallelization, modularized data streaming, array partitioning, pipelining, and ping-pong buffers, introduces additional complexities and potential sources of latency variations, particularly in-memory operations. These optimizations may interact differently with the memory subsystem in the hardware, resulting in deviations from the simulation results. Moreover, the presence of non-idealities specific to the FPGA hardware's memory subsystem, such as resource allocation constraints, memory access patterns, or architectural limitations, can significantly contribute to the observed latency discrepancies. These hardware-specific factors may manifest differently during the hardware run, leading to unexpected changes in memory latencies.

It is worth noting that these inferences are based on our analysis of the provided latency numbers and are subject to further investigation and validation. The proposed hypothesis paves the way for future research endeavors, which involve the insertion of probes into the FPGA design to gain deeper insights into the specific modules responsible for the observed latency discrepancies.

VIII. Future Work: Profiling and Analysis of Module Runtimes for Validating Latency Discrepancies

In light of the latency discrepancies observed between HLS simulations and actual hardware runs, further investigation and experimentation are essential to validate our hypothesis that this latency discrepancy is largely due to memory behavior not simulated accurately in the Vitis HLS tool, and gain deeper insights into the underlying factors, in this future work, we propose the insertion of probes into each module to analyze and profile their individual runtimes, by modularizing the code in primarily three modules: 'Load', 'Compute', and 'Store'.

A. Methodology

To validate our hypothesis we need a way to monitor elements within each module and evaluate their respective execution times. Our primary focus will be on the Load, Compute, and Store modules. To do so, we will employ a hardware-based method that will rely on signals and triggers(interrupts) that will be integrated into each module to correctly quantify the time of each module's activity on board and ensure independent reporting of latency.

B. Resource Allocation Considerations

Adding counters to each module to report its latency will nonetheless incur an overhead. To combat this, a comprehensive resource allocation analysis will be conducted to determine whether an additional FPGA should be utilized or if the existing FPGA can accommodate the counter modules without compromising resource allocation to functional modules.

C. Data analysis

The data collected from the probes will be analyzed to gain insights into the specific modules causing the observed latency discrepancies. By examining the runtime of each module independently, we aim to identify any disparities between the predicted latencies from the HLS simulations and the actual runtime observations on the hardware. Through detailed analysis and profiling, we aim to understand the impact of optimizations, hardware-specific non-idealities, and the memory subsystem on latency variations.

D. Significance

This future work will be critical in enhancing the accuracy of HLS simulations and FPGA performance estimation, particularly for memory operations. We will improve our understanding of the reasons causing latency variations by resolving the identified latency disparities, paving the path for more dependable and efficient FPGA designs. The inquiry of resource allocation issues and the development of a profiling methodology are significant research milestones that will help progress FPGA design methodologies.

IX. Original Execution Plan

Our project aims to investigate the discrepancy in latency numbers between simulation and hardware runs for different types of computations with various optimizations. The following execution plan outlines the steps and timeline we will follow to complete this project.

- Week 1-2 (March): The team will work together to gather a sample space of different types of computations, which

will involve collecting a set of 20 codes with five different types of computations and four different optimizations for each of them (VectorAdd, ShiftRegister, ApplyWatermark, MatrixMultiplication, and Convolution).

- Week 3 (April): The team will begin accumulating the results from co-synthesis, co-simulation, and hardware runs to simulate each computation and collect latency numbers. The hardware runs will be completed during this time, and the team will also analyze the latency numbers to determine if there is a discrepancy between the simulation and hardware run results. With the help of our advisor, Prof. Callie Hao, we will validate our data and results to ensure their accuracy and reliability.
- Week 4-5 (April): The team will devise a way to insert a probe in each module to find their run time, specifically in the Load, Compute, and Store modules. This will be used to determine which module is causing the discrepancy. The data collected from the probes will be analyzed to determine the cause of the discrepancy. After all the data has been collected and analyzed, The team will combine all results to create a clear and concise report that details the research process, data analysis, and conclusions.

X. Conclusion

In conclusion, this research study focused on investigating the accuracy of simulation latency numbers in predicting actual hardware performance for FPGA designs implemented using High-Level Synthesis (HLS) techniques. Through the evaluation of twenty codes representing various computation types and optimizations, we sought to examine the disparity between simulation results and actual hardware speed-up.

The findings of our analysis revealed significant discrepancies between the latency numbers obtained from HLS simulations and the results observed during the hardware run. Specifically, for the ”Apply Watermark” code function, we observed notable deviations in latency between simulation and hardware, indicating limitations in accurately predicting hardware performance based solely on simulations. Similar trends were observed in the ”Convolution” and ”Matrix Multiplication” code functions.

To further investigate and refine our understanding of the latency discrepancies, we propose a modular approach that involves inserting probes into the Verilog code to measure the runtimes of individual modules. This approach aims to provide insights into the specific modules causing latency variations and facilitate more accurate performance estimation.

In conclusion, this study adds to the body of knowledge on the simulation versus hardware performance discrepancy in HLS-based FPGA designs. The findings highlight the need for a more thorough and reliable method of performance estimate that incorporates simulations as well as hardware profiling. We can make more informed design decisions and optimize FPGA implementations for increased efficiency and speed by bridging the gap between simulation forecasts and actual hardware performance.

References

[1] https://github.com/Xilinx/Vitis_Accel_Examples

[2] https://docs.xilinx.com/r/en-US/ug1393-vitis-application-acceleration/Enabling-Profiling-in-Your-Application